32

Escuela Nacional de Estudios Profesionales Aragón

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

REDES NEURONALES APLICADAS A LA INGENIERIA DE CONTROL

TESIS QUE PARA OBTENER EL TITULO DE INGENIERO EN COMPUTACION

PRESENTAN :

MARCO ANTONIO QUINTAL ALVAREZ
ARTURO SOTELO MARTINEZ

1994

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

AGRADECIMIENTOS

A mi creador, por darme vida y esperanza.

Para mi Mamá, por ser al mismo tiempo la mejor madre y el mejor padre, que ha logrado formar con mucho esfuerzo a sus hijos haciendolos personas responsables y de bien, por lograr en mí a un hombre en todo el sentido de la palabra.

A mis hermanos, por tener la suficiente paciencia a mi difícil carácter y por el apoyo que me han brindado cuando más lo he necesitado.

Para la banda, que me enseñó que es imposible llegar solo a la meta.

A Mauricio y su familia, por abrirme siempre las puertas de su hogar y de su corazón.

Para mis hijos, Aline, Viris y Totochito, por marcar un nuevo rumbo en mi vida y darme la más grande de las alegrías.

A mis maestros, que me guiaron con su luz por el camino del conocimiento.

Para la Ing. Silvia Vega, por su confianza, amistad y apoyo en todo momento.

Al M. en I. Victor García Portilla, por indicarme la ruta a seguir para lograr una de mis metas y por su invaluable y desinteresada confianza en mí.

Para mi compañero de tesis, Arturo, que ha sido más que un amigo y siempre me ha demostrado su lealtad y su nobleza.

A Eva, que desde el lugar en que se encuentra, junto al creador, nunca ha dejado de tener fe en mí.

Para Claudia Angélica, por convencerme que todavía existen personas nobles, sencillas y bondadosas como lo es ella y por las cuales vale la pena vivir.

Marco Antonio.

Agradecimientos

Mientras el mundo exista, el hombre tiene que luchar por mantenerse libre de la influencia inicua de este mundo; sin embargo, nuestra lucha no es solo contra este mundo, sino, también contra nuestros propios deseos. Pero se necesita más que el estar consiente de esto, es necesario que se llegue a apreciar profundamente lo que se ha aprendido. Para que realmente se desee obrar en armonia con elló debemos tener presente: **"SIGAN, PUES, BUSCANDO PRIMERO EL REINO Y SU JUSTICIA Y TODAS ESTAS OTRAS COSA [LAS NECESIDADES MATERIALES DE LA VIDA] LES SERAN AÑADIDAS"** (mateo 6:33).

Dios. **"ALEJA DE MI LA FALSEDAD Y LA PALABRA MENTIROSA. NO ME DES NI PODRESA NI RIQUEZA. DEJAME DEVORAR EL ALIMENTO PRESCRITO PARA MI, PARA QUE NO VAYA A QUEDAR SATISFECHO Y REALMENTE TE NIEGUE Y DIGA QUIEN ES JEHOVA?. Y PARA QUE NO VENGA A PARAR EN POBREZA Y REALMENTE HURTE Y ACOMETA EL NOMBRE DE MI DIOS"** (PROVERBIOS 30:8,9).

Gracias JEHOVA por haberme permitido terminar mi carrera y tesis.

A mi Papá, por que siempre me dio lo necesario para que yo siguiera estudiado, debido a que siempre se preocupo (y aun ahora) por nuestro alimento, calzado, y estudios. A ti Papá te agradezco que hallas tenido la mano firme para que ninguno de tus hijos fueramos borrachos, ni rateros, ni mariguanos.

A mi Mamá, por que siempre supo apartar lo necesario, para que cuando hiciera falta nos comprara un par de zapatos, un sueter, o un libro. Además por que siempre estaba al pendiente de que nunca nos fueramos a la escuela sin alimento, aunque fueran las cinco de la mañana, a ella nunca le importo la hora. Y de que no se dormia cuando a alguno de mis hermanos no llegaba a casa. A ti mamá te doy las gracias de que siempre tengas unida a la familia, que solamente somos tu, mi Papá, mis hermanos y yo.

A mis hermanos Pepe, Lola y Rico, que sin ellos a lo mejor no hubiera seguido estudiando. Gracias por que sin tener la obligación de aportar dinero a la familia, lo hicieron y nunca de mala gana y sin amor. Pepe gracias por aguantarme durante todo el tiempo que estuvimos solos (**aunque no lo demuestre tu seras para mi más que un hermano**).

A mis demás hermanos Miguel, Raúl, Florencio y Tere, por que algún modo (que solo yo se) contribuyeron en la culminación de mi carrera y de este trabajo. Carnales y Jefes gracias por ser parte de mi familia.

A los cuates del Centro de Computo, que me brindaron su apoyo, su tiempo y trabajo para que se realizara esta tesis durante las horas de trabajo. A Toño, Enrique, Gabriel y Reynaldo gracias por su apoyo.

A los compañeros de estudio durante tanto tiempo, desde la secundaria hasta la culminación de mi carrera que de algún otra manera estuvieron conmigo en los tiempos más dificiles, en el hambre, desvelo, lagrimas, y alegrias que se atraviezan cuando uno estudia y madura. A Claudia, Cesar, Raymundo, Arminda, Marco, Claudio y Martin, gracias por esos tiempos.

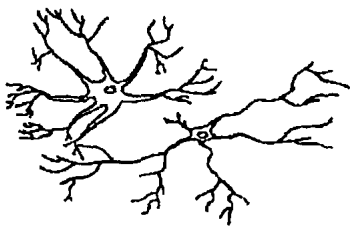
Con especial gratitud a los Ingenieros Silvia Vega Muntoy y a Juan Gastaldi Perez por haberme tenido paciencia y comprension y sobretodo por haber creido en mi a lo largo de todo este tiempo.

ARTURO SOTELO MARTINEZ

INDICE

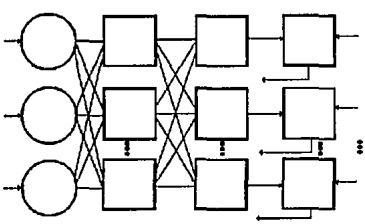
CAPITULO 1

ELEMENTOS TEORICOS Y DESAROLLO EN REDES NEURONALES



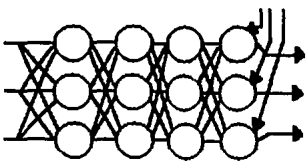
CAPITULO 2
PROGRAMACION ORIENTADA A OBJETOS: UN NUEVO ENFOQUE EN REDES NEURONALES

CAPITULO 3
IMPLEMENTACION DE UNA RED NEURONAL BASADA EN LA PROGRAMACION ORIENTADA A OBJETOS



CAPITULO 4
GENERACION DEL MEDIO AMBIENTE DE TRABAJO DE LA RED NEURONAL

CAPITULO 5
APLICACION DE LA RED NEURONAL EN UN PROYECTO DE LA INGENIERIA DE CONTROL



INTRODUCCION

Las redes neurales están inspiradas biológicamente; es decir, los investigadores al considerar la configuración y los algoritmos de una red neural, se basan principalmente en la organización del cerebro. El conocimiento acerca de la operación del cerebro está tan limitado que no es suficiente para guiar a los investigadores; estos tienen que ir más allá del conocimiento biológico, al buscar estructuras que desempeñen funciones útiles. En muchos casos, las redes neurales construidas son orgánicamente inadecuadas o requieren de un alto grado de suposiciones acerca de anatomía del cerebro y su funcionamiento.

Frecuentemente no parece existir relación entre el funcionamiento del cerebro y la construcción de las redes neurales. En un principio, tales comparaciones no eran tan buenas y crearon esperanzas irreales que inevitablemente desilusionaron a muchos investigadores, teniendo como consecuencia que en los años 60's este campo prometedor decayó en una total inactividad.

A pesar de tales advertencias, es muy provechoso el entender el funcionamiento del sistema nervioso, el cual es una entidad que desempeña exitosamente las tareas a las cuales los sistemas de inteligencia artificial tratan de imitar.

El sistema nervioso humano esta constituido de unas células llamadas neuronas, las cuales son de una complejidad sorprendente. Quizas unas 10^{11} neuronas participan en unas 10^{15} interconexiones de trayectorias de transmisión que pueden estar en un rango de un metro o más. Cada neurona comparte muchas características con las otras células en el cuerpo, pero cada una tiene capacidad para recibir, procesar y transmitir señales electroquímicas sobre todo el camino de la neurona que abarca todo el sistema de comunicación del cerebro.

El complejo sistema nervioso de los vertebrados, en especial el de los humanos, está conformado por redes perfectamente coordinadas compuestas por neuronas.

Las neuronas (como suelen llamarse a las células nerviosas) son fundamentalmente semejantes a otras células, en el sentido de que tienen cuerpo celular con citoplasma, núcleo y membrana celular.

Por lo tanto, obedecen a las mismas leyes que las demás células vivas. Pero por su adaptación a una función especial, poseen formaciones que les dan una maravillosa capacidad para conducir impulsos electroquímicos a través del sistema nervioso. Las **dendritas**, son prolongaciones especializadas en recibir estímulos provenientes de otras células, y los **axones**, a los que también se les designa **fibras nerviosas**, son prolongaciones mucho más largas y son las que llevan los impulsos hasta la neurona siguiente (Ver figura A).

La figura A muestra la estructura de una par de neuronas. Las dendritas extienden su cuerpo de célula a otra neurona donde reciben señales en un punto de conexión llamado sinapsis (la sinapsis se define como la unión funcional entre dos neuronas).

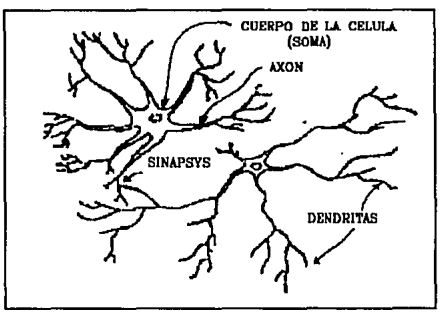
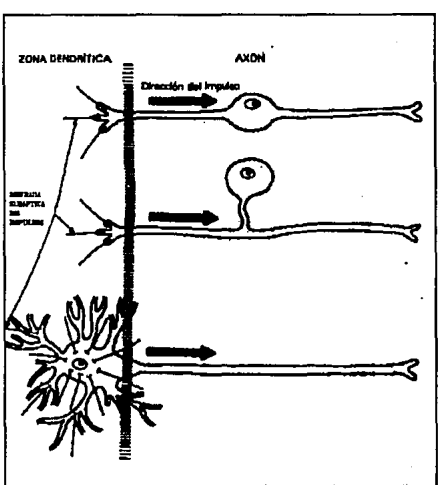


Figura A.

Las neuronas están en contacto funcional con otras neuronas, así como con células del músculo esquelético, cardiaco, liso, y con las glándulas. El contacto que se establece entre las neuronas y estas células se denomina sinapsis, término que también significa "conexión". La conexión, en realidad se establece a través de un espacio sináptico lleno de líquido extracelular que separa la membrana de la célula presináptica de la membrana celular postsináptica (Ver figura B). Este espacio estrecho por lo general mide $20x10^{-9}$mm de ancho, espacio suficiente como para interrumpir en forma súbita la transmisión de impulsos nerviosos.



La porción que recibe los estímulos de otras neuronas o de receptores, es la dendrita; la que se especializa en conducir impulsos es el axón.

La mayor parte de lo que se sabe sobre las sinapsis se basa en observaciones efectuadas en la neurona motora espinal. La sinapsis neuronal se compone de una terminación presináptica (TPS), de un espacio sináptico y de una membrana postsináptica. Con frecuencia estas sinapsis se clasifican según el lugar donde se realiza la conexión con la neurona receptora; por tanto, existen sinapsis *axodendríticas*, *axosomáticas*, y *axoaxónicas* dependiendo de si la TPS se pone en contacto con una *dendrita*, con el *soma* (cuerpo celular, también llamado pericarion) o con un *axón*, respectivamente. En términos generales, las sinapsis más comunes son las axodendritas y las axosomáticas. Con frecuencia se encuentran cientos de miles de sinapsis axodendríticas y axosomáticas en una sola neurona motora.

La llegada de un impulso a la terminación presináptica, produce la liberación del transmisor y su subsecuente difusión a través del espacio, donde activa al receptor postsináptico al abrir los canales para iones específicos (figura B). En la sinapsis excitatoria, el flujo de iones de estos canales tiende a despolarizar la membrana, mientras que las sinapsis inhibitorias diferentes modelos de flujo iónico hiperpolarizan la membrana.

En el lado que se recibe la sinapsis, las entradas están conduciendo al cuerpo de la célula(soma). Ahí son sumadas, algunas entradas tienden a excitar la célula, otras tienden a inhibir su excitación, cuando la excitación acumulativa en el cuerpo de la célula excede un umbral, la excitación de la célula transmite una señal bajo el axón a otra neurona.

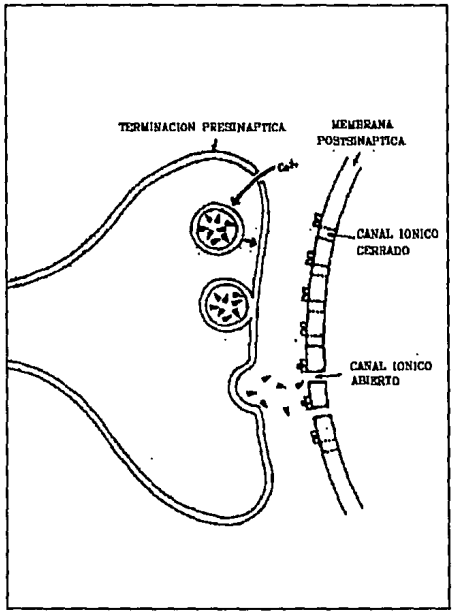
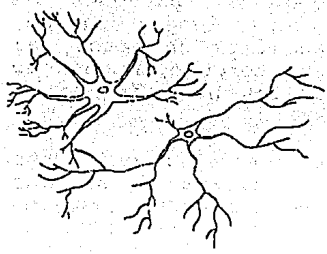


Fig B. Esquema de una sinapsis típica. La llegada de un impulso a la terminación del neurotransmisor. La corriente hace que las vesículas sinápticas emigren hacia la membrana de la terminación, uniéndose a ella las membranas vesículas, lo cual promueve la liberación de las moléculas de neurotransmisor dentro del espacio sináptico; luego los transmisores se dispersan en pocos microsegundos y entran en contacto con los sitios receptores específicos en la membrana postsinaptica, donde abren los canales iónicos. La membrana postsinápica se despolariza o se hiperpolariza, según los canales que los neurotransmisores abran.

NOTA: LA PALABRA NEURAL, SE UTILIZARA DE MANERA INDISTINTA QUE NEURONAL, DURANTE EL PRESENTE TRABAJO DE TESIS, DEBIDO A QUE PARA LA MAYORIA INVESTIGADORES SE LE ES MAS ENGORROSO UTILIZAR LA PALABRA NEURONAL; POR POR LO QUE ES VALIDO E INDIFERENTE UTILIZAR LOS TERMINOS REDES NEURONALES QUE REDES NEURALES.

CAPITULO 1



CAPITULO 1

El cambio sustancial que está ocurriendo en la Computación tiene su origen por un lado, en el empleo de computadoras de procesamiento paralelo y por otro lado, en la utilización de ciertas ideas de la Neurofisiología y la Psicología experimental. Esto ha creado un nuevo paradigma en la computación contemporánea, que se resume en la construcción de una clase de computadoras conocidas como "Redes Neuronales" o "Neurocomputadoras", que permiten resolver una cantidad inusitada de problemas. Repentinamente parece posible aplicar un cálculo previamente restringido para la inteligencia humana a un problema real para hacer que las máquinas aprendan y recuerden, con una semejanza impactante a los procesos mentales humanos.

En 1988 y 1989, se realizaron dos conferencias internacionales, en donde un grupo de personas analizaron 2000 ponencias. Profesionales de diversas areas se interesaron por el potencial ofrecido por este campo, tales como biólogos, psicólogos, ingenieros, matemáticos, psicoanalistas, filósofos, y otras ramas sociales.

Las redes neurales pueden modificar su comportamiento en respuesta a su entorno. Este es el motivo, más que cualquier otro, por el cual se ha recibido tanto interés. Dependiendo de una serie de entradas, la red neural se ajusta a éstas, produciendo respuestas consistentes. Para lograr esto se han desarrollado una gran variedad de algoritmos, cada uno con sus propias fortalezas y debilidades. Esto es lo que viene a convertir rápidamente a las redes neurales en una moda.

Muchos otros factores han ayudado a contribuir al rápido incremento de interés hacia las redes neurales, pero son tres los mas importantes: Primero, las aplicaciones en la industria en campos como el conocimiento de producción, robótica, patrones de clasificación, reconocimiento de caracteres y de lenguaje; segundo, métodos experimentales neurobiológicos y análisis de datos, al mismo tiempo, modelos de neuronas biológicas; tercero, muchas de las publicaciones acerca de las redes neurales han atraído el interés de científicos e ingenieros.

Las redes neurales han contribuido al incremento en el conocimiento del cerebro, las funciones cognocitivas y la habilidad de aplicar estas funciones en las máquinas. Debido a que las redes neurales están inspiradas biológicamente, es decir, están compuestas de elementos que desempeñan de manera análoga a las funciones más elementales de una neurona, estos elementos están organizados de modo que puedan ser relacionados a la anatomía del cerebro. A pesar de esta semejanza superficial, las redes neurales artificiales exhiben un número sorprendente de las características del cerebro. Por ejemplo, aprenden de la experiencia, es decir, generalizan de las experiencias previas, para reaccionar adecuadamente ante nuevas experiencias.

Una red neural es capaz de abstraer la esencia de un conjunto de entradas. Por ejemplo, la red neural puede estar recibiendo una secuencia de versiones distorsionadas de la letra A. Después de un entrenamiento adecuado, la aplicación de una señal distorsionada hará que la red produzca una letra perfectamente formada. Por decirlo así la red ha aprendido a producir algo que no ha visto antes.

Una vez entrenada la red, las respuestas de ésta, pueden ser de un grado menor e insensible a las variaciones de la entrada. La habilidad para ver a través del ruido y distorsión el cual es un patrón que yace dentro de la red, es vital para el reconocimiento de patrones en un entorno de tiempo real. Venciendo a la computadora convencional, se produce un sistema que puede tratar con el mundo imperfecto en que nosotros vivimos.

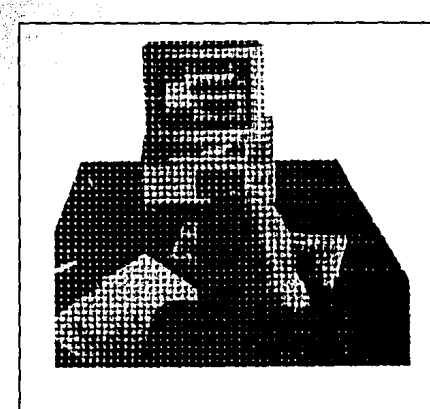
Las redes neurales artificiales no son una panacea, son inadecuadas para diversas labores, por ejemplo, en sistemas de nóminas. Sin embargo, la técnica preferida para la red es una clase de tareas de reconocimiento de patrones, las cuales las computadoras convencionales hacen pobremente. Las redes neurales se emplean para ejecutar una variedad muy complicada de trabajos que en el pasado solamente podían llevarse a cabo por número limitado de personas expertas intensamente entrenadas.

1.1

PERSPECTIVA HISTÓRICA

A la gente siempre le ha fascinado la idea de los artefactos inteligentes. El autómata jugador de ajedrez de Wolfgang Von Kempelen asombró y confundió por muchos años a la Europa del siglo XVIII. Con el tiempo quedó demostrado que se trataba de un fraude: en su interior ocultaba, con el auxilio de ingeniosos aparatos mecánicos, a un jugador humano. De hecho en esa época no existía la tecnología para crear una máquina inteligente. Incluso al principio de ese siglo, cuando se habló acerca de su robot, todavía se consideraba que la máquina inteligente era un sueño fantástico. Así como la gente se ha fascinado de las máquinas, también siempre se ha maravillado acerca de sus propios pensamientos, por que ha supuesto que es unicamente una característica propiamente humana. Esto ha motivado a investigadores como neurólogos y neuroanatomistas a estudiar el cerebro y su sistema nervioso, pero han encontrado que es muy difícil de observar su asombrosa organización; al estar trazando cuidadosamente la estructura y función del sistema nervioso humano, entendieron gran parte del alambrado de cerebro, descubrieron que cientos de billones de neuronas, cada una conectadas a cientos o miles de otras, abarcando un super sistema que empequeñece nuestros sueños más ambiciosos de supercomputadoras.

La mejor comprensión del funcionamiento de la neurona y el patrón de sus interconexiones permite a los investigadores producir modelos matemáticos para probar sus teorías. Ahora los experimentos son realizados en las computadoras sin involucrar a humanos o animales, resolviendo muchos problemas prácticos y éticos. Al verse cuan fácilmente podrían utilizarse las computadoras para resolver los problemas que les resultaban demasiado tediosos a los seres humanos, comenzó a especularse si también podrían usarse para resolver aquellos que las personas encontraban en extremo difíciles.



Los avancez de la computadora en la ciencia han sido considerables. En medicina, por ejemplo, se utiliza para intervenciones quirúrgicas.

Desde el principio de los experimentos, fue aparente que estos modelos no únicamente imitarán al cerebro, puesto que fueron capaces de perfeccionar funciones muy útiles con características propias. Por lo tanto, se definen dos objetivos de modelación neural: primero, entender el funcionamiento fisiológico y psicológico del sistema neural humano, y segundo, producir sistemas computacionales (redes neurales artificiales) que desempeñen el funcionamiento del cerebro.

La historia sobre las redes neurales aparece en los inicios de los equipos de cómputo. Las primeras ideas tienen su origen a fines de los 40's y principios de los 50's, basado en los conceptos teóricos de Alan Turing. Después de la muerte de Turing la inteligencia artificial nació en una conferencia de neurobiologos, psiquiatras, matemáticos e ingenieros reunidos en Hanover (New Hampshire, U.S.A.) en el verano de 1956. Fue el profesor John McCarthy, del colegio de Dartmounth, quien propuso el nombre de inteligencia artificial. Por un acuerdo táctico entre los asistentes, el concepto habría de referirse exclusivamente al estudio del pensamiento humano y a cómo el proceso de pensar podría representarse digitalmente con máquinas y programas.

En años recientes una gran cantidad de emociones han sido dirigidas hacia una área de la inteligencia artificial llamada redes neuronales. Un grupo de psicólogos conocen estas conexiones y están creando nuevos modelos matemáticos para mostrar cómo trabajan las neuronas en el cerebro humano; científicos en computadoras están construyendo redes neuronales que imitan la compleja actividad del cerebro.

Uno de los modelos que haya sido más fructífero, fue el de Hedd, que en 1949 propuso una ley acerca del conocimiento, el cual es el punto de partida para muchos de los actuales algoritmos de entrenamiento de redes neurales. Esta ley ha crecido por muchos otros métodos, pero demuestra a científicos de hoy en día, cómo una red neural puede realizar el proceso del aprendizaje.

Entre los años 50's y 60's, un grupo de investigadores combinaron estos conocimientos biológicos y psicológicos para producir las primeras redes neurales artificiales. Inicialmente se implemento como una serie de circuitos electrónicos, después estos se convirtieron en el medio más flexible de simulación de una computadora. Estas primeras redes producieron una gran actividad y optimismo. Marvin Minsky, Frank Rosenblatt, Bernard Widrow y otros, desarrollaron redes consistentes de un simple nivel de neuronas artificiales, llamadas PERCEPTRONES, las cuales se aplican a problemas diversos tales como predicción, análisis de electrocardiogramas, diagnóstico, planeación, interpretación, control, monitoreo de estado e interpretación. Esto parecía ser que la llave para la inteligencia había sido encontrada.

Esta ilusión pronto fue descartada. Las redes fallaron al resolver problemas que habían sido resueltos satisfactoriamente con anterioridad. Estas fallas inexplicables iniciaron un período de análisis muy intenso, no tardaron en aparecer ideas para atacar este tipo de problemas. Marvin Minsky, aplicó cuidadosamente técnicas matemáticas y desarrolló rigurosos teoremas considerando la operación de la red. Esta investigación llevó a la publicación de su libro Perceptrons (Minsky and Papert, 1969), desafortunadamente para los amantes de las "células", se demostraba que el perceptron no era capaz de resolver muchos problemas muy simples, aunque reconocía algunas virtudes que el ingenioso mecanismo poseía, y culminaba aseverando que su única utilidad práctica sería con fines didácticos.

El perceptron ha sido el punto de estudio a pesar de sus muchas limitaciones. Tiene muchas características que atraen su atención: su linealidad, su intrigable aprendizaje, y su simplicidad que la caracteriza. Pero no hay razón para suponer que cualquiera de estas virtudes alcancen a las versiones de varios niveles.

La brillantez que caracterizó a Minsky, su rigor, y su prestigio, le dio al libro mucha credibilidad. Sus conclusiones no dieron lugar a ninguna duda. Muchos investigadores se desalentaron y abandonaron el campo para áreas de mayor promesa. Las agencias de gobierno no planteaban construir tales maravillas en corto plazo y retiraron los fondos de investigación para tales máquinas; y las redes neurales cayeron en una total obsolencia por casi dos décadas. Mientras las compañías se dedicaban a áreas de mayor promesa, los pioneros en redes neurales cayeron en un dilema; al no poder contar con las máquinas diseñadas por ellos mismos, los menos persistentes decidieron capitalizar algunas ideas en el campo del software, y desarrollaron estupendas metodologías para atacar problemas que requerían de la experiencia y el razonamiento humano, siendo su gran virtud el haber advertido que existían esa clase de tareas, con las que la computadora no podría lidiar fácilmente.

Otra parte de investigadores decidió, sin embargo, persistir en su empeño de manejar sus métodos en base a otra arquitectura de computadora diferente. A pesar de existir varios enfoques, algunos investigadores empezaron a trabajar en las ideas expresadas, por McCulloch-Pitts en los 40's. La máquina descrita por ellos se considera un autómata finito, que consta de una serie de "nodos" o "células" interconectados entre sí, que poseen solamente dos estados: "inhibido" o "exaltado"; el "estado" de la célula es consecuencia de una serie de entradas que recibe, y es a su vez una señal que se transmite a las demás células.

Lo verdaderamente ingenioso del trabajo de McCulloch-Pitts es que lograron determinar que, bajo ciertas condiciones, estas células podían adquirir una forma de "memorizar" la información que habían recibido, además de una retroalimentación esencial para que la célula logre "recordar".

No obstante, algunos científicos dedicados como Teuvo Kohonen, Stephen Grossberg y James Anderson continuaron sus esfuerzos. Frecuentemente sin fondos y menospreciados, algunos investigadores con gran dificultad publicaron sus resultados durante los años 70's y principios de los 80's encontrándose esparcidos entre una variedad de diarios, algunos de los cuales están en un obscuridad total.

En años pasados, la teoría se ha estado trasladando hacia la aplicación, y nuevas corporaciones aparecieron en la comercialización apartir de esta tecnología, por lo que ha aumentado la cantidad de actividad de investigación. Con cuatro convenciones en 1987 en el campo de redes neurales artificiales, y sobre 500 documentos técnicos publicados, la razón de crecimiento es fenomenal.

La historia acerca de la ciencia es una crónica de errores y de aciertos. Indudablemente la aceptación de hechos puede invalidar la pregunta científica. Desde un punto de vista, excelentes trabajos científicos de Minsky guían hacia una desafortunada pausa en el progreso de las redes neurales.

1.2 LAS REDES NEURALES HOY

Se han realizado muchas demostraciones impresionantes de la capacidad de una red neural, por ejemplo: Una red se ha estado entrenando para transformar texto a representaciones fonéticas, (1987, Sejnowsky y Rosenberg), otra en la que una red puede reconocer caracteres escritos a mano (1987, Zumbido); y por último en que una red neural que esta basada en imágenes de comprensión, en la que todavía se encuentra en proyecto (Cottrell, Munro, y Zipser, 1987), en la figura siguiente se ve que no es muy exacto este reconocimiento, las palabras que no reconocio, las sustituye por el caracter @. Todas éstas

Un momento culminante de la historia: el hombre en la Luna
En un discurso de mayo de 1961. el Presidente J. F. Kennedy de E. U. comprometió a su país a desembarcar una nave espacial tripulada er@ la Luna "antes del fin de la década". Y se puso en marcha el proyecto Apolo.
Le tocó al Apolo ((servir de vehículo a los que realizarían la hazaña que habían soñado tantos visionarios. El 16 de julio de 1969, los aslronaulas norteamericanos Neil A. Armstrong, Edwin E. Aldrim, Jr. y Michael Collins fueron impulsados en su nave espacial por un poderoso cohete Saturno V. Armstrong y AI-.
drim pasaron al Módulo Lunar, y descendieron y desembarcaron en la Luna el 20 de julio a las 8: 17 PM, hora de @reenwich.

Red neuronal que reconoce caracteres escritos a mano.

demostraciones utilizan la retropropagación, quizás el más exitoso de los actuales algoritmos. La Retropropagación, se desarrolló por varios investigadores (1974, Werbos; 1982, Parker, Rumelhar, y Hinton, y Williams, 1986), preeviendo unos medios sistemáticos para instrucción de redes multicapas, y así venciendo las limitaciones presentadas por Minsky.

Es fácil suponer que una de las primeras aplicaciones es la simulación del cerebro humano y sus cientos de miles de millones de neuronas. Hasta la fecha sólo se han logrado simular unas 10,000, lo que no está nada mal, pero queda lejos del órgano original. En ésas simulaciones se han empleado grandes ordenadores que han necesitado horas para repetir procesos que el cerebro realiza en segundos. se sigue intentando desarrollar un modelo donde experimentar los efectos de nuevas medicinas, el comportamiento ante diferentes estímulos, etc.

Muchos de los otros algoritmos que se han desarrollado tienen cada una su ventaja específica, pero enfatizando que ninguna de las redes de hoy en día representan una panacea, todas sufren limitaciones en su habilidad para aprender y recordar.

A veces los autores tienden a publicar sus éxitos y hacer de sus fallas una pequeñez, de ese modo crean una impresión que muchas veces no son realistas. Estos autores buscan poner en marcha nuevas firmas presentando una proyección convincente de realizaciones y beneficios sustanciales; existe, aquí por lo tanto, un peligro, las redes neurales artificiales están siendo vendidas antes de tiempo, teniendo un desempeño prometedor sin la capacidad de entrega. Si esto ocurriera, el campo puede sufrir una pérdida de credibilidad, como en los años 70's.

Las redes neurales han sido propuestas para labores muy variadas desde tareas de un campo de batalla, hasta trabajos del nivel de la mente de un bebé. Las aplicaciones potenciales son aquellas donde funciones de inteligencia humana, esfuerzo y cálculo convencional ha sido definido como molesto o inadecuado.

El éxito que se ha tenido en las áreas de la "Inteligencia Artificial" ha motivado a algunos países como Japón, a anunciar recientemente el proyecto de la Sexta Generación, que está basado precisamente en la tecnología neural. Por otra parte, el gobierno americano también anunció hace poco un proyecto del Departamento de Defensa denominado DARPA, también basado en redes neurales.

Muchas grandes empresas en todo el mundo han vuelto otra vez los ojos hacia la investigación de las neuronas en robótica. Existen compañías que han desarrollado ya componentes electrónicos con procesamiento neural. Con el tiempo los investigadores lograrán delimitar las aplicaciones de esta tecnología.

La mayoría de las aplicaciones de redes neurales, están diseñadas por simulaciones en software. De hecho, el algoritmo, aunque difícil en concepto, no es demasiado complicado en implementación, y han habido aplicaciones importantes en algunos campos clásicos de la "Inteligencia Artificial". Día a día se reportan nuevas aplicaciones y ésto, aunado a la certeza de la fácil adaptación, hace que el campo sea muy atractivo para los investigadores. En el futuro, a medida que se produzcan nuevas arquitecturas de equipos que soporten de una manera más directa la ejecución de las redes neurales, es razonable esperar un desarrollo de sistemas que se aproximen asintóticamente al comportamiento humano en muchas áreas.

Antes de que redes neurales artificiales puedan ser aplicadas a la vida humana o a los objetos de valor que están en riesgo, se deben de tener en cuenta muchas cosas en cuanto a su confiabilidad. Como la estructura de cerebro humano son imitadas, las redes neurales artificiales tienen un grado de incertidumbre. A menos que la entrada sea sin distorsión, pero de ninguna forma la entrada será exacta, más que en algunas circunstancias en que esto es intolerable; por ejemplo, ¿Cuál sería una tasa de error aceptable para una red que controla un sistema de defensa del espacio?. La mayoría de la gente diría que cualquier error es intolerable; su resultado sería muerte y destrucción. Este rango de error no cambiaría si un humano se encontrara en la misma situación, podría cometer también los mismos errores.

El problema yace en que a pesar de que se tiene la esperanza de que las computadoras están absolutamente libres de cualquier error, las redes neurales algunas veces cometen errores cuando están funcionando correctamente.

Tal vez la característica más fascinante y poderosa de las redes neurales que las distingue de la mayoría de las aplicaciones tradicionales de la computación, es su capacidad para enfrentar problemas que constituyen un reto del mundo real, por medio de la aplicación de procesos que reflejan el discernimiento y la intuición humana.

1.3 NEURONA ARTIFICIAL

El cerebro humano usa billones de neuronas con complejas conexiones llamadas sinapsis; aún así cada neurona opera tranquila y lentamente en comparación a las computadoras modernas. El poder del cerebro reside en su habilidad para procesar datos de un modo paralelo, ésto es, usar más de un camino neuronal a la vez. Las computadoras modernas operan de un modo secuencial en vez de un modo paralelo. Mientras muchos investigan un camino para desarrollar una computadora con procesos paralelos, la aproximación de las redes neuronales simulan procesos en paralelo en un sistema de microcomputadora secuencial.

Una red neuronal artificial es un sistema computacional el cual consiste de un número simple de elementos (nodos) altamente interconectados, Los cuales procesan información para determinar el valor de una señal de salida basada en los valores de varias señales de entrada. Cada neurona toma una o más entradas, hacen relativamente simples los cálculos y envía los resultados a otra neurona. El truco es combinar largos números de éstas simples unidades dentro de las redes que aprende a resolver problemas reales.

Una red neuronal es, básicamente, el resultado de los intentos por reproducir mediante ordenadores el funcionamiento del cerebro humano. Nuestro órgano pensante está compuesto por miles de millones de neuronas interconectadas de forma variante y compleja.

Su peculiar forma de trabajo lo convierte en el dispositivo más eficaz para procesar la información que suministra el mundo real: cada neurona recibe impulsos procedentes de otras neuronas, que procesa individualmente dándoles un peso determinado. Después transmite la señal resultante a otras neuronas, siguiendo una configuración variante para cada caso. Las redes neurales tratan de similar este proceso en un equipo informática. El papel de las neuronas corresponden en ellas a los llamados nodo, pequeñas unidades inteligentes en capacidad para almacenar y procesar señales.

Al igual que las neuronas en el cerebro humano, cada nodo recibe unas señales que proceden del exterior de la red o de otros nodos, las procesa (*dando distinto peso a cada una*) y la suma de cada una de las entradas sirve para determinar el nivel de activación de la

neurona, en la figura 1-1 se muestra lo anterior. Aquí, un conjunto de entradas marcadas con x1, x2,..., xn, corresponden a la señal de sinápsis de una neurona. Cada señal es multiplicado por un peso W1, W2,..., Wn, antes de que sea aplicado el bloque de suma, representado como E. Cada peso corresponde a la excitación de la sinápsis de cada conexión de neuronas. El bloque de sumas, corresponde al cuerpo de células biológico, sumando todo el ancho de entradas, algebraicamente produce un salida llamado NET. Esta se representa en notación vectorial como sigue:

$$NET = XW$$

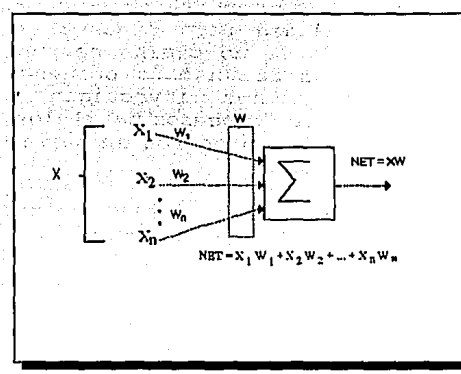


Fig 1.1 Neurona artificial

Desde el punto de vista práctico, un nodo debe poseer los medios para almacenar señales y procesarlas según peso o funciones. Por ello, los nodos deben ser circuitos electrónicos, ordenadores o fragmentos de programas. Para simplificar el diseño de las redes, los especialistas realizan topologías sencillas de distribución de nodos, disponiéndolos en capas sucesivas. Una de las más empleadas los sitúa en tres niveles (entrada, intermedio y salida) en los que cada nodo actúa con las señales de la misma forma.

En los inicios de la revolución tecnológica, cuando se querían obtener diferentes señales de salida de un sistema, dependiendo de los tipos y niveles de estímulo presentes en su entrada, era necesario dotarle de una relación matemática entre dichas entradas y salidas. Con la aparición de las redes neuronales, el sistema se autorregula, deduciendo los pesos que deba dar a las distintas señales y la forma de conectar los nodos. Tras esta regulación, puede trabaja en situaciones para las que aún no se hayan establecido reglas de funcionamiento. Una red neuronal nueva

establece las conexiones de cada nodo de forma convencional. Su proceso de aprendizaje empieza al darle una pareja de datos de entrada y salida. La red va haciendo pruebas mediante la determinación de los pesos mas convenientes y de las conexiones entre nodos. Al cabo de varios intentos, con diferentes parejas de datos de entrada y salida conocidos, el sistema está ya educado, es decir, en condiciones de trabajo la información que almacena una red se halla dispersa por todos sus nodos, lo que le confiere características distintas a las de un ordenador convencional y le hace menos propensa a las fallas.

1.4 FUNCIÓN DE ACTIVACIÓN

Esta función de umbral es generalmente igual a una función no lineal. Una función no lineal simple que es apropiada para redes neurales discretas es una función de paso (Figura 1.2). Una variante de la función de paso es:

1 si x>0

f´(x) si x 0, donde f´(x) se refiere al valor previo de f(x) (es decir, la activación de la neurona no cambiará).

1 si fx< 0

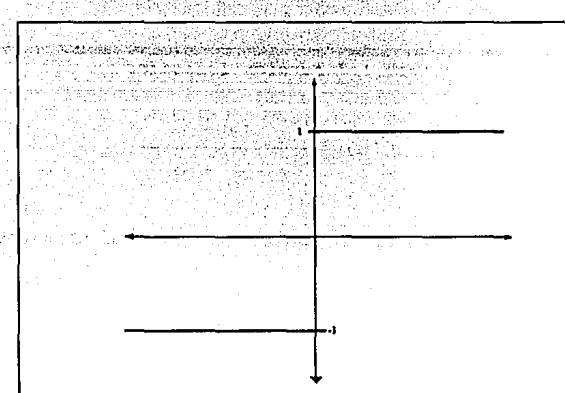


Fig. 1.2 Función de paso

Donde x es la suma del producto de la activación de la neurona entrante y el peso de la conexión sináptica:

$$x = \sum_{i=o}^{n} A_i W_i$$

Donde n es la cantidad de entrante de neuronas, A es el vector entrante de neuronas, y w es el vector de peso, conectando las neuronas entrantes a la neurona que se esta examinando. Otra clase popular de función, más apropiada a redes analógicas, es la función sigmoidal, o de encuadramiento. Un ejemplo es la función logística que se ilustra en Figura 1.3:

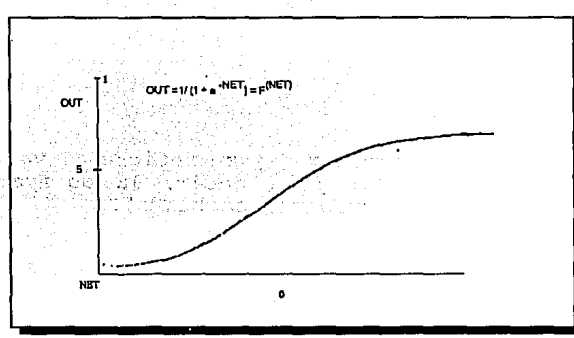


Fig 1.3 función logistica

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

Otras funciones sigmoidales están disponibles. Otra alternativa común es:

$$f(x) = \tanh(x)$$

La característica más importante de nuestra función de activación es que no es lineal. Si deseáramos utilizar una función de activación en una red multinivel, la función de activación tiene que ser no lineal, o el poder computacional será equivalente a una red única-capa.

1.5 REPRESENTACIÓN DE UNA RED REURONAL ARTIFICIAL

La figura 1-4 muestra que una red multinivel, consiste de un conjunto de neuronas y pesos, el nivel de entrada no es una sumatoría, estas neuronas sirven únicamente como un punto de ventilación hacia el primer conjunto de pesos y no afecta la capacidad de cálculo de la red. También los pesos de un nivel son asumidos para estar asociados con las neuronas que siguen a ellas; por lo tanto un nivel consiste en un conjunto de pesos y la posterior neurona que le suma a las señales de su correo.

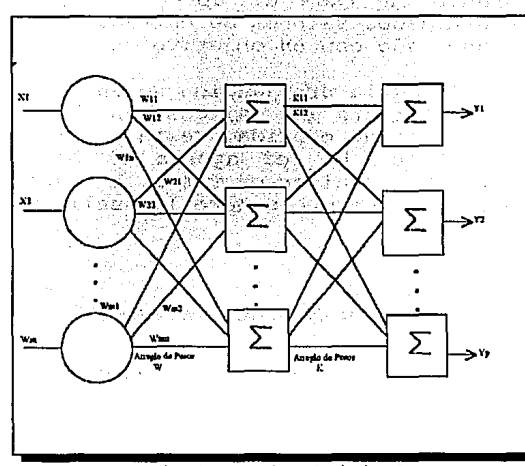


Fig 1.4 red multinivel

1.6 DESCRIPCIÓN DE UNA RED

Nuestro objetivo es crear una propagación de redes constando de neuronas organizadas dentro de capas. Cada neurona en una capa conectada con cada neurona en la próxima capa.

Cada neurona tiene una característica numérica llamada una activación igual a la suma de todos los pesos de entrada seguido a través de la capa de neuronas. Cada neurona mantiene su peso de entrada a través de una función de activación para determinar su salida.

Al colocar sus valores de entrada en la primera capa empieza el proceso. Los valores se mueven desde la capa uno a la próxima vía describiendo previamente las conexiones y operaciones. Estas son llamadas propagación, la cantidad total de la última capa determina la salida de la red, generalmente una clasificación de entrada coloca una de varias categorías.

Durante la fase de aprendizaje, la red recibe un número de determinados valores de entrada. Cada juego de valores de entrada es asociado con un objetivo de salida.

Después de la propagación de entrada a través de todas las capas, la red compara las salidas con el objetivo de salida y calcula la diferencia de cantidad, esta cantidad también es llamada "error". Finalmente, la red ajusta el peso de cada conexión según su contribución al error total. La meta de la fase de aprendizaje es tener una red que genere las salidas correctas.

1.7 DESARROLLO DE LAS REDES NEURALES

Software.

El software esta frecuente disponible para microcomputadoras con el fin de poder simular redes neuronales.

Al igual que en los sistemas expertos, hoy las redes neuronales están usualmente construidas dentro del ambiente aplicado al desarrollo del software.

Semejante a los diseños de un sistema experto los cuales permiten crear sistemas que usan una variedad de procesos de razonamiento, alguna red neuronal desarrolla herramientas que permiten crear sistemas en una variedad de modelos. Muchas herramientas contienen capacidad de gráficos que ilustran la actividad neuronal.

Hardware.

Una red bien ajustada, al igual que una persona bien formada, tiene los valores correctos. Es de imaginarse, que todos los ajustes y reajustes llegan a éstos valores vinculando una gran cantidad de procesos numéricos. Debido a que ésto puede tomar mucho tiempo, muchas redes trabajan con tarjetas aceleradoras para software específico.

Las tarjetas aceleradoras no son la única clase de hardware de una red neuronal. Inventos más elaborados, llamados "neurocomputers", son coprocesadores cuya arquitectura está armada para construir y operar redes neuronales.

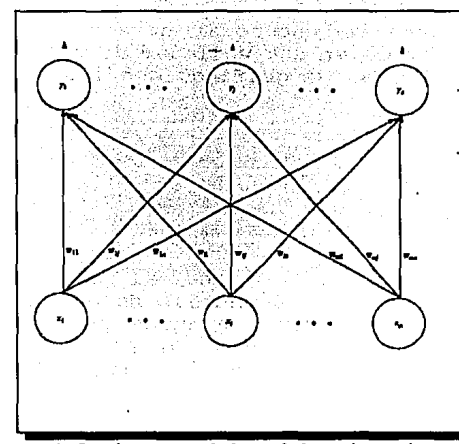
Aún más elaborados, son computadoras construidas para trabajar como redes neuronales. Este tipo de máquinas usa una gran cantidad de pequeñas interconexiones procesando unidades (lo contrario de un CPU). Porque éste procesamiento de unidades trabaja en paralelo, ésta configuración es llamada un proceso en paralelo. La mejor conocida es "Thinkin Machines Corporation's Conection Machine", una computadora programable en LISP, la cual es tan conveniente para tareas de bases de datos.

Con Software y Hardware relativamente barato, los experimentadores pueden simular una pequeña parte del cerebro humano en computadoras. Un paquete comercial de redes "the Neural Works Explorer" desarrollado por Neural Ware, fue usado para desarrollar una red neuronal.

1.8 APRENDIZAJE

Todo el "conocimiento" que una red neural posee está almacenado en la "synápsis" (los pesos de las conexiones entre las neuronas) (Figura 1.5). La sinápsis entre dos capas de neuronas se representa como matrices.

Una vez que el conocimiento se presenta en los pesos sinápticos de la red, presentando un patrón de entrada a la red producirá la salida correcta. Sin embargo, ¿cómo adquiere la red el conocimiento?. Este ocurre durante el "entrenamiento". Los patrones de asociación se presentan en la red consecutivamente, y los pesos ajustan este conocimiento como la regla de aprendizaje.



1.5 Diagrama del modelo sinaptico

Una de las primeras reglas de aprendizaje formuladas fueron por Hebbian. Donald Hebb, en su Organización de la Conducta [Hebb 49] formuló el concepto de "aprendizaje de correlación". Está es la idea donde el peso de una conexión es ajustada basada en los valores de las conexiones de las neuronas:

$$\Delta W_{ij} = \alpha a_i a_j$$

Donde α es la tasa de aprendizaje, a_i es la activación de la i-esima neurona en una capa de la neurona, a_j es la activación de la j-esima neurona en otra capa, y w_{ij} es el peso de conexión entre las dos neuronas. Una variante de esta regla de aprendizaje es la regla de Hebbian:

$$\Delta W_{ij} = -W_{ij} + S(a_i)S(a_j)$$

Donde S es una función sigmoidal.

1.8.1 [redacted]

APRENDIZAJE NO SUPERVISADO

Desde el método de aprendizaje descrito anteriormente, no prueba que los pesos resultantes otorgarán salidas aceptables, este método se describe como un método de aprendizaje no supervisado . En general, en un método de aprendizaje no supervisado, los pesos ajustados no están basados en comparación con alguna salida deseada. No hay "señal de enseñanza" en el peso ajustado. Esta propiedad es también conocida como autoorganización.

1.8.2 [redacted]

APRENDIZAJE SUPERVISADO

En muchos modelos de aprendizaje se toma la forma de entrenamiento supervisado. Tomando patrones de entrada de la red neuronal y observando al patrón de salida en comparación con nuestro resultado deseado. Entonces se necesita algún modo de ajustar los pesos que tienen en cuenta cualquier error en el patrón de salida. Un ejemplo de una ley de aprendizaje supervisado es la regla de corrección de errores:

$$\Delta Wij = \alpha ai[c_j - b_j]$$

Donde α es de nuevo la tasa de aprendizaje, a_i es la activación de la i-ésima neurona y b_j es la activación de la j-ésima neurona en el patrón recordado, y el c_j es la activación deseada de la j-ésima neurona.

1.8.3 [redacted]

APRENDIZAJE REFORZADO

Otro método de aprendizaje, conocido como refuerzo de aprendizaje se encuentra dentro de la categoría del aprendizaje supervisado, pero esta fórmula difiere desde la corrección de errores. Este tipo de aprendizaje es similar al aprendizaje supervisado, excepto que cada neurona de salida obtiene un valor de error, y solamente un valor de error está calculado por cada salida de la neurona. La formula del ajuste de pesos es entonces:

$$\Delta Wij = \alpha(v - \Theta_j) e_{ij}$$

Donde α es de nuevo la tasa de aprendizaje, Δ es el valor único indicando el error total del patrón de salida, y Θ_j es valor del umbral para la j-ésima neurona de salida. necesitamos esparcir este error para la j-ésima neurona de salida a cada neurona entrante. e_{ij} es un valor que representa el peso a actualizar. Este puede ser calculado como:

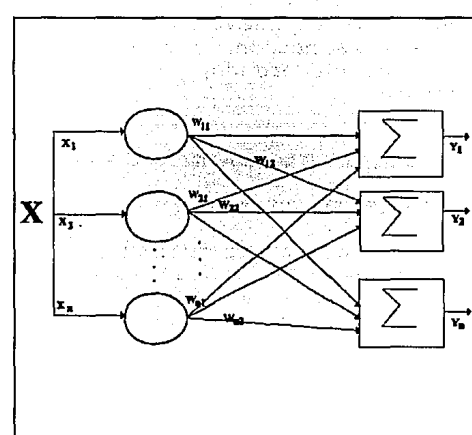
$$e_{ij} = \frac{d\ln(g_i)}{dW_{ij}}$$

Donde g_i es la probabilidad de que la salida este correcta dada la entrada de la i-ésima neurona entrante.

Sin embargo, el algorítmo de corrección de error puede estar observado como justamente un ejemplo específico de este aprendizaje de reforzamiento, donde el "esparcimiento" del error sobre la neurona entrante i está hecho por la activación de la neurona entrante misma. En lugar de utilizar más un error global, la regla de corrección de error utiliza el error específico en la j-ésima neurona de salida.

1.9 REDES NEURALES DE UN SOLO NIVEL

Las redes neurales de un solo nivel es un sistema altamente interconectado de procesadores sencillos llamados "neuronas", que ejecutan cálculo paralelo sobre señales complejas que simulan nodos agrupados en dos o más capas y conectadas en forma de red (que se muestra en la figura 1.6). Cada neurona consta de cuatro elementos principales: (1) conexiones de entrada, a través de las cuales recibe los impulsos que le envían las neuronas conectadas a ella; (2) una función acumuladora, generalmente,aditiva, mediante



1.6 Red neuronal de un simple nivel

la cual se combinan todas las señales recibidas por la neurona para calcular su estado de activación; (3) una función de umbral o activación, de preferencia no lineal, con la cual se convierte el estado de activación de una neurona en una señal de salida y (4) una serie de conexiones de salida a las neuronas con que está conectada. Las neuronas de las capas inicial y final son casos especiales, las primeras que reciben como entrada la información del exterior, normalmente un número para cada neurona, el cual constituye su estado de activación y, las segundas, porque sus salidas son los elementos del resultado.

Las conexiones entre las neuronas también son variables cuyo valor, o carga de la conexión, representa la relación entre dos neuronas. Si la carga de una conexión es positiva la neurona que transmite a través de ella estimula a la receptora y, si es negativa la inhibe. Una carga igual a cero representa la ausencia de relación entre las neuronas.

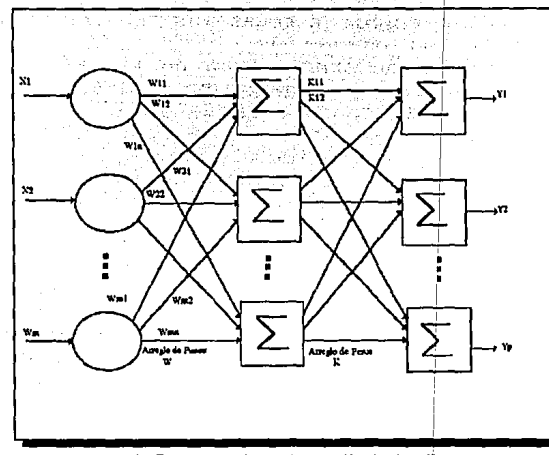
El conjunto de entradas X tiene cada uno de estos elementos que son conectados a una neurona a través de un peso; además, puede ser conectada entre las entradas y las salidas de neuronas en un mismo nivel.

Es conveniente al considerar los pesos por ser elementos de una matriz W con dimensiones de m filas por n columnas, donde m es el número de entradas y n el número de las neuronas. Por ejemplo, el peso conectado a la tercer entrada de la segunda neurona sería $w_{3,2}$. De este modo se ve que para calcular el conjunto de neuronas de salida N para un nivel es una simple multiplicación de matrices, de esta manera N=XW, donde N y X son vectores.

1.10 REDES NEURALES MULTINIVEL

Las más complejas redes generalmente ofrecen grandes capacidades de cálculo, aunque pueden construirse con una inimaginable configuración, imitando a las neuronas en niveles similares a ciertas porciones de la estructura del cerebro. Las redes multinivel se pueden formar con una simple cascada de simples redes de un solo nivel; la salida de un nivel provee la entrada al subsecuente nivel. La figura 1-7 muestra tal red. Estas redes tienden a proveer más allá de las capacidades de las de un sólo nivel, y en recientes años, se han desarrollado algoritmos para lograr esto.

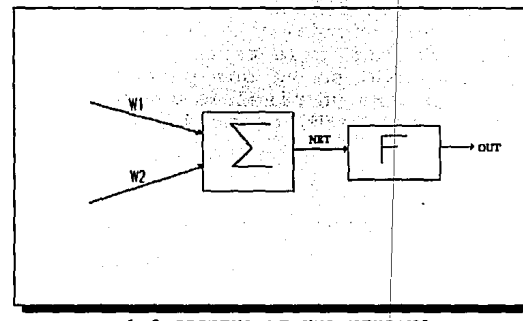
Anteriormente se discutió como dos niveles de perceptrones son limitadas en su habilidad para mapear funciones no lineales entre patrones de entrada y salida. Un ejemplo muy simple de esto es el problema del OR-exclusivo.



1.7 RED NEURONAL MULTINIVEL

El problema del or-exclusivo

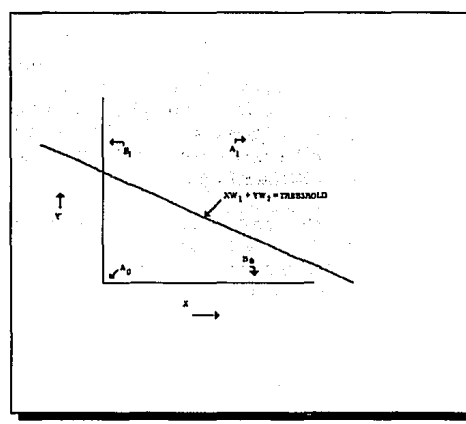
Los resultados de Minsky, muestra que un perceptrón de un sólo nivel, no puede simular una función OR-EXCLUSIVO. Esta función acepta 2 entradas que solo pueden ser cero o uno. Esto produce una salida de 1, únicamente si alguna de las salidas es uno, pero no ambas. El problema se muestra considerando un sólo nivel, un sistema de una sola neurona con 2 entradas como se muestra en la figura 1-8. Con una entrada llamada X y la otra Y, todas sus posibles combinaciones caen dentro de una área encerrada por 4 puntos en el plano X-Y, como se muestra en la figura 1-9. Por ejemplo, los puntos X=0 y Y=0, son etiquetados como el punto A0 en la figura. La tabla 1-1, muestra la relación deseada entre entradas y salidas, donde las combinaciones de entrada que deben producir una salida cero, son etiquetadas como A0 y A1; y aquellos que producen un 1 etiquetadas como B0, B1.



1.8 SISTEMA DE UNA NEURONA

TABLA 1-1 TABLA DE VERDAD DEL OR-EXCLUSIVO

PUNTO	VALORVALOR X	Y	SALIDA DESEADO
A0	0	0	0
B0	1	0	1
B1	0	1	1
A1	1	1	0



1.9 PROBLEMA DEL OR-EXCLUSIVO

En la red de la figura 1-8, La función F es un simple umbral que produce un cero para OUT cuando NET esta por debajo de 0.5 y un 1 cuando es igual o arriba de 0.5. La neurona entonces realiza el siguiente cálculo:

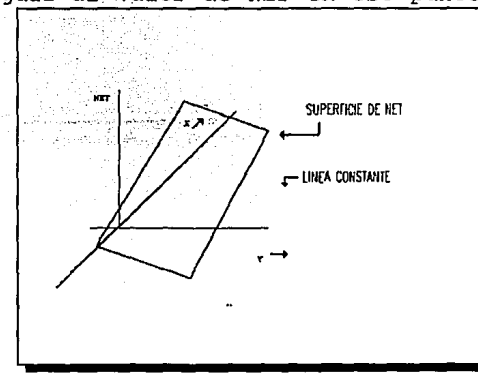
$$NET=xw_1+yw_2 \qquad (1-1)$$

Ninguna combinación de valores para los 2 pesos, w_1 y w_2, producirán una relación entrada-salida como en la tabla 1-1. Para entender esta limitante considere que NET se mantiene constante si el valor de la entrada es 0.5. La ecuación 1-2, describe la red en este caso. Esta ecuación es lineal en x y en y; es decir, todos los valores de X y Y, que satisfagan esta ecuación caen en alguna línea recta en el plano x-y.

$$XW_1 + YW_1 = 0.5 \qquad (1\text{-}2)$$

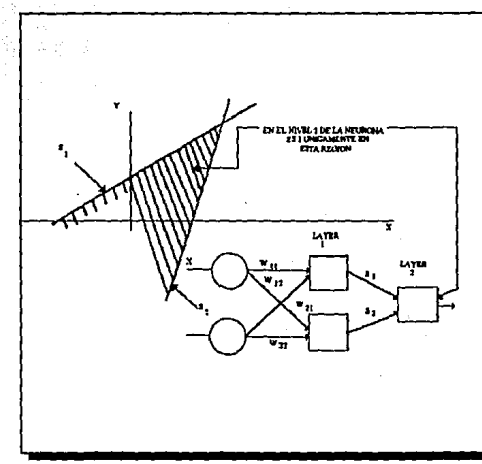
Cualquier valor de entrada para X y Y en esta línea producirá el valor de umbral de 0.5 para NET. Los valores de entrada de un lado de la línea producirán un valor de NET más grande de que el de umbral. Por lo tanto OUT=1; los valores del otro lado de la línea producirán un valor de NET, menor que el de la entrada, habiendo OUT=0. Cambiando los valores para W1 y W2 y el de la entrada, cambiará la pendiente y la posición de la línea. Para una red que produzca la función OR-EXCLUSIVA de la tabla 1-1, es necesario poner la línea tal que todas las A's están de un lado y todas las B's del otro lado. Tratar de dibujar una línea como la figura 1-9, esto no se puede realizar. Esto significa que no importa que valores sean asignados a los pesos y a la entrada, esta red es incapaz de producir la relación entrada-salida requerida para representar la función OR-EXCLUSIVA.

Analizando el problema desde una otro punto de vista, NET es una superficie que flota arriba del plano X-Y. Cada punto de esa superficie esta directamente arriba del punto correspondiente en el plano X-Y por una distancia igual al valor de NET en ese punto. Esto muestra que la pendiente de esa superficie de NET es constante sobre todo el plano X-Y. Todos los puntos que producen un valor de NET igual al valor de entrada se proyectarán por encima de un nivel constante en el plano NET (ver figura 1-10). Todos los puntos de un lado de la línea de entrada se proyectarán por encima de los valores de NET más grandes que de la entrada y los puntos del otro lado deben resultar por debajo de los valores de NET. así, la línea de entrada subdivide el plano X-Y en 2



1.10 Plano del perceptron

regiones. Todos los puntos de un lado de la línea producirán un 1 para OUT y todos los puntos del otro lado producirán un cero.



1.11 Red neuronal con una capa adicional

Sí construyéramos una red neural con un nivel adicional de neuronas entre las capas de entrada y salida, podemos lograr el mismo resultado (Figura 1.11). Nos referimos a este nivel adicional como un "nivel oculto" ya que no interactúa directamente con el nivel exterior. Las neuronas que tienen contacto con el mundo exterior es el nivel de entrada, y las neuronas que estan conectadas los niveles ocultos de neuronas y el mundo exterior se les llaman niveles de salida.

Podemos ahora postular un conjunto de pesos para la synápsis entre la entrada y las capas ocultadas y entre las capas de salida y ocultadas que permitirá a la red emular el comportamiento de la función X-OR. Los pesos entre las capas ocultas de neuronas y capa de salida de neuronas ambas son +1 (hay dos synápsis que están entre el nivel de salida y el nivel de las ocultadas y estas son 1 positivo). Estos pesos son tal que cuando sea tanto Ceros como Unos son aplicadcs, el nivel oculto de neuronas se hace 0. Si (1,0) ó (0,1) patrones de entrada están aplicados, uno de los niveles ocultos de neuronas se hace 1, que es suficientemente para establecer el nivel oculto de neuronas a 1.

Veamos este proceso con mas detalle. Supongamos que si utilizamos la siguiente función de activación para nuestro nivel oculto de neuronas:

$$f(x) = \begin{cases} 1 \text{ si } x > 0 \\ 0 \text{ si } x \leq 0 \end{cases}$$

Donde x es la suma de las activaciones de los niveles de entrada de neuronas cada una multiplicada por el peso synáptico, como sigue:

$$x = \sum_{i=0}^{n} A_i W_i$$

Donde n es la cantidad de entrante de neuronas, A es el vector entrante de neuronas, y W es el vector de pesos sinápticos conectando las neuronas entrantes a las que se están examinando.

Dada esta función de activación, vamos a examinar que es lo que ocurre cuando las entradas de las neuronas son 0 y 0 (viendo la tabla 1.1). La suma del producto de entradas y pesos en el primer nivel ocultó de neuronas tiene 0 ya que cada activación tiene 0. Así, x para la función f(x) tiene 0. Así, basada en f (x), la salida de la segunda neurona tiene 0. Ya que cada nivel oculto de neuronas es 0, la entrada para la función de activación es cero, y de modo que la neurona es 0. Así, para la entrada de (0,0), la salida se ajusta a la función XOR.

Si la entrada es (0,1), entonces el argumento para la función de activación del primer nivel oculto de neuronas es

$$(0 \times 1)+(1 \times -1) = -1$$

de modo que el primer nivel oculto de neuronas de activación tiene f(-1) = 0. El segundo nivel ocultó de neuronas tiene la siguiente expresión:

$$(0 \times -1)+(1 \times 1) = 1$$

de modo que el segundo nivel oculto de neuronas de activación es f(1) = 1. La entrada a la neurona de salida es:

$$(0 \times 1) + (1 \times 1) = 1$$

de modo que su activación es f(1), o 1. Así, la entrada de (0,1) el resultado es también correcto para la función XOR. Desde los valores de peso para cada nivel de sinápsis son simétricos, la salida para una entrada de (1,0) tiene también 1.

Si la entrada es (1,1), la entrada para cada nivel de neuronas ocultas tiene 0, desde los pesos entrantes positivos y negativos cancelan unos a otros. La activación de cada nivel de neuronas ocultas tiene por lo tanto 0, y así la entrada del nivel de neuronas de salida y activación tienen 0. Para todas las entradas

posibles, las salidas ajustan el comportamiento esperado de la función XOR. Así, dados los pesos correctos, esta red puede emular las funciones OR exclusivo. En general, cualquier red con un nivel oculto puede aprender a mapear funciones no lineales.

1.11 REDES RECURRENTES

Las redes consideradas hasta este momento, no tienen conexiones retroalimentadas, es decir, conexiones a través de los pesos extendiéndose desde las salidas de un nivel hasta las entradas del mismo. Esta clase especial de redes, son las llamadas no recurrentes o redes retroalimentadas, son de considerable interés y son extensamente aplicadas.

En algunas configuraciones, las redes recurrentes retroalimentan sus salidas; por lo tanto, su salida se determina por su entrada actual y sus salidas previas. Por esta razón las redes recurrentes pueden exhibir propiedades muy similares a las de la memoria de un humano, donde el estado de las salidas de la red dependen en parte de sus entradas previas.

1.12 ENTRENAMIENTO DE UNA RED NEURAL.

Una característica muy importante de una Red neuronal es que no se almacena el conocimiento adquirido en localidades de memoria (como en el caso del CPU); más bien, éste conocimiento se encuentra contenido dentro de la misma arquitectura de la red. El desarrollo de redes neuronales por medio del hardware, se logra a través de reforzamientos analógicos de las señales que van de un nodo a otro. En simulaciones de la red por medio de software utiliza, una localidad de memoria, aunque la idea es, en principio, completamente diferente al concepto de almacenamiento.

La estructura de una Red Neuronal se encuentra definida por: la interconexión entre sus nodos; es decir, reglas que determinan qué tipo de respuesta deberá proporcionar un cierto nodo (función de transferencia), y por ultimo, las reglas que determinan los cambios que deben sufrir las interconexiones de unos nodos con otros (lo que se conoce como entrenamiento de la red).

En una red neuronal, el programador no especifica un algoritmo a ser resuelto; en lugar de ello, se indican las interconexiones, función de transferencia, y leyes de entrenamiento de la red. El entrenamiento se declara en una primera fase, en la cual la red "aprende" a dar una respuesta a diferentes estímulos externos. Después, en una segunda fase, a la red le son presentados estímulos ya conocidos o desconocidos por ella, y se observa su respuesta. Si ésta respuesta no es muy acertada, entonces la red requiere más entrenamiento; si por el contrario, la respuesta se considera aceptable.

De todo el interés característico de una red Neural, lo mejor es su habilidad de aprender; el entrenamiento de la red muestra muchos paralelos en el desarrollo intelectual del ser humano.

Una red es entrenada para que en la aplicación de un conjunto de entradas produzca de un conjunto de salidas las deseadas. Cada conjunto de entradas o salidas es referenciado como un vector. El entrenamiento se acopla por una aplicación secuencial de vectores de entrada, y se ajusta el peso de la red de acuerdo con un procedimiento predeterminado.

Durante el entrenamiento, los pesos de la red convergen gradualmente hacia los valores que cada vector de entrada produciendo un vector de salida deseada.

1.13 ENTRENAMIENTO DE ALGORITMOS.

La mayoría de los algoritmos de entrenamiento de hoy en día tienen como base el concepto de D.O. Hebb (1961); en el que la fortaleza synáptica (peso) aumenta si el destino y origen de la neurona son activados. De este modo las trayectorias en la red se fortalecen, y los fenómenos de habito y aprendizaje a través de la repetición son definidos.

En una red neural al utilizarse el aprendizaje de Hebb se incrementan los pesos de la red teniendo en cuenta los niveles de excitación desde el origen de una neurona al destino de otra, representándose:

$$w_{ij}\ (n+1) = w_{ij}(n) + \alpha\ OUT_i OUT_j$$

donde

$w_{ij}(n)$ = el valor del peso de una neurona i, a una neurona j antes de ser ajustada.

$w_{ij}\ (n+1)$ = el valor del peso de un neuronal i a una neurona j, después de ser ajustadas

α = coeficiente del radio del aprendizaje.

OUT_i = salida de la neurona i y entrada de la neurona j

OUT_j = salida de la neurona j.

Las redes más recientes han sido construidas utilizando el aprendizaje de Hebb, sin embargo, los algoritmos de entrenamiento más efectivos han sido desarrollados en los pasados 20 años, produciendo redes que aprenden con una amplia cantidad de patrones de entrada, además de una alta cantidad de aprendizaje.

1.14 MODELOS NEURONALES

Se presentarán varios modelos neuronales en su forma clásica. Esto no quiere decir que los algoritmos y topologías no puedan ser modificados. no obstante el intento es presentar una sección razonable de los tipos de modelos disponibles. Por ejemplo, el perceptron (aunque históricamente significativo y discutido anteriormente) ha limitado su aplicación por razones que ha estado discutido, y que se presentarán aquí.

Como se plantea, se intentará presentar un modelo en su forma clásica, pero ésta no deberá ser asumida como una restricción o como una técnica que puede ser aplicada siempre. Por ejemplo, la retropropagación tiene su aplicación mucho más allá de los modelos "clásicos" de propagación que estamos a punto presentar.

Las aplicaciones en que pueden usarse las redes tienen el propósito de dar un sentido de como los modelos podrían estar modificados (o hasta combinados) en caso de haber una aplicación específica. Por ejemplo, el modelo de contrapropagación es una combinación de dos modelos existentes.

La retropropagación muestra una red multicapa con aprendizaje supervisado. En su forma canónica es una de las redes más populares en aplicaciones. Antepropagación contiene un método de aprendizaje no supervisado, que para muchas aplicaciones es más práctica que la retropropagación, pues incorpora dos redes neuronales más simples.

Estos modelos son derivados de una clase de red neural abstracta. Contienen métodos de entrada-salida para almacenar y recuperar pesos para utilizar el entrenamiento progresivo. Con una codificación abstracta, "recuerda" métodos, que están basados en información previa, presentado patrones de entrada y salida cuyo tamaño tiene que estar especificado.

Por cada modelo se presentarán: la topología (la arquitectura de los niveles de neuronas y su conexión synáptica para el modelo), el algoritmo y una discusión de los puntos más importantes del modelo. Al describir el algoritmo, se empleará una notación de vector, los cuales se representarán por matrices y el código será escrito utilizando vectores y matrices aritméticas

1.14.1 [redacted]

PERCEPTRONES

Al hacer su aparición las redes Neuronales en los años 40's, los investigadores buscaron duplicar las funciones del cerebro humano al desarrollar un modelo de una neurona biológica y su sistema de interconexión, Aunque estos primeros intentos se vieron a groso modo, lograron un gran empuje para que las más sofisticadas redes se realizarán.

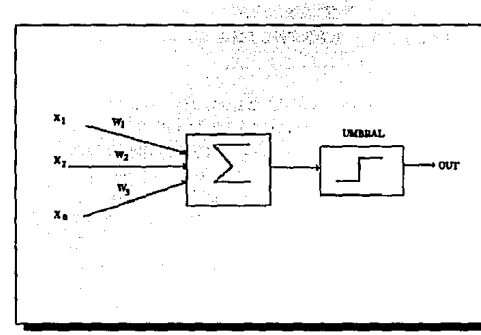
Mccullach y Pitts (1943) publicaron el primer estudio sistemático sobre redes neuronales, en un trabajo posterior (1947), exploraron los paradigmas de red para el reconocimiento de patrones, basándose en un modelo de una neurona mostrado en la figura 1-12, donde el parámetro Σ multiplica cada entrada X por un peso W, y suma los pesos de las entradas. Sí esta suma es más grande que un umbral predeterminado, la salida es 1; en otro caso será cero. Estos sistemas colectivamente son llamados **PERCEPTRONES**. En general, éstos consisten de un sólo nivel de neuronas artificiales asociados por sus pesos a un conjunto de entradas como se muestra en la figura 1-13 (aúnque es más complicada, pero lleva el mismo nombre).

En los años 60's. Los perceptrones crearon una gran división de interés y optimismo. Se realizaron un gran número de demostraciones convincentes a cerca de los perceptrones, e incluso investigadores de todo el mundo afanosamente exploraron el potencial de estos sistemas. Pero pronto esta euforía se vio reemplazada por desilusiones debido a que en los perceptrones se encontraron fallas en tareas muy simples de aprendizaje. Minsky y Papert en 1965 analizaron este problema y encontraron que hay restricciones en que un perceptrón de un solo nivel se puede representar, y por ende en lo que puede aprender. Puesto que no había técnicas conocidas en ese tiempo para el entrenamiento de redes multinivel, los investigadores desilusionados se enfocarón hacia áreas más provisorias, y la investigación de redes neuronales se vio en un terrible estancamiento.

Pero a pesar de las limitaciones de los Perceptrones estos han sido extensivamente estudiados. Su teoría es el fundamento para muchas otras redes neuronales.

1.14.1.1 REPRESENTACIÓN DE UN PERCEPTRON.

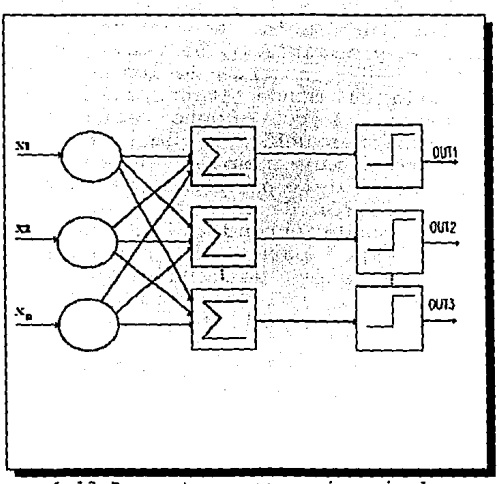
La prueba del teorema de aprendizaje del perceptron (Rosenblatt 1962) demostró que un perceptron puede aprender cualquier cosa si esta puede ser representada (fig 1.13). Es importante distinguir entre la representación y aprendizaje, la representación se refiere a la habilidad de un Perceptron (u otra red) para simular una función especifica. El aprendizaje requiere de la existencia de un procedimiento sistemático para que el ajuste de los pesos de la red produzcan la función deseada.



1.12 Red neural utilizando perceptrones

Para ajustar el problema de la representación, supongase que se tiene un conjunto de caracteres con la numeración de cero al nueve, supongase también que se tiene una máquina hipotética que es capaz de distinguir los caracteres pares, encendiendo un indicador en su panel sí es par y apagado sí es non (como se muestra en la figura 1-14). ¿Puede tal máquina ser representada por un perceptron? . Es decir, ¿puede un perceptron ser construido y sus pesos ajustados tal que este tenga la misma capacidad discriminatoria?. Si es así,

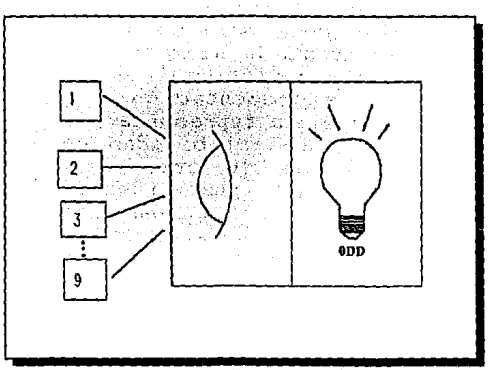
se dice que el perceptron puede representar la máquina deseada. No obstante se sabe que el perceptron es muy limitado en cuanto su habilidad de hacer representaciones. Hay muchas máquinas simples que el perceptron no se puede representar sin importar como sean ajustados los pesos.



1.13 Perceptron con varios niveles

1.14.1.2 APRENDIZAJE DEL PERCEPTRON

Hay serias cuestiones acerca de la eficiencia de almacenamiento del perceptron (y otras redes) relativas a la memoria de la computadora y métodos de recuperación. Por ejemplo debería ser posible almacenar todos los patrones de entrada en una memoria de computadora y entonces buscar el patrón deseado y responder a su clasificación.



1.14 Reconocimiento de imagenes

La habilidad de aprendizaje de la red neuronal es su propiedad que más intriga. Así como los sistemas biológicos son modelados, éstas redes se modifican así mismás como resultados de la experiencia para producir el patrón de comportamiento más deseado.

Usando el criterio de separación lineal es posible decidir en todo caso si una red de un solo nivel puede representar una función

deseada. Si la respuesta es siempre "SI", este caso es algo bueno si no se tiene la forma de encontrar los valores necesitados para los pesos y los umbrales. Si la red se tiene para un valor práctico, se necesita un método sistemático (un algoritmo), para calcular los valores. Rosenblatt (1962) proporcionó en el algoritmo de entrenamiento del perceptrón, junto con su demostración de que un perceptron puede ser entrenado sí éste se puede representar.

El aprendizaje puede ser tanto supervisado como no supervisado. El supervisado requiere de un "maestro" externo, que evalúe el comportamiento del sistema y direccione las modificaciones subsecuentes. El no supervisado no requiere de maestro, la red se auto organiza para producir los cambios deseados. El aprendizaje del perceptron es del tipo supervisado.

El algoritmo de entrenamiento del perceptron puede ser implementado en una computadora digital o en otro Hardware electrónico y la red se convierte en cierto sentido autoajustable. Por esta razón al ajuste de pesos es comúnmente llamado "entrenamiento", y se dice que la red "aprende". La demostración de Rosenblatt proporcionó un gran ímpetu a la investigación en este campo. Actualmente, de una forma u otra, los elementos del entrenamiento del perceptron se encuentran en muchos paradigmas de red modernas.

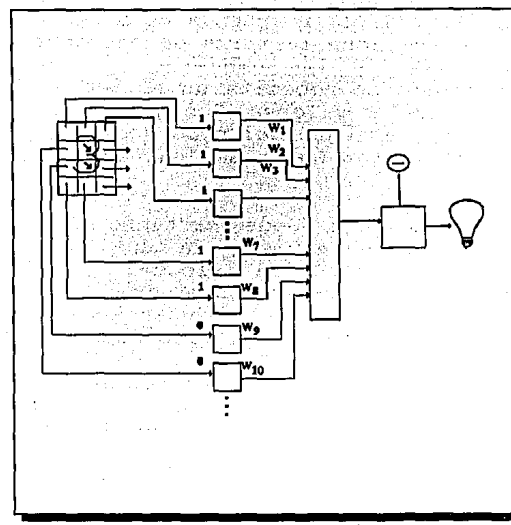
1.14.1.3 ALGORITMO DE ENTRENAMIENTO DEL PERCEPTRON.

Un perceptron es entrenado por la presencia de un conjunto de patrones en su entrada, y ajustando los pesos hasta que la salida deseada ocurra para cada uno de ellos. Supóngase que los patrones de entrada se encuentran en cartas (tarjetas). cada carta se marca como un cuadrado, y cada cuadrado puede proporcionar una entrada al perceptron. Si un cuadro tiene una línea a través de el, su salida es uno, en otro caso, su salida es cero. El conjunto de cuadros en una carta representa el conjunto de unos y ceros presentados como entradas al perceptron. La razón es que el perceptron aprenda, es decir, que cuando se aplique un conjunto de entradas representadas por numeros impares, el panel siempre se encienda (siempre en 1), mientras que para los números pares permanezca siempre apagado (siempre en 0).

La figura 1-15 muestra la configuración del perceptron. Supóngase que el vector x representa un patrón de la carta ha ser reconocida. Cada componente (cuadro) de x, (x1,x2,...,xn), es multipli-cado por su correspondiente componente del vector de peso W, (w1,w2,...,wn). Estos productos se suman. Si la suma excede el umbral η, la salida

de la neurona Y es uno (y la luz se enciende), en otro caso su salida es cero. Esta operación puede ser representada de manera compacta en forma de vector como Y=XW, seguido por la operación de comienzo.

Para entrenar a la red un patrón X es aplicado a la entrada y se calcula la salida Y. Si Y es correcto, nada cambia, sin embargo si la salida es incorrecta, los pesos conectados a las entradas mejoran estos resultados y modifican el valor para corregir el error.



1.15 Reconocimento de imagenes

Para ver como esto se realiza, se asume que una carta lleva el número como entrada al sistema y la salida Y es 1 (indicando par). Desde que esta respuesta es correcta, ningún peso se cambia. Sin embargo, si en una carta con el número 4 es la entrada al perceptron y la salida Y es 1 (impar), los pesos conectados a las entradas que están en 1 deben ser decrementados. Simílarmente sí una carta con el número 3 produce una salida de cero, aquellos pesos conectados a las entradas que están en uno deben ser incrementados, así tiende a corregir esta condición errónea.

Este método de entrenamiento se puede enumerar como sigue:

1.- Aplicar patrón de entrada y calcular la salida Y

2.-
   a) Si la salida es correcta ir al paso 1;
   b) si la salida es incorrecta, y es cero, adicionar a a cada entrada su peso correspondiente; ó
   c) Si la salida es incorrecta, y es uno, sustraer a cada entrada de se correspondiente peso.

3.- Ir al paso 1.

En un número finito de pasos la red aprenderá a separar las cartas en categorías de pares e impares. Proporcionando que los conjuntos de salidas son linealmente separables. Es decir, para todas las tarjetas impares la salida será más grande que el umbral y para todas las cartas tarjetas pares la salida estará por abajo del umbral. Observe que este entrenamiento es global, o sea, que la red aprende sobre el conjunto entero de cartas. Para minimizar el tiempo de entrenamiento, el conjunto debe ser aplicado secuencialmente una y otra vez.

1.14.1.4 PROBLEMAS DEL ALGORITMO DE ENTRENAMIENTO DEL PERCEPTRON.

Es difícil el determinar si la separación lineal satisface un entrenamiento particular que se tiene a la mano, es más en muchas situaciones del mundo real las entradas son usualmente variantes en el tiempo y pueden ser separables para un tiempo y no para otro. También, no hay un enunciado en la demostración del algoritmo de aprendizaje del perceptron que indique que cantidad de pasos se requieran para entrenar a la red.

Estas preguntas nunca han sido satisfactoriamente contestadas, y son relacionadas ciertamente a la naturaleza del conjunto que se esta aprendiendo. Generalmente las respuestas no son tan satisfactorias para las redes modernas como lo son para los perceptrons. Estos problemas representan aéreas importantes para la investigación actual.

LA REGLA DELTA.

Una importante generalización del algoritmo de entrenamiento del perceptron, es la llamada regla delta, extiende esta técnica para entradas y salidas continuas. Para ver como fue desarrollado, observe el paso 2 del algoritmo de entrenamiento del perceptron, puede ser apoyado y generalizado por la introducción del término δ, el cual es la diferencia entre la salida deseada o destino T y la salida actual A. En símbolos :

$$\delta=(T-A) \qquad (1-3)$$

El caso en el cual $\delta=0$ corresponde al paso 2a, en el cual la salida es correcta y nada se hace. el paso 2b corresponde cuando $\delta>0$ mientras que el paso 2c corresponde cuando $\delta<0$.

En cualquiera que estos casos el algoritmo de entrenamiento del perceptron se satisface si δ es multiplicado por el valor de cada entrada X_i y este producto es sumado al correspondiente Peso. Para generalizar esto, el coeficiente del "rango de aprendizaje" n multiplicando el producto δX_i permite el control del tamaño promedio de cambios de peso. Simbólicamente:

$$\Delta_i = n \delta X_i \qquad (1\text{-}4)$$

$$W_i(n+1)=W_i(n)=\Delta_i \qquad (1\text{-}5)$$

donde

Δ_i = la corrección asociada con la i-ésima entrada Xi
$W_i(n+1)$ = el valor del peso i después del ajuste
$W_i(n)$ = el valor del peso 1 antes del ajuste

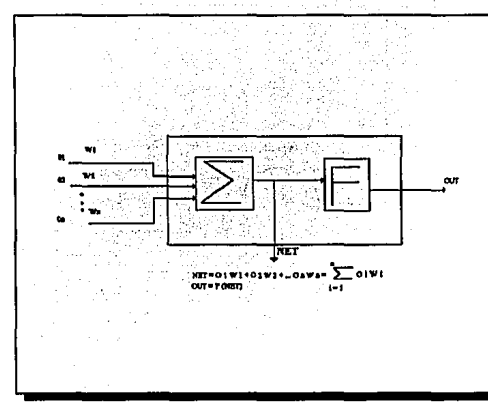
La regla delta modifica los pesos apropiadamente para las salidas deseadas y actual de alguna polaridad y para ambas entradas y salidas continuas y binarias. Estas características han abierto una riqueza de nuevas aplicaciones.

1.14.2 RETROPROPAGACION

Como mencionamos anteriormente, la retropropagación es un concepto relativamente genérico, puesto que es aplicable para una gran variedad de problemas. Este modelo es el algoritmo entrenador supervisado más predominante. El aprendizaje supervisado implica que tenemos que tener un conjunto de asociaciones de patrón "buenas" para entrenar con él. Tenemos que tener un conjunto de técnicas disponibles, y una topología (en función de patrones de entrada y salida y el número y el tamaño de niveles ocultos) apropiada para el problema.

El modelo de retropropagacion presentado en la figura 1.16 tiene tres capas de neuronas: Una capa de entrada, una capa oculta, y una capa de salida. Hay dos capas de pesos sinapticos. Hay un término de tasa de aprendizaje indicando la cantidad del cambio del peso que se efectúa en cada paso. Este número se encuentra entre 0 y 1. También hay un término momentum, que indica cuando un cambio de peso anterior deberá de realizarse por el peso actual. También existe un término qué indica la tolerancia en que podemos aceptar una salida como buena.

El método empleado para entrenar una red neuronal se conoce como retropropagación de errores. Consiste en presentar a la red un par de patrones relacionados entre si, por lo general en forma de vectores, uno de entrada y otro de salida. En su forma más simple el proceso de aprendizaje consta de dos fases. En la primera, la señal generada por la presentación de uno de los vectores de entrada, se propaga hacia la salida, donde produce una señal de salida. Los elementos del vector de salida se comparan con los del vector esperado de lo cual resulta una diferencia. La segunda etapa implica la propagación de la diferencia encontrada, en sentido contrario, de la salida a la entrada y la corrección simultánea de las cargas de las conexiones entre neuronas. Corrigiendo las cargas, la red se adapta y "aprende" por asociación, a producir los resultados correctos.



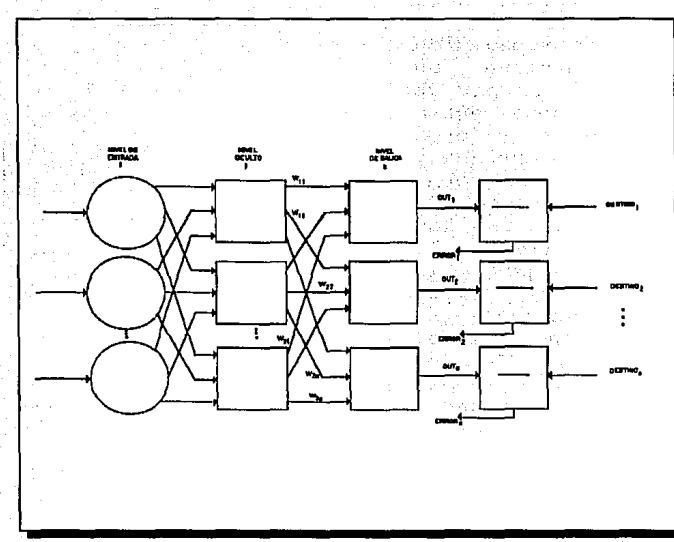
1.16 Neurona Artificial

Por muchos años no hubo un algoritmo para el entrenamiento de redes neuronales artificiales, Desde que las redes de un solo nivel proveen muchas limitaciones, entran en un total eclipsamiento.

La invención de el algoritmo de entrenamiento juega un papel importante en el resurgimiento del interés en redes neuronales. La retropropagación es un método sistemático para el entrenamiento de redes neuronales multinivel.

La alternativa es inconsistente en la definición del número de niveles en las redes. Algunos autores hacen referencia al número de niveles de neuronas, otros hacia el nivel de pesos. Debido a que esta última definición es más funcionalmente descriptiva; La red en la figura 1-17 es considerada en esta definición, La cual consiste de 2 niveles, también, una neurona es asociada con un conjunto de pesos que son conectadas por sus entradas. Así, el peso en la capa 1 termina en la neurona de la capa 1. La entrada o descripción de la capa es designada por la capa 0.

La retroalimentación puede estar aplicada hacia las redes con un único número de niveles; sin embargo únicamente 2 niveles de pesos son necesitados para demostrar el algoritmo. Por este punto en discusión, únicamente la antepropagación de redes son considerados. Es posible aplicar la retropopagación a las redes con conexiones antepropagación.

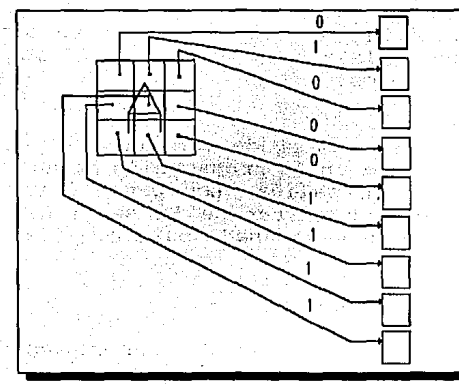


1.17 Red neuronal de dos niveles utilizando Retropropagación

1.14.2.1 ENTRENAMIENTO.

El objetivo del entrenamiento de una red es ajustar los pesos de modo que la aplicación de un conjunto de entradas produzcan el conjunto de salidas. Por estas razones, este conjunto de entrada-salida pueden estar referenciadas como vectores, el entrenamiento asume que cada vector de entrada es apareada con un vector destino representado por la salida deseada, al mismo tiempo estos son llamados entrenamiento de apareamiento. Usualmente una red es entrenada siempre con un número de entrenamiento de apareamiento. Por ejemplo, la parte de entrada de un entrenamiento de apareamiento podría consistir de un patrón de unos y ceros representando una imagen binaria de una letra del alfabeto. La figura 1-18 muestra un conjunto de entradas para dibujar la letra A. Si una línea pasa a través de una pared, la correspondiente entrada de la neurona es uno, de otro modo esta entrada de la neurona es cero. La entrada podría ser un número que representa a la letra A, o quizás otro conjunto de unos y ceros que pueda ser usado para producir un patrón de salidas.

Antes de empezar el proceso de entrenamiento todos los pesos tienen que ser inicializados por un pequeño número aleatorio. Esto asegura que la red no se sature por un valor largo de pesos, y evitar ciertos entrenamientos patológicos. Por ejemplo si todos los pesos empezaron con valores iguales y el deseo de desempeñar requiera un valor igual, la red no podría aprender.



1.18 Reconocimiento de imagenes

El entrenamiento de propagación de una red requiere de los siguientes pasos:

1.- Seleccionar el siguiente entrenamiento de apareamiento para el conjunto de entrenamiento; aplicar el vector de entrada la entrada de la red.
2.- Calcular la salida de la red.
3.- Calcular el error entre la salida de la red y la salida deseada.
4.- Ajustar los pesos de la red de modo que se minimicé el error.

5.- Repetir los pasos del 1 al 4 para cada vector en el conjunto de entrenamiento hasta que el error para que todo el conjunto sea aceptablemente bajo.

Las operaciones requieren los pasos 1 y 2 (antes mencionados). Son similares al modo en el cual el entrenamiento de la red podría esencialmente ser usaao, esto es, un vector de entrada es aplicado y el resultado de entrada es calculado. Los cálculos se desempeñan en el principio **Nivel por Nivel.** Refiriéndose a la figura 1-17, primero las salidas de la neurona en el nivel j son calculadas; éstas son usadas como entradas para el nivel k, las salidas de la neurona del nivel k son calculadas y éstas constituyen el vector de salida da la red.

En el paso 3, cada salida de la red es etiquetada como OUT (ver fig. 1-17), es substraída de su correspondiente componente del vector destino, para producir un error. Este error es usado en el paso 4 para ajustar los pesos de la red, donde la polaridad y la magnitud de los pesos son cambiados y determinados por el algoritmo de entrenamiento.

Después de repetir satisfactoriamente estos 4 pasos, el error entre las salidas actuales y las salidas deseadas estarán produciendo un aceptable error, y la red se dirá que está **entrenada.** En este punto, la red es usada para el reconocimiento y sus pesos no serán cambiados.

Con ésto se puede ver que los pasos 1 y 2 constituyen un **pase de adelanto** en la propagación de una señal desde la entrada de la red hacia su salida. Los pasos 3 y 4 son un **pase de reversa**; estos cambian el error de la señal de propagación a través de la red, donde este es usado para el ajuste de pesos.

Pase de Adelanto

Los pasos 1 y 2 pueden ser expresados como un vector : un vector de entrada X es aplicado y se produce un vector de salida Y. El par de vectores de entrada X y T provienen de un conjunto de entrenamiento. El cálculo se desempeña en X para producir un vector de salida Y.

Como se ha visto, el cálculo en redes multi-nivel se hace nivel por nivel, comenzando por el nivel más cercano a las entradas. El valor NET de cada neurona en el primer nivel es calculado como la sumatoria de todos los pesos de esta neurona de entrada. La función de activación F entonces comprime a NET para producir el valor OUT para cada neurona en este nivel. Una vez que se encuentra el conjunto de salidas para un nivel, éste sirve como entrada para el siguiente nivel. Este proceso se repite nivel por nivel hasta que se produzca el conjunto final de salidas de la red.

Este proceso puede ser declarado brevemente en notación vectorial. Los pesos entre las neuronas se definen por la matriz W. Por ejemplo, el peso desde la neurona 8 del nivel 2 a la neurona 5 del nivel 3 es definido como $w_{8,5}$. En lugar de usar la sumatoría de productos, el vector NET para el nivel N puede ser expresado como el producto de X y W. En notación vectorial N=X*W. Aplicando la función F al vector NET para el nivel N, componente por componente, produce el vector O. Así, para un nivel dado, la siguiente expresión describe el cálculo para este proceso :

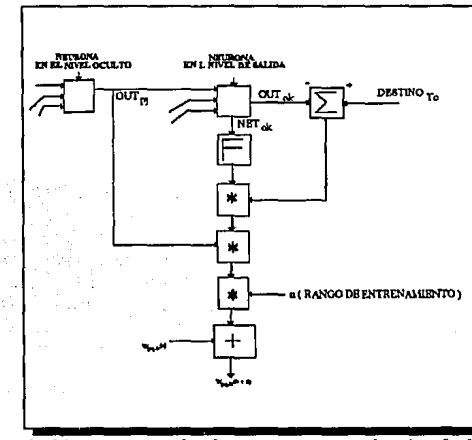
$$O = F * (X*W) \qquad (1\text{-}6).$$

El vector de salida para un nivel es el vector de entrada para el nivel siguiente, y calculando las salidas del nivel final se requiere al aplicación de la ecuación 1-6 para cada nivel, desde la entrada a la salida de la red.

Paso de Reversa

Ajuste de los pesos del nivel de salida. Ya que el valor deseado está disponible para cada neurona en el nivel de salida, ajustando los pesos asociados es fácilmente realizable usando una modificación de la regla Delta. Los niveles internos son referidos como **niveles ocultos**, como sus salidas no tienen un valor deseado para compararse, el entrenamiento es más complicado.

La figura 1-19 muestra el proceso de entrenamiento para un sólo peso desde la neurona p del nivel oculto j a la neurona q del nivel de salida k. La salida de una neurona en el nivel k se substrae desde el valor deseado para producir una señal de error. Esta es multiplicada por la función de compresión [OUT(1-OUT)] calculada para la neurona del nivel k, por tanto se produce el valor δ.



1.19 Entrenando los pesos en el nivel de entrada

$$\delta = OUT(1-OUT)*(Val.Deseado - OUT) \qquad (1-7).$$

Entonces δ es multiplicada por OUT desde una neurona del nivel j, que es la neurona fuente para el peso en cuestión. Este producto es ahora multiplicado por el coeficiente de razón de entrenamiento η (típicamente entre 0.01 y 1.0) y el resultado se suma al peso. Un proceso idéntico se realiza para cada peso procedente de una neurona del nivel oculto a una neurona del nivel de salida.

Las siguientes ecuaciones ilustran este cálculo :

$$\Delta\omega_{pq,k} = \eta * \delta_{q,k} * OUT_{pj} . \qquad (1-8).$$

$$\omega_{pq,k}(n+1) = \omega_{pq,k}(n) + \Delta\omega_{pq,k} \qquad (1-9).$$

donde,

$\omega_{pq,k}(n)$ = el valor del peso de la neurona p en el nivel oculto a la neurona q en el nivel de salida en el paso n (antes del ajuste); observe que el subíndice k indica que el peso es asociado con su nivel de destino.

$\omega_{pq,k}(n+1)$ = valor de peso en el paso n+1 (después del ajuste).

$\delta_{q,k}$ = valor de δ para la neurona q en el nivel de salida k.

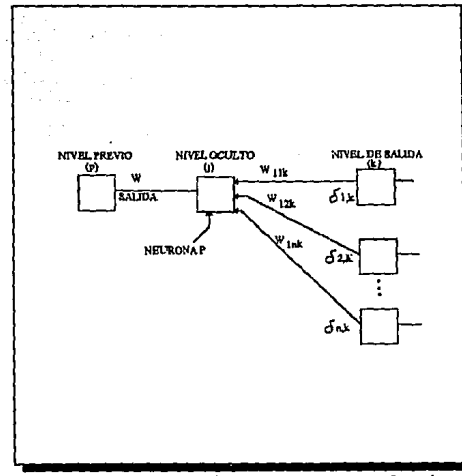
OUT_{pj} = valor de OUT para la neurona p en el nivel oculto j.

Obsérvese que los subíndices p y q se refieren a una neurona específica, mientras que los subíndices j y k se refieren al nivel.

Ajuste de los pesos de los niveles ocultos.

Los niveles ocultos no tienen un vector deseado, de modo que el proceso de entrenamiento descrito anteriormente no puede ser usado. Este impedimento de un entrenamiento deseado frustra los esfuerzos para entrenar una red multi-nivel hasta que la retropropagación proporcione un algoritmo factible. La retropropagación entrena a los niveles ocultos por la propagación de regreso del error de salida a través de la red nivel por nivel, ajustando los pesos para cada nivel.

Las ecuaciones 1-8 y 1-9 se usan para todos los niveles, tanto de salida como ocultos; sin embargo, para los niveles ocultos δ debe ser generada sin el beneficio del vector deseado. La figura 1-20 muestra como se realiza esto. Primero, δ se calcula para cada neurona del nivel de salida como se indica en la ecuación 1-7. Esta se usa para ajustar los pesos que alimentan al nivel de salida, entonces este se propaga de regreso a través de los mismos pesos para generar el valor δ para cada neurona en el primer nivel oculto. Estos



1.20 Entrenando los pesos en el nivel oculto

valores de δ se usan ahora para ajustar los pesos de este nivel oculto y de igual modo, es propaga de regreso para todos los niveles precedentes.

Considérese una sola neurona en el nivel oculto justo atrás del nivel de salida. En el pase de adelanto, esta neurona propaga su valor de salida a las neuronas del nivel de salida a través de la interconexión de pesos. Durante el entrenamiento estos pesos operan en reversa, pasando el valor de δ del nivel de salida de regreso al nivel oculto. Cada uno de estos pesos es multiplicado por el valor δ de la neurona que está conectada en el nivel de salida. El valor de δ necesitado por la neurona del nivel oculto se produce por la sumatoría de todos esos productos y multiplicando por la derivada por la función de compresión :

$$\delta_{pj} = OUT_{pj} * (1-OUT_{pj}) \left(\sum_{q} \delta_{q,k} \omega_{pq,k} \right) \qquad (1-10)$$

(Ver figura 1-20). Con la última δ, los pesos que alimentan al primer nivel oculto se pueden ajustar usando las ecuaciones 1-8 y 1-9, modificando los índices para indicar los niveles correctos.

Para cada neurona de un nivel oculto dado, las δ's se pueden calcular y todos los pesos asociados con ese nivel se deben ajustar. Esto se repite, moviendo de regreso hacia la entrada nivel por nivel, hasta que todos los pesos sean ajustados.

Con notación vectorial, la operación de propagación del error de regreso se puede expresar de forma más compacta. Llamando al conjunto de las δ's del nivel de salida $\mathbf{D}_k$ y al conjunto de pesos para el nivel de salida como el arreglo $\mathbf{W}_k$. Para llegar a $\mathbf{Dj}$, el vector δ para el nivel oculto, se deben de satisfacer los siguientes dos pasos:

1. Multiplicar el vector δ del nivel de salida ($\underline{D}_k$) por la transpuesta de la matriz de pesos conectados al nivel oculto de salida ($\mathbf{W}_k^t$).

2. Multiplicar cada componente del producto resultante por la derivada de la función de compresión para la correspondiente neurona del nivel oculto.

Simbólicamente,

$$D_j = D_k * W_k^t \$[O_j \$ (I - O_j)] \qquad (1\text{-}11)$$

donde el operador **$** se define para indicar la multiplicación componente por componente de dos vectores. O_j es el vector de salida del nivel j, e **I** es un vector, cuyos componentes son todos uno.

Adicionando una Neurona Predispuesta.

En muchos casos, es deseable proporcionar a cada neurona con un entrenamiento predispuesto. Esta compensación la origina la función logística, produciendose un efecto similar al ajuste del umbral de la neurona del Perceptron, por tanto permitiendo una mayor rapidez de convergencia del proceso de entrenamiento. Esta característica fácilmente es incorporada en el algoritmo de entrenamiento; un peso conectado a +1 se suma a cada neurona. Este peso es entrenable de la misma manera que para todos los demás pesos, excepto para el fuente que es siempre +1 en vez de ser la salida de una neurona en el nivel previo.

Momentum

Rumelhart, Hinton y Williams (1986) describen un método para mejorar el tiempo de entrenamiento del algoritmo de propagación, mientras que se aumenta la estabilidad del proceso. Llamado **momentum**, el método implica sumar un término al ajuste del peso que es proporcional a la cantidad del cambio de peso anterior. Una vez que se hizo el ajuste es "memorizado" y sirve para modificar todos los subsecuentes ajustes de peso. Las ecuaciones de ajuste se modifican como sigue :

$$\Delta\omega_{pq,k}(n+1) = \eta * (\delta_{q,k} * OUT_{pj}) + \alpha[\Delta\omega_{pq,k}(n)] \qquad (1\text{-}12)$$

$$\omega_{pq,k}(n+1) = \omega_{pq,k}(n) + \Delta\omega_{pq,k}(n+1) \qquad (1\text{-}13)$$

donde α es el coeficiente de momentum que está comúnmente puesto a un valor alrededor de 0.9.

Usando el método del momentum, la tendencia de la red a seguir el fondo de abismos (cuencas) angostos en la superficie de error (si ellos existen) en lugar de cruzar de un lado a otro. Este método se observa que trabaja bien en algunos problemas, pero tiene un pequeño efecto negativo en otros.

Sejnowsky y Rosenberg (1987) describen un método similar basado en un amortiguamiento exponencial que puede resultar ser superior en algunas aplicaciones.

$$\Delta\omega_{pq,k}(n+1) = (1-\alpha) * \delta_{q,k} * OUT_{pj} + \alpha\Delta\omega_{pq,k}(n) \qquad (1\text{-}14)$$

Entonces el cambio se calcula:

$$\omega_{pq,k}(n+1) = \omega_{pq,k}(n) + \eta\Delta\omega_{pq,k}(n+1) \qquad (1\text{-}15)$$

Donde α es la constante de amortiguamiento en el rango de 0.0 a 1.0. Si α es 0.0, entonces el amortiguamiento es mínimo; el ajuste completo de peso proviene de calcular nuevamente el cambio. Si α es 1.0, el nuevo ajuste es ignorado y se repite el anterior. Entre 0 y 1 es una región en donde el ajuste de peso es amortiguado por una cantidad proporcional a α. Una vez más η es el coeficiente de la razón de entrenamiento, que sirve para ajustar el tamaño del cambio del peso promedio.

1.14.2.2 APLICACIONES

La retropropagación ha sido aplicada a una amplia variedad de aplicaciones, un par de estas se utilizan para demostrar el poder de este método.

NEC en japón ha anunciado recientemente que ha aplicado retropropagación a un nuevo sistema de reconocimiento óptimo de carácter, así mejorando la precisión por arriba del 99%. Esta mejora fue realizada a través de una combinación de algoritmos convencionales con una red de retropropagación proporcionando una verificación adicional.

Sejnowski y Rosenberg (1987) lograron un proceso espectacular con NetTalk, un sistema que convierte texto impreso un lenguaje hablado altamente inteligible. Sus cintas de registro del proceso de

entrenamiento dio una fuerte semejanza a los sonidos de un niño en varios estados del aprendizaje para hablar.

Burr (1987) ha usado retropropagación en máquinas de reconocimiento de palabras manuscritas. Los caracteres son normalizados, por tamaño puestos en rejillas, y se realizan proyecciones de líneas a través de los recuadros de las rejillas estas proyecciones forman entonces las entradas a la red de retropropagación. El reporta una aproximación del 99.7% cuando es usado con un filtro diccionario de filtro.

Cottrell, Munro, y Zipser (1987) reportan una aplicación lograda de compresión de imagen la cual fue representada por un bit por pixel, una mejoría de 8 veces por encima del dato de entrada.

A pesar de muchas aplicaciones realizadas de la retropropagación esta no es una panacea. Lo más molesto es el gran e incierto proceso de entrenamiento. Para problemas complejos este deben requerir días o semanas para entrenar la red, y no serán del todo entrenadas. El tiempo más grande de entrenamiento puede ser el resultado de un paso de tiempo no óptimo. Las características completas de entrenamiento generalmente surgen de dos fuentes: La mínima local y parálisis de red.

PARÁLISIS DE RED

Así como el entrenamiento de la red, los pesos pueden ser ajustados a valores muy grandes, esto fuerza a todas y a la mayoría de las neuronas a operar a vectores muy grandes de OUT, en una región donde la derivada de función de comprensión es muy pequeña, ya que el error que se envía al proceso de entrenamiento puede tener una pausa virtual.

Ahí se tiene un pequeño entendimiento teórico de este problema, es comúnmente evadido por la reducción de tamaño de peso η,pero este extiende el tiempo de entendimiento. Varios heurísticos han sido empleados el parálisis o para el reconocimiento de sus defectos, pero esto puede ser descrito como experimental.

MÍNIMO LOCAL

La retropropagación emplea un tipo de gradiente negativo, esto es sigue a la pendiente de la superficie decadente de error, constantemente ajustando los pesos hacia un mínimo. La superficie de error de una red compleja es altamente convulcionada en el total de montes, valles, pliegues y abismos en un espacio n-dimencional. La red puede estar atrapada en un mínimo local (un valle profundo) cuando este cercano en mínimo más profundo. Desde el punto de vista limitado de la red, todas las direcciones de la red están arriba y no hay forma de escapar. Un método de entrenamiento estadístico puede ayudar a evitar esta trampa, pero estos tienden a ser lentos.

Wasserman (1988a) propone que combinando los métodos estadísticos de la máquina Cauchy con el gradiente descendiente de retropropagación para producir un sistema que encuentre un mínimo global, que la región de entrenamiento más grande de retropropagación.

TAMAÑO DEL PASO

Una lectura cuidadosa de la demostración de convergencia de Rumelhart, Hinton y Williams (1986) muestran que un pequeño ajuste infinitesimal de peso se sume. Esto es claramente impráctico, ya que esto implica un tiempo infinito de entrenamiento, es necesario seleccionar un peso de tamaño finito, y que este se muy pequeño como para producir otra decisión a la de la experimentada. Si el tamaño de pesos es muy pequeño, la convergencia puede ser muy lenta, y si es muy grande, puede resultar en parálisis o en inestabilidad continua. Wasserman (1988b) describe un algoritmo de tamaño de peso alartivo, intentando ajustar el tamaño de peso automáticamente conforme se realiza el proceso de entrenamiento.

INESTABILIDAD TEMPORAL

Si una red esta aprendiendo a reconocer el alfabeto, no es bueno que aprenda la letra B,si de hecho olvida aprender la letra A. Un proceso es necesario para enseñar a la red para aprender un conjunto completo de entrenamiento si una disrrupta de lo que esta a aprendido. esta demostración de convergencia de Rumelhart cumple con esto, pero requiere que la red este mostrando todos los vectores en el conjunto de entrenamiento antes de cualquier peso. El cambio necesario de pesos deberá de acumularse antes de cualquier otro (el deseado), requiriendo así un almacenamiento adicional. Este método no debe de ser muy usado,si la red presenta un ambiente continuamente cambiante donde en la red nunca debe haber el mismo vector de entra dos veces. Este puede extraviarse u oscilar desorientadamente, en este sentido la retropropagación falla para simular sistemas biológicas. Esta discrepancia conducen o llevan al sistema ART de Grossberg.

1.14.3 CONTRAPROPAGACION

El modelo de red neural multinivel denominado Contrapropagación (CPN, por sus siglas en inglés) fue desarrollado por Robert Hecht-Nielsen. Este modelo va mucho más allá de los límites con respecto a la representación (mapeo) de las redes de un solo nivel.

Una de sus fuertes ventajas es que su entrenamiento es mucho más rápido que en las redes de retropropagación; este tiempo se reduce en algunas ocasiones hasta más de cien veces, proporcionando por lo

tanto una solución para aplicaciones en donde no se permiten largos períodos de entrenamiento. Las redes CPN también tienen la capacidad de generalizar esto les permite un mejor manejo de datos cuando estos son errores o cuando no se han manejado anteriormente.

La arquitectura CPN es una combinación de otros dos modelos: el Vector Cuantizador Lineal de Kohonen y el sobresaliente Codificador de Grossberg. Este modelo también resalta las ventajas de las redes autoorganizadas de Kohonen (en donde el vector cuantizador lineal es un ejemplo) y el muy brillante codificador de Grossberg. El nivel Kohonen manifiesta la generalización y las capacidades de "búsqueda por tabla" de las redes autoorganizadas. El nivel Grossberg manifiesta la sobresaliente habilidad de actuar como un codificador mínimo de patrones. Juntando ambas características de ambos modelos, se obtienen ambás propiedades que no tienen cada uno de manera individual, por lo tanto, se mejora el comportamiento de cada uno de ellos.

1.14.3.1 TOPOLOGÍA.

Las redes de contrapropagación son estructuras de tres niveles (patrón de entrada, nivel oculto, patrón de salida) con dos niveles de conectividad (Sinápsis). La Sinápsis entre el nivel de entrada y el nivel oculto actúa como un Vector Cuantizador **"el ganador toma todo"** de los patrones de entrada. Para cada patrón de entrada, la neurona del nivel oculto cuyo vector de Sinápsis con el nivel de entrada es muy similar al patrón de entrada que se escoje como la neurona ganadora, efectivamente "categorizando" el patrón de entrada. El segundo nivel de Sinápsis (el sobresaliente codificador de Grossberg) tiene sus pesos ajustados a cada patrón asociado basandose en la activación de la neurona ganadora del nivel oculto y el vector de salida. La red CPN se muestra en la figura 1.21.

1.14.3.2 ENTRENAMIENTO.

Los pesos entre el nivel de entrada y el nivel de salida se inicializan con valores aleatorios. Un patrón de entrada se aplica a la red CPN, y se escoge la neurona del nivel oculto cuyo vector de pesos se aproxima más a este patrón de entrada. El vector de pesos elegido en el nivel Kohonen es entonces ajustado en base a la diferencia entre el vector de peso y el vector de entrada, dispuesto por la razón de aprendizaje en el tiempo **t**.

Los pesos entre la neurona ganadora en el nivel oculto y cada

neurona en el nivel de salida es entonces ajustado por la producción de la activación de la neurona ganadora y cada activación de la neurona ganadora, modificada por una constante razón de aprendizaje.

La figura 1.22 muestra una red CPN completa. En modo de operación (después del entenamiento) se le aplican los vectores de entrada **x** y **y**, y la red produce los vectores de salida **x´** y **y´**, que son aproximaciones de **x** y **y**, respectivamente. En este caso, se asume que **x** y **y** son vectores unitarios normalizados que por lo tanto, tienden a producir vectores normalizados como salida.

En el entrenamiento, se aplican los vectores **x** y **y** como entradas y como salidas de la red, de tal manera que **x** se emplea para entrenar la salida **x´**, mientras que **y** se utiliza para entrenar a la red aproducir la salida **y´** del nivel Grossberg. En suma, toda la red es entrenada usando el mismo metodo descrito para una propagación hacia adelante (FORWARDPROPAGATION o COUNTERPROPAGATION). Las neuronas Kohonen reciben las entradas tanto de **x** como de **y**, y estas entradas no se pueden distingir, pareciendo que se trata de un único y gran vector formado por **x** y **y**.

El resultado obtenido es un mapa de identidad en el cual la aplicación de un par de vectores de entrada produce sus propias réplicas (aproximaciones) a la salida.

Lo interesante de este paradicma resulta en que si se aplica solo uno de los vectores de entrada (en la fase de operación), por decir el vector **x**, ignorando el vector **y**, se produciran las dos salidas **x´** y **y´**. Es decir, si **F** es una función que mapea **x** hacia **y´**, entonces la red permite tal aproximación. De la misma forma, si existe la inversa de **F**, esto es, solo aplicando el vector **y** (poniendo x en cero), se produce el vector **x´**. Esta singular habilidad de la contrapropagación de generar una función y su inversa la hace adecuada para una gran cantidad de aplicaciones. El nombre de la red proviene precisamente de este efecto de contra flujo generado a tiempo de aprendizaje.

1.14.3.3 VENTAJA Y DESVENTAJA DEL MODELO CPN.

La capacidad de aprendizaje no supervizado del CPN, es uno de sus más grandes potenciales. Sin embargo, es posible crear una versión de aprendizaje supervisado con la misma topología CPN descrita. También es posible tener "multiples ganadoras"; más de una neurona puede ser activada en el nivel oculto.

Una muy grande desventaja del CPN, es que necesita una neurona individual en el nivel oculto para cada patrón que se desee aprender. En otras palabras la capacidad es **n** patrones, donde **n** es el número de neuronas en el nivel oculto.

CAPITULO 2

2.1 PROGRAMACION ORIENTADA A OBJETOS
- 2.1.1 OBJETO Y CLASE
- 2.1.2 ENCAPSULACIÓN
- 2.1.3 HERENCIA
- 2.1.4 POLIMORFISMO

2.2 CUESTIONES ACERCA DE OOP CUANDO SE CREA GRANDES SISTEMAS

2.3 LENGUAJE C++
- 2.3.1 DESARROLLO DE REDES NEURONALES EN C++

2.4 REPRESENTACION DE DATOS EN REDES NEURALES
- 2.4.1 TRANSFORMACIÓN DE DATOS
- 2.4.2 TIPO Y CANTIDAD DE DATOS
- 2.4.2.1 RECOLECCION DE DATOS
- 2.4.2.2 ORGANIZACION DE DATOS

OOP

CAPITULO 2

El campo de la Inteligencia Artificial está dominado en la actualidad por disciplinas de manipulación lógica y simbólica. Recientemente ha surgido un estilo de programación que se ha puesto de moda, presentándolo como lo más innovador y es conocida como Programación Orientada a Objetos (OOP). Un gran número de arrebatadoras aseveraciones, particularmente con respecto a la productividad del programador y el código reusable, son hechas por la OOP, muchas de ellas por las firmas que se encuentran en el negocio de hacer y vender herramientas de OOP. La programación orientada a objetos es ampliamente aclamada y ha logrado muchos éxitos notables (así como muchos fallos), puesto que la programación orientada a objetos se emplea para ejecutar tareas muy complicadas que en el pasado solamente podían llevarse a cabo por un número limitado de personas expertas intensamente entrenadas. A través de la aplicación de las técnicas de Inteligencia Artificial, la programación orientada a objetos capta el conocimiento básico que permite a una persona desempeñarse como un verdadero experto frente a problemas muy complicados.

La programación orientada a objetos es una forma nueva de aproximarse a las tareas de programación. Las aproximaciones de programación han ido cambiando de manera dramática desde la invención de la computadora. La razón principal de los cambios ha sido adaptarse a la creciente complejidad de los programas. Por ejemplo, cuando aparecieron las computadoras, la programación se realizaba insertando en el panel de control mediante conmutadores

las instrucciones binarias para la máquina. Este método funcionaba, siempre y cuando los programas no fuesen tan largos. A medida que los programas crecieron, se invento el lenguaje ensamblador para realizar programas cada vez más grandes y complejos. Pero a medida que estos seguían creciendo, se fueron presentando los lenguajes de alto nivel para dar al programador más herramientas con las cuales pudiese hacer frente al problema de la complejidad.

Los años 60 vieron nacer la programación estructurada. Al utilizar lenguajes estructurados fue posible, escribir programas moderadamente complejos con bastante facilidad. Sin embrago, incluso con métodos de programación estructurada, cuando un proyecto alcanza unas ciertas dimensiones se vuelve incontrolable. Hoy en día, hay muchos proyectos que han alcanzado o están alcanzando el punto en que ya no funciona la programación estructurada. Para resolver este problema, se invento la programación orientada a objetos.

La programación orientada a objetos ha tomado las mejores ideas de la programación estructurada y las ha combinado con varios conceptos nuevos y potentes que incitan a contemplar las tareas de programación desde un nuevo punto de vista. La programación orientada a objetos permite descomponer más fácilmente un problema en subgrupos de partes relacionadas del problema. Entonces, utilizando el lenguaje, se pueden traducir estos subgrupos a unidades autocontenidas, llamadas objetos.

El más fundamental de los propósitos de la programación orientada a objetos es el de permitir a los programadores crear programas más grandes y complejos. La clave para alcanzar esta meta es el objeto. La programación orientada a objetos implica la creación de objetos, los cuales combinan y encapsulan tanto los datos como el código que opera sobre estos. Los objetos pueden contener elementos públicos y privados. Cuando un elemento de un objeto es privado, solamente los elementos que son de ese objeto pueden tener acceso a él. Los elementos públicos pueden tener acceso en cualquier otra parte del programa. La ventaja de utilizar objetos es que, si se aplica correctamente, un objeto es una única entidad lógica que resulta más sencilla de entender y de manejar que los distintos elementos que forman el objeto.

Otra característica de la programación orientada a objetos es que permite crear una jerarquía de objetos, pasando de lo más general a lo más específico. En esta jerarquía, cada objeto hereda las características de los que le anteceden. La posibilidad de crear estructuras jerárquicas permite al programador organizar cuidadosamente las distintas partes del programa en unidades claras e independientes unas a otras.

2.1

PROGRAMACION ORIENTADA A OBJETOS

La programación orientada a objetos, surge en el preciso momento en el que el programador logra un nivel de abstracción de información superior al que lograba bajo condiciones convencionales. Cuando se programaba en el lenguaje de máquina, solo se podía representar con ceros y unos la información y los programas que operaban sobre ésta. Una preocupación que seguramente se presentaría, era la de representar un problema del mundo real en el lenguaje de ceros y unos, lo cual debió de ser un gran reto.

Bajo el lenguaje de máquina, la representación de un problema por medio de un programa resultó ser una actividad exclusiva de los grupos científicos, por lo que la producción de software, sólo la realizaron este tipo de personas.

En el momento en que a uno de estos científicos se le ocurrió cambiar el nivel de abstracción para la representación de un mismo problema, surge el primer lenguaje de alto nivel que permitió manejar la misma información por medio de un conjunto de símbolos (lenguaje de programación), y también representar a un entero por medio de una variable en lugar de que el programador lo asociara a una localidad física de memoria, sustituyéndose así, al lenguaje de máquina por otro que sería más cercano al lenguaje humano.

Con la aparición de los lenguajes de alto nivel, se puede decir que la programación se acercó más al manejo de objetos reales que cuando se programaba en el lenguaje de máquina. Dentro de los lenguajes de alto nivel tradicionales, existen algunos en los que el manejo de estructuras de datos se realiza de tal forma, que se asemejan mucho a la manera en la que se usa la información en la vida real, y con estos lenguajes se han construido otros que manejan la información en forma mucho más simple. Un ejemplo lo podemos ver con el lenguaje de programación C; con él se han construido otros tipos de lenguajes (como Dbase), lográndose niveles de productividad bastante aceptables.

Cuando el desarrollo de software se realizaba exclusivamente en grandes computadores, era muy común encontrar numerosos equipos humanos para el desarrollo de sistemas. Sin embargo, con la aparición de las microcomputadoras, nace el concepto del desarrollo personal del software, donde es muy común encontrar a un solo

programador, realizando desde el análisis del sistema, programación, documentación y soporte técnico, hasta mantenimiento del mismo.

Este concepto predominó bastante durante la década de los ochentas, en la que el abatimiento de los costos de las computadoras personales fue impresionante. Hacia finales de la década de los ochenta, resulta que los microcomputadoras en muchos casos, compiten en poder de cómputo con las minicomputadoras y bajo ciertas arquitecturas, hasta con las grandes computadoras.

Para el desarrollo de software es inevitable pasar por las etapas de análisis, programación, documentación, soporte técnico y mantenimiento. Cualquiera de estos pasos que se omitan o se realicen en forma deficiente, provocará problemas en el desarrollo de la etapa posterior. Para que un sistema de software logre tener una penetración comercial aceptable y una vida estable considerablemente prolongada, deberá estar cimentado en cada uno de los puntos mencionados.

Un programa de cómputo que en un principio haya tenido buena aceptación en el mercado, pero que resulte lento y costoso darle mantenimiento, por otro lado, un sistema mal analizado, puede traer como consecuencia una programación muy deficiente y muy lenta lo que provocará que el producto salga al mercado tardiamente.

Cuando los sistemas son realizados por grandes grupos de trabajo, los problemas de comunicación son complejos, y más cuando el grupo de trabajo se encuentra disperso a grandes distancias.

Las facilidades proporcionadas por los conceptos usados en la programación orientada a objetos tiende a resolver estos problemas. Es importante hacer notar que los conceptos de este tipo de programación es la parte medular para la programación en un lenguaje de este tipo, ya que es muy fácil caer en el error de usar un lenguaje orientado a objetos y hacer programas sin usar los conceptos fundamentales. Es así que unos programadores usan compiladores de C++ o de Pascal Orientado a Objetos y de ninguna manera programar en Objetos.

La programación orientada a objetos no es una metodología nueva, como algunos afirman. Varios de sus conceptos principales que manipula, ya antes habían sido manejados y han evolucionado conjuntamente con los lenguajes para el desarrollo de software, tales como simula 67. Dichos conceptos se derivan de diversos lenguajes según su aplicación. Realmente no se ha inventado nada nuevo actualmente, solo se ha estado renombrando viejos conceptos.

Esta técnica se basa en el hecho de que los datos y las funciones para manejarlos tienen una íntima relación entre sí, y un problema puede ser descompuesto en un número cualquiera de

subproblemas o cajas negras, ocultando de los demás los detalles de la implementación y manipulación de datos, el problema deberá ser resuelto en base a modelos y no a algoritmos. El punto primordial que caracteriza esta técnica es llamado **ABSTRACCION DE DATOS.**

Esta es una técnica que se ha desarrollado para facilitar la creación de sistemas, así como su mantenimiento. Como tal tiene algunas ventajas -cuando se trata de pequeños sistemas- y algunas desventajas -en la creación de grandes sistemas-.

La programación orientada a objetos es una nueva forma de abordar el trabajo de programación. Los enfoques de programación han cambiado drásticamente desde la invención de la computadora. La razón principal de este cambio ha sido para atender la creciente complejidad de los programas. Por ejemplo, cuando se inventaron las computadoras, la programación se hacía desde el panel de control de la computadora, donde se introducían las instrucciones de máquina binarias por medio de computadores (toggles). Este enfoque funcionó, siempre y cuando los programas no tuvieran más de unos cientos centenares de instrucciones de extensión. A medida que fueron creciendo los programas, se inventó el lenguaje ensamblador, de modo que un programador pudiera hacer frente a programas más grandes y cada vez más complejos, mediante la presentación simbólica de las instrucciones de máquina. A medida que siguieron creciendo los programas, se introdujeron los lenguajes de alto nivel, que le proporcionan al programador más herramientas para manejar dicha complejidad. El primer lenguaje de gran difusión fue indiscutiblemente el FORTRAN. Aunque el FORTRAN fue un impresionante primer paso, no se puede decir que sea un lenguaje que estimula la preparación de programas claros y eficientes.

La programación orientada a objetos se basa en los siguientes conceptos:

Herencia, Encapsulación, Polimorfismo, Clase y Objetos.

La OOP involucra software alrededor los " objetos " en cuestión. Estos objetos son instancias de "tipos de datos abstractos" (un término general de ingeniería de software) o "clases". Un tipo de dato abstracto es simplemente una definición de un tipo de dato complejo así como los " métodos " (funciones) que pueden ser aplicados sobre ese tipo de dato.

Un aspecto desafortunado del conjunto actual de terminología es la inconsistencia o redundancia de su aplicación. En diversas áreas,

diferentes términos son utilizados para aplicar el mismo concepto, y (a veces) el mismo término es aplicado a diferentes conceptos. Una razón de está confusión es que diferentes estructuras en los lenguajes de OOP (como C++, Smalltalk, y Eiffel) utilizan diferentes nombres para cosas idénticas.

2.1.1 Objeto y Clase

La característica fundamental de un lenguaje orientado a objetos es el **objeto**. Un objeto es una unidad lógica que almacena datos junto con el código para manejar esos datos. Cualquier cosa de tipo de dato abstracto que la clase defina es un objeto. En un programa orientado a objetos, trabajamos solamente con objetos. Nuestras únicas llamadas de función son mensajes para objetos.

Parte del código y/o de los datos que están contenidos dentro de un objeto se pueden manejar de forma privada para este objeto, y nada que este fuera de este objeto podrá tener acceso directo. Así entonces, los objetos tienen un nivel de protección contra otra sección del programa que no tiene ninguna relación con él y que en algún momento pudiera modificar o utilizar indebidamente el código que es privado para el objeto.

Por lo tanto, un objeto es algún dato definido por el programador además de una serie de operaciones que pueden accesar a ese dato. Es decir, al definir un objeto se está definiendo un nuevo tipo de dato.

Un objeto se define como un bloque de información encapsulado por el código que lo opera. El comportamiento de un objeto esta sujeto a ese código, y el estado en que se encuentre (es decir, sus atributos) son representados por su información, la cual se protege de la intervención de otros objetos.

En una aplicación, los objetos se comunican entre si y cooperan para lograr un objetivo establecido, pero cada uno es ajeno a la mecánica de operación de otro.

Imaginemos que se tiene un anaquel con muchas divisiones, como el que se usa para clasificar diskettes. Cada división contiene una pieza de información escrita en un pedazo de papel. Una división llamada "Procesadores de texto" contiene los paquetes concernientes a la edición de textos; otra división llamada Lenguajes contiene los compiladores para los lenguajes de programación, y demás por el estilo. A un anaquel como este, el cual tiene varias divisiones o espacios para insertar, remover y

actualizar información se le conoce con el nombre de estructura. En la OOP, a una estructura se le denomina **Clase.**

Una clase puede estar descrita como un "tipo de dato abstracto". Es un conjunto de elementos de información relacionados junto con todos los "métodos" que pueden operar en ese tipo de dato para representar un concepto unificado. Para un tipo de dato abstracto verdadero, el único acceso a la información misma es a través de los métodos definidos.

¿Como se construye una clase y como se determinan las etiquetas de las divisiones?

Como un ejemplo, supongase que se quiere definir y categorizar toda la información que se tiene respaldada en un disco magnético. Para este fin, se traza un plan para construir una clase y determinar cuantas divisiones (directorios) se necesitarán y como se deberán llamar basándose en el tamaño y naturaleza de los archivos y programas. Este plan entonces deberá dársele a un administrador de archivos el cual construirá una clase de acuerdo a las especificaciones del plan.

Las clases son también usadas para recolectar y ordenar información en programas de inteligencia artificial. En términos de Inteligencia Artificial (AI), las divisiones en las clases son llamadas "ranuras" y las etiquetas en las divisiones son llamadas "atributos". Las piezas de información colocadas dentro de las ranuras son llamados "valores". Cuando los valores son colocados dentro de las ranuras, se crea una entidad llamada "objeto".

Por ejemplo, supongase que mucha gente constantemente necesita reservar un salón de conferencias. Para poder acomodar a toda esta gente, es necesario tener un procedimiento formal para reservar el salón. Es necesario construir una clase para el salón de conferencias. Uno de las ranuras de la clase deberá tener el atributo "tiempo que es necesitado el cuarto de conferencias".

Aplicando esto, no es posible programar dos conferencias en el mismo salón a la misma hora. Otra ranura deberá tener como atributo "nombre de la persona que reserva el salón de conferencias", y así continuar de esta manera, para todos los atributos pertinentes de esta clase.

Cuando alguien desea reservar un salón, los valores (tiempo y nombre) necesarios para la construcción de un objeto de la clase "salón de conferencias" deberán ser recabados.

Los elementos comentados anteriormente se pueden resumir así:

1.- *Clase: un plan para la construcción de un objeto.*
2.- *Objeto: Una clase que contiene información.*
3.- *Ranura: Un lugar para guardar información.*
4.- *Atributo: Una etiqueta de una ranura.*
5.- *Valor: Información asociada con un atributo y colocada en las etiquetas de las ranuras.*

Consideremos que el salón de conferencias de el ejemplo con mas detalle. Supongase una compañía que tiene un gran numero de salones de conferencias y necesita un sistema para asignar tiempo en cada uno de ellos. La clase que la compañía necesita usar es una forma general que los empleados deberán llenar cuando necesiten usar un salón de conferencias. Nosotros llamaremos a esta forma **clase** "salón de conferencias". Al llenar cada forma, una instancia específica de la clase es creada, la cual es llamada objeto. En este caso, el objeto contendrá el nombre de la persona y todos los demás datos que deberán ser llenados para completar la forma. Durante el transcurso de un día habrá mas de una conferencia; cada conferencia representa un objeto por separado. Repitiéndolo para que quede mas claro: una clase define atributos que son importantes para hacer algo, y los objetos son un caso especifico de la clase.

La clase "salón de conferencias" deberá permitir a cualquier empleado reservar un salón y también proporcionará suficiente información para que todos los demás empleados puedan saber que es lo que sucede en el salón en cualquier momento. Una clase "salón de conferencias" típica se vera como sigue:

1. *Nombre del salón de conferencias.*
2. *Tema de discusión.*
3. *Nombre del ponente.*
4. *Hora de comienzo de la conferencia.*
5. *Hora de clausura de la conferencia.*

Se puede asumir que son cinco diferentes salones de conferencias en el edificio de la compañía. Un empleado que desee usar un salón de conferencias llenará una forma que contenga todos los atributos de la clase "salón de conferencias" y proporcionará toda la información requerida. Este salón de conferencias es ahora reservado para esa hora. Un ejemplo de un objeto del salón de conferencias que fue creado llenando una forma seria:

1.	*Cristal.*	*(nombre del salón de conferencias)*
2.	*Redes Neuronales.*	*(tema de discusión)*
3.	*Victor García*	*(nombre del ponente)*
4.	*10:00*	*(comienzo de la conferencia)*
5.	*11:00*	*(clausura de la conferencia)*

El valor del primer objeto, Cristal, corresponde a el atributo "nombre del salón de conferencias", y el segundo valor, Redes Neuronales, corresponde al atributo "tema de discusión". Los otros valores también corresponden a los atributos de la clase.

Tanto en las grandes y pequeñas compañías, existen diferentes actividades para cada hora de trabajo. Cada actividad requiere de empleados que sigan un procedimiento específico para desempeñarse en la actividad. El procedimiento podría ser tan simple como una lista de verificación. Los mecánicos de aeropuertos tienen listas de verificación de puntos que deben de verificar antes de que un avión despegue. Estas listas de verificación son derivadas de una clase. Las aerolíneas tienen muchas cosas que hacer, sin embargo ellos tienen muchas formas, cada una con una clase especifica para hacer algo. Cada forma deberá tener un número de identificación, o nombre, de manera que cada clase pueda ser solicitada y localizada. El nombre de la lista de verificación de los mecánicos puede ser llamada "preparación para el despegue". Esto nos dice algo importante acerca de las propiedades necesarias en una clase, las cuales son:

1. Nombre, con la cual la clase puede ser relacionada.
2. Atributos, los cuales definen el contenido de las clases.

Continuemos con el trabajo del mecánico. Después de obtener una forma de la clase "preparación para el despegue", el mecánico recorrerá la lista de verificación, la llenara, la firmara y la archivara, creando una forma de objeto de la forma. A este objeto se le dará un numero de identificación (el cual es realmente su nombre). Debido a que muchas de estas formas serán llenadas en un día, por lo menos una en cada despegue, esta forma en particular puede ser recobrada posteriormente conociendo su nombre (numero de identificación).

De lo anterior se puede concluir que un objeto deberá tener las siguientes propiedades:

1. *Nombre de clase, para conocer el significado de las reglas dadas a los objetos.*

2. *Nombre del objeto, para poder ser relacionado.*

3. *Atributos, los cuales definen los temas de importancia.*

4. *Valores, que son piezas de información correspondientes a los atributos.*

Otra cosa que debe acompañar a una clase y sus objetos son una serie de procedimientos y/o funciones, que también se conocen como métodos. Por ejemplo, ¿Qué método deberá ser seguido si una llanta se baja? ¿Se reparará con todos los pasajeros en el aeroplano? ¿Existe algún método para planear con un tanque casi vacío? Los métodos son por consiguiente una parte integral de la clase.

Algunos métodos deberán ser llevados a cabo antes que otros. Por ejemplo, antes de checar algunos indicadores en un avión, la energía eléctrica de éste deberá ser conectada. Por lo tanto, debemos tener un método para conectar la energía eléctrica antes de continuar con el método de chequeo de los indicadores. El anterior procedimiento, el cual no esta directamente relacionado pero que debe ser ejecutado, es llamado "demonio anterior" (Los procedimientos que existen pero que no son invocados directamente por el usuario son referidos como demonios). De manera similar, deberá haber métodos que serán invocados después que el método principal es finalizado. Estos procedimientos no están directamente relacionados pero deberán ser ejecutados. Estos son llamados "demonio posterior". Un ejemplo de un demonio posterior seria un procedimiento para llenar la forma de preparación de despegue después de que los medidores han sido checados.

Los métodos pueden ser inicializados directamente en la clase o declarados al usuario de la clase de otras maneras. En el ejemplo del mecánico, una parte de la lista de verificación seria así:

1. *Leer el medidor de gasolina y anotarlo.*
2. *Si la medición es menor de 2000 galones, recurrir al método de forma 34555.*
3. *Checar la presión en las llantas y anotarlo.*
4. *Si la presión es menor de ..... , recurrir al método de forma 34556.*

Las ranuras 2 y 4 son referencias a métodos para el manejo de ciertas ocurrencias. Pero la manera actual para medir la gasolina y checar la presión de las llantas son parte del entrenamiento del mecánico y están contenidas quizá en una forma separada con la cual el mecánico esta ya familiarizado. Estas no son detalladas en la lista de verificación.

CREANDO UNA CLASE

Cada una de las propiedades que describimos para las clases y los objetos son disponibles para que el programador construya un sistema orientado a objetos. En resumen, estas propiedades son:

1. *La habilidad para crear una clase.*
2. *La habilidad para crear objetos de esa clase.*
3. *La habilidad para crear métodos para hacer cosas con los objetos.*

Una clase deberá tener un nombre para poder relacionarla, y los atributos, los cuales describen la forma de la clase. Al dar nombres a las clases, el usuario puede fácilmente relacionar una clase requerida por su nombre, asignar valores a sus atributos, y además crear un objeto. Esto es lo que los mecánicos harán cuando ellos acudan al salón donde todas las formas son guardadas: Solicitar la forma requerida por nombre, llenar los espacios, y en el proceso crear un objeto.

Para crear un programa orientado a objetos, se requiere un procedimiento para hacer una clase y para especificar cada atributo. Aquí se encuentra el formato que deberá ser usado para describir como una subrutina de clase es creada en un programa de computadora.

```
crea-clase
     (
      sname = nombre de la clase,
      nattr =  numero de atributos para esta clase.
      attr  = nombre del primer atributo.
      attr  = nombre del segundo atributo.
                    .
                    .
                    .
      attr  = nombre del ultimo atributo.
     )
```

Para ilustrar como se crea una clase, consideremos la siguiente clase del salón de conferencias:

1. Nombre del salón de conferencias	*(Salón)*
2. Tema de discusión	*(Tema)*
3. Nombre del ponente	*(Ponente)*
4. Comienzo de la conferencia	*(Inicio)*
5. Fin de la conferencia	*(Fin)*

Para facilitar esta clase, se ha abreviado los atributos como se muestra en los paréntesis. Debido a que la clase de la sala de conferencias debe tener un nombre, se llamara AULA. Cuando se utilice AULA en el programa, el sistema reconocerá la clase del salón de conferencias junto con los cinco atributos relacionados con él. La forma de relacionar lo anterior es:

```
crea-clase
    (
      sname=AULA,
      nattr=5,
      attr=Salón,
      attr=Tema,
      attr=Ponente,
      attr=Inicio,
      attr=Fin
    )
```

CREANDO UN OBJETO

Cuando se requiera crear un objeto de una clase, todo lo que se tiene que hacer es decir al sistema el nombre de la clase y además los valores que son dados a cada atributo. El formato general para crear una subrutina de un objeto es:

```
crea-objeto
   (
    sname  = nombre de la clase de la cual el objeto es derivado.
    oname  = nombre de este objeto.
    attri1 = valor del atributo.
    attri2 = valor del atributo.
    attrin = valor del atributo.
   )
```

sname es seguido por el nombre de la clase, oname es seguido por el nombre del objeto, y attr1 es seguido por el valor que es dado a ese atributo en particular. Este último aparece para cada atributo definido en la clase. Si se suprime el nombre de atributo, el objeto tendrá un espacio en esa locación cuando el objeto es creado. El atributo puede ser inicializado en otro momento, llenando los espacios en blanco posteriormente.

Como un ejemplo, consideremos que Victor García necesita el salón de conferencias Cristal por una hora comenzando a las 10 AM para tratar sobre las Redes Neuronales. El creará un objeto y le dará un nombre que aun no se ha usado, tal como IMP1, como sigue:

```
crea-objeto
  (
   sname=AULA,
   oname=IMP1,
   Salón=Cristal,
   Tema=Redes Neuronales,
   Ponente=Victor García,
   Inicio=10:00,
   Fin=11:00
  )
```

Relacionando el objeto con su nombre, IMP1, el objeto puede ser accesado.

CONSULTA Y OPERACIONES SOBRE OBJETOS

Al crear una clase y a los objetos, se crea en realidad una base de datos. Los objetos son registros en la base de datos y la clase especifica el formato de un registro y sus componentes. Algunas de las cosas que se pueden realizar con las bases de datos son:

1. *Colocar y modificar valores en los campos (o atributos) de cada registro (u objeto).*
2. *Obtener estadísticas de la base de datos, tal como el número de conferencias que son realizadas en una hora dada.*
3. *Proveer procedimientos(métodos) para operaciones en uno o mas campos (atributos) de un registro (objeto).*
4. *La posibilidad de consultar la base de datos tal cual.*

Para explicar estas operaciones se crean dos objetos :

```
crea-objeto
  (
   sname=AULA,
   oname=ILAER1,
   Salón=UAMI,
   Tema=Ingeniería de Control,
   Ponente=Abraham Romero,
   Inicio=13:00,
   Fin=15:00
  )
```

```
crea-objeto
  (
   sname=AULA,
   oname=SIEMENS1,
   Salón=Aragón,
   Tema= PLC's,
   Ponente=Martha Granados,
   Inicio=10:00,
   Fin=12:00
  )
```

Algunas veces solo necesitaremos consultar las clases y objetos para hallar cuales se encuentran en el sistema en un momento dado. Para listar todos los objetos, es necesario llamar a una subrutina llamada, por ejemplo, LISTA-NOMBRE-OBJETOS. En nuestro ejemplo, la computadora responderá con:

```
IMP1
ILAER1
SIEMENS1
```

Para obtener una lista de todos los nombres de las clases, se podría llamar una subrutina **LISTA-NOMBRE-CLASES**. Como solo se ha creado una clase, la computadora responderá como

```
AULA
```

Podemos listar todos los valores de los atributos de un objeto especifico con la subrutina **LISTA-OBJETO**. Por ejemplo, si se requiere consultar el objeto ILAER1, se conseguiría como sigue:

lista-objeto (sname=AULA,oname=ILAER1)

y se obtendría :

Salón=UAMI,
Tema=Ingeniería de Control,
Ponente=Abraham Romero.
Inicio=13:00,
Fin=15:00

Además de crear objetos, quizás será necesario eliminarlos. Por ejemplo, cuando la conferencia sobre PLC's de Martha Granados ya fue realizada en el salón de conferencias Aragón, será necesario eliminar ese objeto de nuestra base de datos porque el ha usado ya el salón. Esto se realiza con una subrutina llamada ELIMINA. Al eliminar el objeto SIEMENS1 y verificar lo anterior con **LISTA-NOMBRE-OBJETOS** se obtendría:

IMP1
ILAER1

También habrá ocasiones en las que se desee cambiar el valor de un atributo en un objeto. Una subrutina **INSERTAR** puede ser usada para insertar un valor en los atributos de un objeto. En nuestro ejemplo, las siguientes instrucciones cambiarían la hora de comienzo (Inicio) para el objeto IMP1 de las 10:00 a las 9:00 :

insertar (sname=AULA, oname=IMP1, Inicio=9:00)

Si ahora utilizamos la subrutina **LISTA-OBJETO** :

Salón=Cristal,
Tema=Redes Neuronales,
Ponente=Victor García,
Inicio=9:00,
Fin=11:00

Una vez que un valor esta en la forma de una variable, algunas operaciones pueden ser realizadas con la variable. Para hacer esto, se utiliza la subrutina **COPIAR**. Tomemos el atributo de Ponente de Siemens1 como ejemplo. Podemos usar la subrutina para copiar Ponente a la variable **x** y entonces usar el comando de Pascal:

writeln(**x**); y se obtendría: MARTHA GRANADOS.

Si por ejemplo, deseamos escribir un reporte para la administración de las actividades del salón de conferencias, solo es necesario 'copiar' e 'imprimir' los valores de los atributos en el reporte.

INVOCANDO A LOS METODOS

Cuando se ejemplificó la lista de verificación del mecánico en relación a la presión de aire, se dijo que un método seria invocado para reinflar la llanta. El método únicamente seria colocar una bomba de aire en la llanta. En cualquier evento, el mecánico sabría que el método es para hacer eso. Examinemos esto con mas detalle:

1. Tenemos un método para checar una ranura de un objeto para un valor especifico (en este ejemplo, este fue la presión del aire en la llanta).

2. Si el valor excede alguna condición limite, nosotros invocamos un método para enfrentarlo (en nuestro ejemplo, este fue el método para reinflar la llanta si la presión estaba baja).

Consideremos otro ejemplo para estar seguros que el concepto de método es claro. En nuestro ejemplo del salón de conferencia, se describió como la subrutina INSERTAR fue utilizada para cambiar la hora de comienzo de la conferencia de Victor García de las 10:00 a las 9:00. ¿Que hubiese sucedido si alguien mas también programase una conferencia en el mismo salón entre las 9:00 y las 10:00? Si se permite el cambio anterior sin ninguna acción adecuada, dos grupos llegarían al salón de conferencias a las 9:00. Lo que hubiese sucedido es esto: cuando Victor cambio la hora de la conferencia, un procedimiento se hubiera invocado para checar el numero del salón de conferencias y las horas de comienzo y finalización de todos los objetos con el mismo salón de conferencias para ver si existía conflicto. Si existía, Victor pudo haber sido notificado, permitiéndole hacer los arreglos necesarios.

De otra manera, si Victor únicamente requería cambiar el titulo de la reunión (el atributo TEMA) no habría necesidad de invocar un procedimiento porque el cambio no afectaba a los otros objetos. La filosofía esencial del procedimiento es:

Cuando ciertos atributos son modificados, los métodos pueden ser invocados para ver si cualquier acción deberá ser tomada.

FORMA PARA INVOCAR METODOS

Cuando se comentaron las operaciones en objetos, observamos que un valor es insertado en la ranura de un objeto con la subrutina INSERTAR. La subrutina **INSERTAR** esta compuesta por una serie de instrucciones que localizan un objeto de una clase y posteriormente guardan el valor en una ranura de atributos de el objeto. Por ejemplo:

insertar (sname=AULA, oname=IMP1, inicio=9:00)

creara una serie de instrucciones que primero buscaran en la memoria de la computadora la localidad de la clase AULA y el objeto dentro de la clase llamado IMP1. Entonces, el valor de 9:00 será insertado en la ranura asignada a el atributo INICIO. La pregunta es ¿Cómo se invoca el procedimiento? La respuesta es que una indicación deberá ser colocada en la subrutina **INICIAR** para que cuando un atributo específico tiene un valor asignado a éste, un método deberá ser invocado. Debido a que es la subrutina **INSERTAR** la que cambia el valor, se deduce que esta subrutina estará conectada a el método invocado.

Lo que se debe hacer es hallar un camino para indicarle a la subrutina **INSERTAR** que el atributo cuyo valor se ha cambiado deberá invocar un método. Por ejemplo, cuando cambiemos el atributo hora de comienzo de IMP1, es necesario invocar un método que nos informe de un conflicto posible. Al cambiar el titulo de el atributo TEMA, por otro lado, no deberá invocar un método.

La respuesta es muy sencilla. Cuando en la definición de clase, nosotros consideramos un atributo que debería invocar un método, se realizo lo siguiente:

1. *Preceder el nombre de el atributo con un asterisco.*

2. *Cuando la subrutina **INSERTAR** "encontraba" este atributo, sabia que un método debería ser invocado por el asterisco.*

3. *El nombre del método deberá ser el siguiente atributo.*

Por ejemplo, deseamos invocar un método si la hora de comienzo o de terminación es cambiada. Este método deberá asegurarse que no existe conflicto cuando los empleados contratan el uso de los salones. La forma de implementar esto es utilizando la subrutina **CREA-CLASE**, que contiene lo siguiente:

```
crea-clase
  (
   sname=AULA,
   nattr=7,
   attr=Salón,
   attr=Tema,
   attr=Ponente,
   attr=*Inicio,
   attr=Método,
   attr=*Fin,
   attr=Método
  )
```

Nótese el asterisco antes de INICIO y FIN. A continuación de esos asteriscos hay dos atributos adicionales con la palabra clave METODO (también nótese que el numero total de atributos ha sido incrementado a siete, NATTR=7, como un resultado). Este procedimiento por partes será usado, cuando se crea un objeto de la clase AULA, para especificar que el método será invocado cuando la subrutina **INSERTAR** cambia el valor de INICIO o FIN. Observemos como trabaja. El objeto AULA creado por Victor García será como sigue:

```
crea-objeto
    (
     sname=AULA,
     oname=IMP1,
     Salón=Cristal,
     Tema=Redes Neuronales,
     Ponente=Victor García,
     Inicio=10:00,
     Método=nombre del procedimiento,
     Fin=11:00,
     Método=nombre del procedimiento
    )
```

Si fuésemos a implementar la subrutina **CREA-OBJETO** en BASIC, deberíamos usar un numero en el "nombre del procedimiento" (en otros lenguajes, como el LISP, usaríamos un nombre simbólico como INICIA-PROC). El numero en BASIC deberá ser usado para identificar el numero del método. Consultando los objetos con la subrutina **LISTA-OBJETO** :

```
     Salón=Cristal,
     Tema=Redes Neuronales,
     Ponente=Victor García,
     *Inicio=10:00
     Método=1
     *Fin=11:00
     Método=1
```

El Método=1 indica que el mismo procedimiento es usado cuando la subrutina Inserta cambia tanto la hora de Inicio como la de Fin. Si Victor quisiera cambiar la hora de comienzo, el lo haría con la subrutina **INSERTAR**. Esta invocaría al método 1. Este método checaría todos los objetos con el valor del atributo Salón=Cristal para confirmar si se enciman con otra conferencia en el mismo salón. La secuencia de eventos usados por la subrutina **INSERTAR** para realizar esto es :

1. Hallar la localidad en la memoria de la computadora donde la clase esta localizada. Esta es la clase que nos indica si un asterisco precede un nombre de atributo.

2. Buscar los atributos de la clase que preceden a el nombre de el atributo de los cuales el valor es cambiado.

3. Si existe un asterisco precediendo el nombre, prepararse a invocar a un método.

4. Ahora que sabemos si un atributo de un objeto invoca un método, nosotros podemos localizar el objeto en memoria.

5. Posteriormente, se invoca al método de el atributo que es cambiado si existe un asterisco precediéndolo. Si no existe asterisco, únicamente cambia el valor.

La habilidad para usar la subrutina **INSERTAR** en conjunción con el proceso de invocación es una de las mas importantes razones de la programación orientada a objetos. Debido a que diferentes objetos cumplen diferentes tareas, es importante que los objetos se habiliten con varias responsabilidades para comunicarse entre ellos utilizando un método para enviar mensajes. Este procedimiento es inicialmente declarado cuando el sistema es creado. Por ejemplo, si el método invocado por un cambio en el tiempo del salón de conferencias detecta un traslape con otra conferencia en el mismo salón, se tendrá un conflicto para ambos espacios de tiempo del salón de conferencias, la subrutina **INSERTAR** dirigirá un mensaje a el objeto Ponente.
Este mensaje invocara otro método que mandara un mensaje a la terminal, quien le informara al Ponente del conflicto del traslape.

Una clase es un objeto que se encarga de crear objetos basados en reglas preestablecidas, es decir, crea código y datos correlacionados en función de un modelo. Esto implica que pueden existir objetos de una misma clase. Una clase es una fabrica de objetos.

En Clipper, Existen algunos tipos de objetos predefinidos que son llamados "clases", las cuales se diseñan para soportar operaciones especificas; sin embargo, la creación de nuevas clases o subclases no es soportada por Clipper.

2.1.2 Encapsulación

La encapsulación es la base de la OOP. Su contribución es restringir los efectos de cambio cuando se modifica un sistema o programa, colocando una pared alrededor de cada pieza que conforman los datos. Todo el acceso a los datos es manejado por procedimientos que fueron puestos conjuntamente con ellos para mediar su acceso.

La encapsulación significa que un dato del objeto esta escondido dentro de él mismo y esta protegido.

2.1.3 Herencia

La herencia es la creación de una nueva clase, extensión o especialización de una existente, es decir, la herencia es la manera por la cual un objeto adquiere las propiedades de otro objeto. Permite una relación conceptual entre diferentes clases que están explícitas.

Por ejemplo, la clase **PAGINA** podría ser un descendiente de la clase **Libro**, pero puede incluir algunos métodos o información adicional. En lugar de duplicar las características de las dos clases, podemos decir que **PAGINA** "hereda" su definición de **libro.**
La herencia también permite clases existentes y las bibliotecas de clase para ser fácilmente modificados. Por ejemplo, si una clase no tiene todos los métodos necesarios o si no tiene un modelo que necesite, la herencia puede crear una nueva clase con sus nuevas necesidades. La herencia utiliza el código reusable, es decir, si existe el código no se tiene qué modificar todo.

La herencia puede ser múltiple o única. La herencia única implica características de obtención de sólo una clase de padre. El ejemplo precedente utilizó herencia única.

La herencia múltiple obtiene métodos e información desde múltiples clases. Un ejemplo podría ser un objeto de clase **Mesa Circular** que está derivado de las clases **Mesa** y **Circulo;** tiene disponible todos los métodos de círculos así como los métodos apropiados a Mesas.

Otra característica de la herencia es la habilidad para tratar objetos de diferentes clases como objetos de una misma clase "genérica". Estos nos permiten manipular grupos de objetos desiguales como instancias de una clase más uniforme. En realidad, podemos concebir cada clase como derivación de la clase más genérica. Un ejemplo más realista podría ser una clase genérica llamada **Color** que tiene varias clases derivadas, como **Rojo, Verde y Blanco.** Estas clases derivadas compartirán métodos y características de información del padre. Si tuviéramos un grupo heterogéneo de objetos que comparten una clase de padre, podemos tratarlo como un grupo de objetos homogéneos de la misma clase. Estos nos permiten organizar estos objetos como si fueran del mismo tipo. Por ejemplo, podemos agrupar el objeto Rojo, el objeto Verde y objeto Blanco en una lista ligada de Colores.

La herencia es la parte más innovadora de la OOP por que no es provista por lenguajes convencionales. Es una herramienta para

emitir código automáticamente a las clases desarrolladas por diferentes miembros de un equipo. Los programadores ya no comienzan cada modulo con una página en blanco, en lugar de eso escriben un simple postulado haciendo referencia a alguna clase que ya existiera en la librería. Cada postulado subsecuente describe como la nueva clase difiere de una existente en la librería.

Si por alguna razón se tuviese que agregar un dato o método nuevo al código existente se puede reusar el antiguo código para heredar sus características en la nueva clase que se le sumaran sus propias peculiaridades, y así no tener que volver a escribir un nuevo código para todo el sistema. El efecto resultante es poner la reusabilidad de frente en la corriente principal de el proceso de desarrollo de software.

2.1.4 POLIMORFISMO

El concepto de enviar diferentes mensajes a diferentes tipos de objetos en programación orientada a objetos se le conoce como polimorfismo, es decir, en esencial significa que un mismo nombre puede ser utilizado para especificar una clase genérica. El polimorfismo nos permite enviar mensajes idénticos a diferentes objetos. Objetos diferentes pueden ser referenciados con la misma clase padre. Podemos invocar un método de muestra en cada miembro de una lista vinculada de formas, los objetos individuales de la muestra pueden ser mesas, sillas, etc. Sin embargo, no es necesario que los objetos estén agrupados o relacionados para poder realizar un polimorfismo.

En el nivel más simple, dos objetos no relacionados de dos diferentes clasificaciones pueden recibir cada uno un mando de muestra, y cada uno responderá al mando de diferente forma. En este caso, el polimorfismo es requerido, pero el código específico para estar ejecutado puede realmente ser determinado en la fase de compilación.

En el ejemplo anterior, objetos de diferentes clases estuvieron agrupados juntos como una lista vinculada de apuntadores a objetos de la clase padre. La lista vinculada podría estar construida dinámicamente, en cuyo caso no conoceríamos en la fase de compilación qué código específico estaría ejecutándose para el método de muestra. Esto tendría que estar determinado en la muestra en la corrida para cada invocación de un método en cualquier elemento de la lista vinculada.

La definición de polimorfismo es un poco abstracta, implica la capacidad de enviar diferentes mandos a diferentes objetos que se están agrupando como colecciones heterogéneas descendientes de una clase de base común. Bajo esta definición, la combinación de herencia y la habilidad para determinar el código real en tiempo de ejecución nos permiten desempeñar un polimorfismo. Esta capacidad del polimorfismo es muy poderosa, y frecuentemente es apropiada.

Por ejemplo, se puede tener un programa que defina tres tipos matrices. Una matriz es para valores enteros, una para valores de punto flotante y otra para caracteres. Por tal, debido al polimorfismo se crearan tres tipos de funciones para estas matrices, denominándolas **sumar()** e **imprimir()**, y el compilador escogerá la función correcta de acuerdo con el tipo de dato con que se llame a la función. En dicho ejemplo, la definición general es la de sumar e imprimir datos en una matriz, las funciones definen la manera específica con la que se realizará esto para cada tipo de dato.

2.2 CUESTIONES ACERCA DE OOP CUANDO SE CREA GRANDES SISTEMAS

No hay evidencia recolectada o experimentos realizados para validar las declaraciones hechas para la OOP, especialmente realizados en grandes sistemas. Los más grandes proyectos emprendidos a la fecha parecen ser desarrollos en OOP, pero no todos estos han tenido buenos resultados. Al desarrollarse un sistema, debe analizarse el tipo de problema a resolver y verificar que la OOP sea lo que realmente se necesite, por ejemplo, se utilizo OOP para realizar un sistema de contabilidad y un sistema de reservaciones, los resultados en esta área no fueron los esperados.

Teniendo en cuenta que un objeto es una subrutina de campo lexicográfico con múltiples puntos de entrada y de estado persistente, la OOP ha sido abordada desde que se inventaron las subrutinas en 1940. Los objetos tuvieron un soporte completo en lenguajes tales como: FORTRAN, AED-0, ALGOL. Pero la OOP fue vista como un mal estilo de programación por los aficionados al FORTRAN.

Como Admiral Grace Hooper afirma , "Actualmente no hemos hecho nada nuevo en computación solamente hemos renombrado viejos elementos fundamentales". Admiral Hooper estuvo haciendo OOP en la HARVARD MARK I en 1944 y probablemente no lo supo.

Desafortunadamente hemos ignorado la declaración de RENTSCH: "Esperamos que hayamos aprendido nuestra lección de la PROGRAMACION Clase DA y descubrir primero lo que un termino significa antes de empezar a emplearlo". C++ fue descrito por primera vez en 1980, y diez años después aun no esta completa su definición. De esto preguntamos, ¿ Puede alguna cosa que es demasiado difícil de definir ser buena para la escritura de programas mantenibles y entendibles ?. Y si se piensa que es difícil restringir la definición de un objeto, simplemente trate de dibujar un cuenta o eslabón en la definición de cadenas heredadas que conectan objetos.

Una de las principales características que se adjudica OOP es la facilidad de reusar código.

La unidad de reuso en OOP es la jerarquía, puesto que el objeto es un conjunto indivisible. A diferencia de una liberia de subrutinas en donde se puede tomar solo el código necesario, en OOP se obtiene todo el conjunto y no el simple elemento que se desea. El problema radica en que las jerarquías no son modulares. Y no podemos tomar el objeto de la jerarquía por que no sabemos como los objetos están encajados en la jerarquía. Así vemos que, el costo de el reuso en OOP es mayor por que debemos reusar mucho más código del requerido. El sistema será más grande, correrá más despacio, y costara más el mantenimiento. Aunque puede haber situaciones en las cuales las conveniencias de el programador pesen más que los intereses de el usuario del sistema (casi nunca se da el caso).

Para reducir el costo de reuso de OOP, las jerarquías deben ser pequeñas y se debe hacer una combinación de varias de ellas para entonces construir nuestro programa. Se pueden necesitar por ejemplo jerarquías: de una aproximación polinomial, de listas encadenadas, de comunicaciones, de registros indexados, de menús de salida y copia de líneas, todas al mismo tiempo. Pero ninguno de los lenguajes utilizado por OOP puede enviar argumentos de una jerarquía a otra. No hay en teoría ni en la práctica una manera de combinar jerarquías en OOP. La OOP debe mapear de una representación interna a otra. Después de todo, no hay ninguna razón para sospechar que una representación interna de una jerarquía que pertenezca a un objeto compuesto, tal como una matriz o una figura se parezca a otro objeto. Esto claramente desmerita uno de los principales beneficios de OOP llamado representación interna oculta. Lo que hemos ahorrado al no haber escrito código para los objetos en la misma jerarquía, ahora se debe gastar al escribir código para hacer un mapa entre objetos en diferentes jerarquías. La parte más difícil del trabajo no es obtener

fragmentos de código para trabajar, si no hacerlos que trabajen juntos. El nombre del juego cuando se refiere al reuso de código, es la integración a escala. La OOP hace la construcción de fragmentos de código fácil, pero lo que hace difícil es ponerlos a trabajar integralmente.

2.3 [redacted]

LENGUAJE C++

C++ fue desarrollado por Bjarne Stroustrup. Stroustrup fue un entusiasta usuario de Simula, un lenguaje de simulación que toma muchas de las capacidades de la programación orientada a objetos. Cuando creyó necesario escribir sobre proyecciones en C, tomó características de la programación orientada a objetos de simula y creó el C con clases, mejor conocido como C++.

C++ puede ser definido como un tipo de C con clases. Una clase en C++ (y Simula) es un término para un tipo de dato abstracto. Por supuesto, ya C tiene un modo de agregación de información en un tipo complejo de clase. Una clase es una clase con algunos caracteres diferentes. Primero, parte de la información que puede ser mantenida privada con el único acceso a esta información a través de los " métodos " asociados con la clase. Un método es una función de C, pero es unir a una clase particular y tener acceso a toda la información de clase, (una clase que es totalmente "pública" es la misma que una clase y en realidad puede ser definido como una clase).

El programador puede "heredar" toda la información, y los métodos, de otra clase y añadir lo que sea necesario. Este es particularmente útil para código reusable. En lenguajes como C y otros, si algún código no se encontrara nuestras necesidades, por lo menos tenemos que modificar el código. La herencia nos permite tomar una clase existente y aumentar o modifica como necesario. En realidad, una clase en C++ tiene la capacidad de "herencia múltiple", una clase puede heredar información de más de una clase.

Aunque C++ fue diseñado para ayudar a la gestión de programas muy grandes, no está en absoluto limitado a esta utilización. De hecho, los atributos orientados a objetos de C++ se pueden aplicar con eficiencia para casi cualquier tarea de programación. No es infrecuente ver que C++ se utiliza para proyectos tales como editores, bases de datos, sistemas personales de archivo y

programas de comunicaciones. Además, como C++ comparte la eficiencia de C, gran parte de los programas de sistema de mucha eficiencia se construye utilizando C++.

Como se ha indicado, usando C++ se puede crear una jerarquía de objetos relacionados. Esta posibilidad permite la creación de bibliotecas especiales orientadas a objetos, que pueden ser compartidas por muchos programadores. Por tanto, incluso los programas que pueden gestionarse fácilmente con C pueden escribirse en C++, sólo aprovechar la ventajas ofrecidas por posibilidades que se encuentren en una biblioteca orientada a objetos.

2.3.1 DESARROLLO DE REDES NEURONALEES EN C++

Cuando creamos una clase en C++, podemos sobrecargar los operadores al aplicar esta clase. Esta permite el uso de la programación de alto nivel, con respecto a vectores y matrices, que es muy importante en la instrumentación de redes neurales, podemos sobrecargar operadores aritméticos para trabajar con todos los vectores y matrices a la vez. Podemos codificar A+B para sumar dos matrices en lugar de utilizar una gran cantidad de procedimientos específicos en la codificación. Esto hace que C++ sea un excelente método para algoritmos explicativos. Podemos entonces dispensar con pseudocódigo, que somos de valor discutible en una guía de practicante de cualquier manera. El pseudocódigo es el código mismo. El código está presentado en el mismo vocabulario que la teoría.

También, cuando creemos diferentes tipos de redes neurales utilizaremos el mismo método para todas ellas. Los métodos y codificación de un patrón de asociación es llamando patrón de salida, este patrón permite el entrenamiento de ellos mismos y también que ellos mismos se corran.

En C++ el polimorfismo nos permite utilizar el mismo método y buscar a cada tipo de objeto red neural. Si cambiáramos el tipo de red neural utilizada para instrumentar una aplicación, necesitaríamos solamente cambiar la declaración del objeto de red neural. Todo el resto del código seguirá siendo el mismo.

2.4 REPRESENTACION DE DATOS EN REDES NEURALES

Cuando se implementa una aplicación de un red neural, es importante evaluar debidamente los datos disponibles antes de hacer cualquier compromiso de proyecto. Puede ser que sea necesario convertir los datos a un formato específico para que sean totalmente intelegibles por la red neural.

La aproximación general a la solución de un problema será la misma sin importar que modelo de red neural se ha escogido. Es muy importante la manera en que los datos se representan para la habilidad de una red en la comprensión de un problema. Una red puede aprender más fácilmente a partir de algunas representaciones que de otras.

Los datos pueden ser de valor contínuo o de valor binario. En algunos casos se pueden representar de las dos formas; como un solo valor contínuo o como un conjunto de rangos a los cuales se les asigna un valor binario, por ejemplo las calificaciones o los tipos de letras en las impresoras. En el caso de las calificaciones, estos datos se pueden representar con un valor numérico (contínuo, del **0** al **10**), o por uno de los cinco posibles valores : **MB**, **B**, **S**, **NA** ó **NP**; en el ejemplo de los tipos de letras, se tienen los tipos clasificados por el tamaño (**10 cpi**, **12 cpi**, etc.), por la forma (**Sanserif**, **Courier**) o por ambas representaciones (**Sanserif 12**).

Cuando se tienen grupos de datos de ocurrencia natural (por ejemplo agrupaciones como : **Vertebrados** e **Invertebrados**, **Reino Animal** y **Reino Vegetal**, **Mamíferos** y **Ovíparos**, etc), representaciones binarias son por lo general la mejor opción para hacer las correlaciones. Cuando los valores son muy contínuos, reunirlos en grupos específicos puede ser un error, ya que sería difícil para la red aprender de ejemplos en los cuales se tienen valores dentro o cerca de los límites entre dos grupos (por ejemplo la temperatura : **frío**, **caliente**, **tibio**, **congelado**, etc.).

Un error común es utilizar entradas con valores contínuos para representar conceptor únicos. Por ejemplo, se puede pensar que es perfectamente razonable para representar las horas del día con numeros del **1** al **24**. Sin embargo, la red neural distinguirá

aquellos datos que sean valores contínuos y tengan calificativos como **"Mas o menos"** o **"Mejor o peor"**. A partir de que las **05:00** no es mejor o peor que las **19:00**, se requieren entradas individuales para cada hora del día. El código postal, los meses del año, el estado civil y demas por el estilo, son ejemplos de datos que requieren más de una entrada.

Una importante decisión para representar datos como valores contínuos está en que si se usan cantidades o cambios en las cantidades. Algunos datos, como el índice de oferta-demanda de la venta de un producto, tienen la tendencia a cambiar con el tiempo. El cambio en el índice de un período a otro puede estar, por ejemplo, en un rango de ±50, es decir, en el período anterior estaba a 25150 unidades y en el período actual esta en 25200 unidades; así entonces, es mejor usar el cambio que la cantidad de unidades.

Otra razón para utilizar cambios en las cantidades es que el rango es mas pequeño, las diferencias de valores pequeños son mas comprensibles para la red neural. Usando el cambio en la cantidad es mas fácil de apreciar por la red, debido a que se tiene un mayor porcentaje del rango total posible.

En todo caso se deberá considerar describir la información como artículos o entidades únicas (por ejemplo: gato, tigre o leopardo) o como un conjunto de cualidades (tales como la resolución en un monitor: CGA, EGA, VGA, SVGA, XGA, etc.; o si es Monocromático o de Color). Al tomarse en cuenta la forma en que se debe describir la información para que pueda ser manejada por una red neural, se podrá observar que estas consideraciones se pueden implementar por medio de la programación orientada a objetos (**OOP**), ya que como se mencionó anteriormente, los **objetos** son entidades únicas de información con cualidades específicas de una **clase**.

Se le denomina **representación no-distribuída** a la información que puede ser exclusivamente categorizada como una de varias entidades posibles. Se deberá asignar una neurona a cada cualidad exclusiva, y el dato será **Falso** o **Verdadero** (**0** ó **1**) para cada una. Una desventaja al utilizar información **no-distibuída** esta en que en una red de tamaño razonable solo puede almacenar un número muy limitado de patrones únicos.

La información es **distribuída** cuando las cualidades que definen a un patrón único se extienden sobre muchas piezas de información. Por ejemplo, el objeto **Agua** se puede pensar como dos tercios **Hidrógeno** y un tercio **Oxígeno**. Así entonces, tomando como entradas primarias a los elementos de la tabla periódica, se pueden hacer varias combinaciones posibles para representar los diversos compuestos químicos sin tener que agregar neuronas. Un esquema de entrada distribuída reduce el número de neuronas necesarias para representar un número grande de patrones que

comparten cualidades comunes. Retomando los conceptos de la **OOP**, la representación de información distribuída se puede llevar a cabo por medio de la propiedad de herencia, en donde se tiene la habilidad de la generalización, debido a que de una clase se pueden derivar otras posibles clases. Sin embargo, el problema que se tiene al usar una aproximación distribuída para la salida es que frecuentemente la salida de la red debe ser decodificada dos veces : primero, de las neuronas de activación a las cualidades distribuídas, y segundo, a la identificación no-distibuída. Por ejemplo, si un compuesto químico fué distribuído, un patrón de salida de .5Na y .5Cl deberá ser decodificado por un programa o por un observador externo como **Sal de Mesa** (a menos de que la salida sea una fórmula química). No obstante, la dimensión de este problema estará en relación al tipo de aplicación que se haga y las interfaces **hombre-máquina** que se tengan. Una red con el nivel de salida distribuído tiene menor capacidad de aprendizaje debido a que ésta tiene menor número de conecciones. Una de las ventajas de una red neural con salida distribuída está en que al usar pocas neuronas en los niveles **oculto** y de **salida**, se tienen pocas conecciones, lo que resulta en menos calculos, y por consecuencia, la respuesta es más rápida, que es una característica muy importante en algunas aplicaciones, sobre todo en aplicaciones orientadas a control. Por otro lado, si se tienen muchas salidas, es más difícil entrenar a la red neural para que sea precisa y exacta en todas sus neuronas de salida. Generalizando, son mejores las redes neurales con mayor número de neuronas de entrada que de salida.

En esta sección sólo se presenta la forma en que los datos deben de tratarse (transformarse) para poder ser procesados por una red neuronal; las estructuras de datos que almacenarán la información se diseñarán de acuerdo a la filosofía **OOP** y se presentarán en el capítulo 3.

2.4.1 TRANSFORMACIÓN DE DATOS

Para introducir los datos a la red neural se necesitan dos cosas : un nivel de entrada de neuronas y un esquema de decodificación. La función del algortimo de codificación es tomar los datos de entrada y convertirlos a una forma apropiada para presentarselos a la red. Para aceptar y entender las soluciones se necesitan dos cosas mas : un nivel de salida de neuronas y un esquema de decodificación. En algunos modelos de redes neurales, un mismo nivel sirve tanto de entrada como de salida. Estos modelos son generalmente del tipo de memoria asociativa. El algoritmo de decodificación toma los valores de las neuronas del nivel de salida y los convierte en una respuesta intelegible.

Los algoritmos de codificación y decodificación son específicos de la red neural, pero algunos principios guía se pueden aplicar a todos ellos. Las neuronas operan con entradas y salidas numéricas que corresponden a las razones de disparo (umbrales), o valores de activación, de las neuronas. Por lo general, los datos tienen una forma diferente a representaciones numéricas con valores que esten contenidos en el rango que entienden las neuronas (usualmente de **0** a **1**, ó de **-1** a **1**). Pueden ser símbolos (palabras como **Sí/No**), numeros mas grandes o mas pequeños, o una imágen. La entrada codificadora deberá normalizar los datos, así como convertirlos en una secuencia de valores numéricos que la red pueda entender. Cada número en la secuencia se asignará a un neurona en particular en el nivel de entrada. La salida decodificadora deberá hacer lo contrario; ésta tomará la secuencia de números que corresponden a los valores de las neuronas del nivel de salida y los convertirá a cualquier forma que sea requerida por la salida final. La mayoría de programas de redes neurales realizan automáticamente esta tarea.

Si los datos numéricos tienen un rango natural que esté fuera del rango de operación de las neuronas, deberán de normalizarse. Por ejemplo, si el dato de entrada es un número entre el **1** y el **100**, el rango total es de **100**. Un valor de entrada de **10** es **10/100** (diez centésimos, o un décimo) del rango, ó **.1** en una escala de **0** a **1**. Algunos programas que construyen redes neurales, como el **BrainMaker**, realizan automáticamente la normalización

2.4.2 TIPO Y CANTIDAD DE DATOS

Algunas consideraciones importantes en el entrenemiento radican en el tipo y cantidad de datos recolectados. A continuación se describirán algunos de estos factores.

2.4.2.1 RECOLECCIÓN DE DATOS

No se necesitan fórmulas ni reglas para entrenar a una red neural. únicamente es necesario saber que clase de información es importante para resolver un problema. Si ni se tiene seguridad, hay que incluírlos. Una red neuronal puede aprender a ignorar entradas que tienen poco o nada que ver con el problema, al suministrarle los suficientes ejemplos. Es raro el caso en que se usen muchas clases de datos. Lo mas frecuente es que no se usen los suficientes datos y, por tanto, las correlaciones se vuelvan

difíciles de encontrar. Cuando no se tienen suficientes clases de datos para hacer asociaciones adecuadas, el tiempo de entrenamiento puede ser muy excesivo. Esta situación puede ser evidente con las redes de retropropagación cuando un número muy grande de neuronas ocultas se requieren para entrenar a la red. El problema que se tiene con demasiadas neuronas ocultas está en la memorización. Este método es un síntoma de una red que se entrena bien pero que se prueba pobremente con nuevos datos.

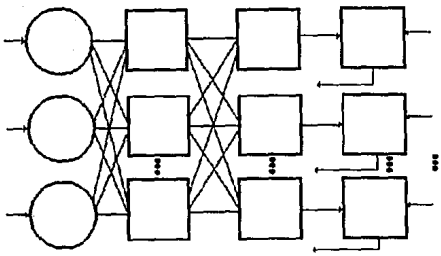
2.4.2.2 ORGANIZACIÓN DE DATOS

Una gran diferencia en como los datos se tienen organizados ocurre entre el modelo **Supervisado**, en el cual los datos de entrenamiento incluyen las salidas asociadas (respuestas conocidas), y el modelo **No-Supervisado**, en donde no se tienen. Si el enfoque de la red neural se orienta a la predicción, a la evaluación, o a la generalización, se deberá usar el modelo **Supervisado**. Las redes neurales supervisadas básicamente aprenden a asociar un conjunto de datos de entrada con un conjunto diferente de salida.

Los modelos No-Supervisados, como la red de **Kohonen**, que es en la que se base en parte la arquitectura de **Contrapropagación**, son mejores en aplicaciones para clasificar o reconocer tipos de problemas. Por ejemplo, se puede almacenar la descripción de algunas personas extraviadas. Todos los datos disponibles se usan como entradas para el entrenamiento. Cuando se encuentra a una persona que por diveros motivos no pueda proporcionar datos que sirvan para identificarlo, la red neural tomará la descripción de esta persona y respondera en su salida con la identificación de una persona almacenada que mas se acerque a los datos de entrada. En la mayoría de los modelos No-Supervisados, el número de ejemplos que pueden ser almacenados es limitado por el número de neuronas. Cuando se utilizan demasidos ejemplos, éstos se interfieren entre sí, y por tanto la habilidad de la red se ve disminuída.

Hay mucho que considerar cuando se empieza a desarrollar una base de datos para una red neuronal. El formato del dato (contínuo, binario, y demás) y el rango de valores representados (patrones frontera y de valor diverso) requieren de cuidadosa atención. Hay que tomar en cuenta que las redes neurales, como la mayoría de los sistemas de cómputo, son muy literales. Estos únicamente entienden lo que uno le dice, no lo que uno piensa.

CAPITULO 3



CAPITULO 3

La descripción del código de una red neuronal, en forma de lenguaje de programación, se le denomina **Neurosoftware.**

David Zipser fue el primero en proponer explícitamente la idea de que un lenguaje formal debe ser creado con el propósito de describir la estructura funcional de las redes neuronales.

El principal atractivo de los lenguajes neurosoftware es que se puede construír la estructura común de todas las redes neurales dentro del lenguaje. Así entonces, cada lenguaje puede ofrecer un incremento significativo en la facilidad de uso y eficiencia sobre las expresiones, es decir, las descripciones escritas en un lenguaje optimizado para la expressión de algorítmos (tales como el **C, LISP, C++, PASCAL,** etc).

Los lenguajes neurosoftware generalmente se construyen en torno a un modelo de red neural. Para la construcción del lenguaje alrededor de un modelo, el programador es libre de tener que expresar repetidamente el conocimineto estructural contenido en el modelo. El uso de un modelo también simplifica la edición y modificación de una red, puesto que únicamente los cambios específicos por sí mismos necesitan ser corregidos, y no los efectos colaterales de estos cambios. En realidad, los lenguajes para redes neurales se prestan para los metodos de ***Orientación a Objetos***, que actualmente se utilizan para eliminar las deficiencias del desarrollo tradicional de Software (Programación Estructurada).

La mayoría de los lenguajes neurosoftware existentes (como el sistema P3 de Zipser) se construyen sobre un modelo restrictivo, asumiendo así que cada red esta constituída de niveles o capas de combinaciones afines con las funciones de activación en sus salidas (como en la red de Retropropagación). Sin embargo, un par de lenguajes (como Candela y AXON, que se presentará en la siguiente sección) se construyen alrededor de modelos muy generales.

3.1 Lenguaje AXON

A continuación se presenta una introducción del lenguaje AXON para la descripción de redes neurales. Este lenguaje fue desarrollado como un tipo de **Pseudocódigo** específico para la construcción de programas de aplicación de redes neurales; posteriormente este peseudocódigo puede ser traducido a cualquier lenguaje de programación que se deseé.
2
La sintáxis del AXON se basa en el lenguaje de programación C.
Las Palabras reservadas usadas en los enunciados en lenguaje AXON se enlistan en la Tabla 3.1. Además de estas palabras, los nombres de las funciones interconstruídas en AXON y los nombres de las rutinas que se encuentran en la librería de funciones matemáticas del AXON también son reservados. Por tanto, estos identificadores no podrán ser utilizados como nombres de variables, funciones o constantes en los programas que se construyan en Pseudocódigo AXON.

En AXON, cada palabra reservada, función, variable constante puede estar delimitada por un espacio en blanco o por un signo de puntuación. Un espacio en blanco usado dentro de una constante de tipo cadena (string) no será visto como un delimitador, sino ue será tratado como cualquier otro caracter.

Las palabras reservadas, los nombres de las funciones interconstruídas, y los nombres de las funciones matemáticas de AXON pueden escribirse con mayusculas o con minusculas, pero no con una combinación de ambos tipos de letras.

El código en AXON que describe una red neural se divide en cinco bloques :

1) Definición de Parámetros.
2) Elementos de Procesamiento y Definición de Capas o Niveles.
3) Creación de la Red y Definición de las Conexiones.
4) Función Programa.
5) Definición de Funciones.

Estos bloques se describen en la siguiente sección usando la descripción en AXON de una red de Retropropagación de tres capas y una red de Contrapropagación.

block	**break**	**case**
char	**class**	**create**
connect	**data**	**default**
do	**domain**	**else**
enum	**external**	**for**
function	**global**	**input**
include	**init**	**local**
layer	**list**	**operate**
net	**of**	**repeat**
random	**real**	**signal**
selection	**show**	**struct**
state	**static**	**unsigned**
type	**union**	**update**
using	**void**	**when**
while		

Tabla 3.1 Palabras reservadas del lenguaje AXON

3.1.1

Programa en AXON que describe una red neuronal con arquitectura Retropropagación de tres niveles.

```
/* COMIENZA DEFINICION DE PARAMETROS */


type record tCtsBpn
  record lt
        int iCnt, hCnt, oCnt, RandomSeed, ConnectInputs;
        real InitWeightMax;
  end lt;
```

```
  record rt
        real HiddenAlpha, OutputAlpha, HiddenBeta, OutputBeta;
        int BatchSize, LinearOutput, LearnFlag;
  end rt;
end tCtsBpn;

type enum tPass
    FORWARD, BACKWARD, FINAL
end enum tPass;

int HidSunCnt, HidPltCnt, OutSunCnt, OutPltCnt;


/* FIN DEL BLOQUE DE DEFINICION DE PARAMETROS */



/* COMIENZA ELEMENTOS PROCESADORES Y DEFINICION DE NIVELES */


net Bpn(tCtsBpn Cts)
   pe PeIn
       state : real x;
   end PeIn;

   pe PeHidPlt
       state : real xw;
       transfer function HiddenPlanetXfer(tPass);
       input class list  F1, F2;
           signal of F1 : real x;
           signal of F2 : real HiddenError;
       local memory
          weight of F1 : real w;
   end PeHidPlt;



   pe PeHidSun
       state : real z;
       transfer function HiddenSunXfer(tPass);
       input class list  F3, F4;
           signal of F3 : real zw;
           signal of F4 : real ew;
       local memory
           data : real I;
    end PeHidSun;

   pe PeOutPlt
       state : real zw_or_ew;
       transfer function OutputPlanetXfer(tPass);
       input class list  F5, F6;
           signal of F5 : real z;
           signal of F6 : real OutputError;
       local memory
           weight of F5 : real w;
   end PeOutPlt;

   pe PeOutSun
       state : real y';
```

```
      transfer function OutputSunXfer(tPass);
      input class list  F7, F8;
          signal of F7 : real zw;
          signal of F8 : real y;
      local memory
          data : real I;
          data : real tmp;
   end PeOutSun;

   slab  PeIn      SlabOne[1];
   slab  PeIn      SlabTrn[OutSunCnt];
   slab  PeHidSun SlabHidSun[HidSunCnt];
   slab  PeIn      SlabIn[Cts.lt.iCnt];
   slab  PeOutSun SlabOutSun[OutSunCnt];
   slab  PeHidPlt SlabHidPlt[HidPltCnt];
   slab  PeOutPlt SlabOutPlt[OutPltCntl;

   create  function  BpnSetup();
   operate function  BpnRun();
end net Bpn;

/* FIN DEL BLOQUE DE ELEMENTOS DE PROCESADORES Y DEFINICION DE NIVELES */




/* COMIENZA CREACION DE LA RED Y DEFINICION DE LAS CONECCIONES */


create function BpnSetup()
   domain           OneAndInput;
   domain           OneAndHidden;
   domain PeHidPlt HPdomain;
   domain PeOutPlt OPdomain;

   int iPe, OutPltPerSun, HidPltPerSun;

   if Cts.lt.iCnt < 1 /* Checa si el tamaño de la salida es razonable */
     then
          Bpn.error = -101;
          return;
   end;

   if Cts.lt.hCnt < 0 /* Checa si el tamaño de la capa oculta es razonable */
     then
          Bpn.error = -102;
          return;
   end;

   if Cts.lt.oCnt < 1 /* Checa si el tamaño de la salida es razonable */
     then
          Bpn.error = -103;
          return;
   end;

   HidSunCnt = Cts.lt.hCnt;
   OutSunCnt = Cts.lt.oCnt;
```

```
   HidPltPerSun = Cts.lt.iCnt + 1;
   HidPltCnt    = HidSunCnt * HidPltPerSun;

   if Cts.lt.ConnectInputs <> 0
     then
          OutPltPerSun = Cts.lt.hCnt + Cts.lt.iCnt + 1;
     else
          OutPltPerSun = Cts.lt.hCnt + 1;
   end;

   OutPltCnt = OutSunCnt * OutPltPerSun;
   build Bpn;    /* Crea las estructuras de la Red */
   OneAndInput = SlabOne + SlabIn;

   iPe = 0;
   for _Pe in SlabHidSun do
       HPdomain = SlabHidPlt[iPe:iPe + HidPitPerSun - 1:1];
       connect class F1 of HPdomain from OneAndInput using one_to_one();
       connect class F2 of HPdomain from _Pe using full();
       connect class F3 of _Pe from HPdomain using full():
       iPe += HidPltPerSun;
   end;

   if Cts.lt.ConnectInputs <> 0
     then
          OneAndHidden = SlabOne + SlabHidSun + SlabIn;
     else
          OneAndHidden = SlabOne + SiabHidSun;
   end;

   iPe = 0;
   for _Pe in SlabOutSun do
       OPdomain = SlabOutPlt[iPe + 1:iPe + HidSunCnt:1];
       connect class F4 of SlabHidSun from OPdomain using one_to_one():
       OPdomain = SlabOutPlt[iPe:iPe + OutPltPerSun - 1:1];
       connect class F5 of OPdomain from OneAndHidden using one_to_one();
       connect class F6 of OPdomain from _Pe using full();
       connect class F7 of _Pe from OPdomain using full();
       iPe += OutPltPerSun;
   end;

   connect class F8 of SlabOutSun from SlabTrn using one_to_one();

   fsrand(Cts.lt.RandomSeed);
   for _Pe in SlabHidPlt do
      for Icn in _Pe.F1 do
          Icn.w = ((2 * frand()) - 1) * Cts.lt.InitWeightMax;
      end;
   end;

   for _Pe  in SlabOutPlt  do
      for Icn in _Pe.F5 do
          Icn.w = ((2 * frand()) - 1) * Cts.lt.InitWeightMax;
      end ;
   end;

   SlabOne[0]. x =  1. 0;
end BpnSetup;

/* FIN DEL BLOQUE DE CREACION DE LA RED Y DEFINICION DE LAS CONECCIONES */
```

```
/* COMIENZO DE LA FUNCION DE PROGRAMA */

operate function BpnRun()

/***** Paso de adelanto(Forward) *****/

    update SlabHidPlt (FORWARD);
    update SlabHidSun (FORWARD);
    update SlabOutPlt (FORWARD);
    update SlabOutSun (FORWARD);

/ ************************************/

/****** Paso de Retraso (Backward) *****/

    if Cts.rt.LearnFlag <> 0
      then
           update SlabOutSun(BACKWARD);
           update SlabOutPlt(BACKWARD);
           update SlabHidSun(BACKWARD);
           update SlabHidPlt(BACKWARD);
           update SlabOutSun(FlNAL);
    end;

/ ************************************/

end BpnRun;

/* FIN DEL BLOQUE DE LA FUNCION DE PROGRAMA */



/* COMIENZA DEFINICION DE LAS FUNCIONES DE TRANSFERENCIA */

transfer function HiddenPlanetXfer(tPass pass)
 case pass of
  when FORWARD:
               for Icn in this_pe.F1 do
                    this_pe.xw = Icn.x * Icn.w;
               end;
  when BACKWARD:
                for IcnF1 in this_pe.F1, IcnF2 in this_pe.F2 do
                 IcnF1.w += Cts.rt.HiddenAlpha * IcnF2.HiddenError * IcnF1.z;
                end;
 end case;
end HiddenPlanetXfer;


transfer function HiddenSunXfer(tPass pass)  real error;
 case pass of
  when FORWARD:
               this_pe.I = 0.0;
               for _Icn in this_pe.F3 do
                   this_pe.I += _ICn.zw;
               end;
               this_pe.z = logistic(this_pe.I);
  when BACKWARD:
```

```
                error = 0.0;
                for Icn in this_pe.F4 do
                    error += Icn.ew;
                end;
                this_pe.z = (1.0 - this_pe.z) * this_pe.z * error;
 end case;
end HiddenSunXfer;

transfer function OutputPlanetXfer(tPass pass)
 case pass of
  when FORWARD:
              for Icn in this_pe.F5 do
                  this_pe.zw_or_ew = Icn.z * Icn.w;
              end;
  when BACKWARD:
               for IcnF5 in this_pe.F5, IcnF6 in this_pe.F6 do
                this_pe.zu_or_ew = IcnF5.w * IcnF6,OutputError;
                IcnF5.w += Cts.rt.OutputAlpha * IcnF6.OutputError * Icnf5.z;
               end;
 end case;
end OutputPlanetXfer;


transfer function OutputSunXfer(tPass pass)
 case pass of
  when FORWARD:
              this_pe.I = 0.0;
              for Icn in this_pe.F7 do
                  this_pe.I += Icn.zw;
              end;
              if Cts.rt.LinearOutput == 0
                then
                     this_pe.z = logistic(this_pe.I);
                else
                     this_pe.z = this_pe.I;
              end;
  when BACKWARD:
               this_pe.tmp = this_pe.z;
               for Icn in this_pe.F8 do
                   this_pe.z = Icn.y - this_pe.z;
               end;
               if Cts.rt.LinearOutput == 0
                 then
                      this_pe.z *= (1.0 - this_pe.tmp) * this_pe.tmp;
                 end;
  when FINAL:
            this_pe.z = this_pe.tmp;
 end case;
end OutputSunXfer;

/* FIN DEL BLOQUE DE DEFINICION DE LAS FUNCIONES DE TRANSFERENCIA */
```

3.1.2

Programa en AXON que describe una red neuronal con arquitectura de Contrapropagación.

```
/* COMIENZA DEFINICION DE PARAMETROS */


type record tCtsCpn
       record lt   /*LoadTime*/
          int  InputSize, GrossbergSize, KohonenSize, RandomSeed;
          real InitWtMin, InitWtMax;
       end lt;

       record rt /*RunTime*/
          real Alpha, Beta, a, b, c, d, r, T, Reject;
          int Winners, StatsFlag, CountFlag, LearnFlag, WinRatioFlag;
       end rt;
end tCtsCpn;


type enum
    PASS1, PASS2, PASS3, PASS4, PASS5, PASS6;
end tPassCpn;


type real tStsCpn;
type real tWtsCpn;
type int tStsMinCpn;
type int tStsCntCpn;


type record rStsMminCpn
        real MinDistance, MinValue;
end tStsMminCpn;


type enum
    INVALID_IN_SIZE  = -101,
    INVALID_KOH_SIZE = -102,
    INVALID_GSB_SIZE = -103;
end tErrorCpn;

/* FIN DEL BLOQUE DE DEFINICION DE PARAMETROS */



/* COMIENZA ELEMENTOS DE PROCESADORES Y DEFINICION DE NIVELES */


net Cpn(tCtsCpn Cts)
    pe PeInCpn
        state : tStsCpn x;
```

```
end;

pe PeKohCpn
    state : tStsCpn z;
    transfer function  KohXfer(tPassCpn);
    input class list FromIn, FromMin, FromMmin, FromSum;
        signal of FromIn   : tStsCpn x;
        signal of FromMin  : tStsMinCpn MinPeIdx;
        signal of FromMmin : tStsMminCpn Mmin;
        signal of FromSum  : tStsCpn Sum;
    local memory
        weight of FromIn : tWtsCpn w;
        data : real d;
        data : real p;
        data : real b;
        data : real tp;
    end;

pe PeGsbCpn
     state : tStsCpn y;
     transfer function GsbXfer(tPassCpn);
     input class list FromKoh, FromTrn;
         signal of FromKoh : tStsCpn z;
         signal of FromTrn : tStsCpn t;
     local memory
         weight of FromKoh : tWtsCpn u;
end;

pe PeMinCpn
    state : tStsMinCpn MinPeIdx;
    transfer function  MinXfer();
    input class list  FromKoh;
        signal of FromKoh  tStsCpn z;
    local memory
        data : int UnbiasedWinner;
        data : int cSame;
        data : int cTotal;
        data : real PercentSame;
end;

pe PeMminCpn
    state : tStsMminCpn MultiMin;
    transfer function  MminXfer();
    input class list FromKoh;
        signal of FromKoh : tStsCpn z;
end;


pe PeSumCpn
    state : tStsCpn Sum;
    transfer function SumXfer();
    input class list FromKoh;
        signal of FromKoh : tStsCpn z;
end;

pe PeStatCpn
    state : tStsCpn Stat;
    transfer function StatXfer();
    input class list FromGsb, FromTrn;
        signal of FromGsb : tStsCpn y;
        signal of FromTrn : tStsCpn t;
    local memory
```

```
            data : real Sum;
            data : real cIter;
    end;

    pe PeCntCpn
        state : tStsCntCpn Count;
        transfer function CntXfer();
        input class list FromKoh;
            signal of FromKoh : tStsCpn z;
    end;

    slab PeInCpn   SlabInCpn[Cts.lt.InputSize];
    slab PeKohCpn  SlabKohCpn[Cts.lt.KohonenSize];
    slab PeGsbCpn  SlabGsbCpn[Cts.lt.GrossbergSize];
    slab PeInCpn   SlabTrnCpn[Cts.lt.GrossbergSize];
    slab PeMinCpn  SlabMinCpn[1];
    slab PeMminCpn SlabMminCpn[1];
    slab PeSumCpn  SlabSumCpn[1];
    slab PeStatCpn SlabStatCpn[2];
    slab PeCntCpn  SlabCntCpn[1];

    create  function CpnSetup()
    operate function CpnRun();
end net Cpn;

/* FIN DEL BLOQUE DE ELEMENTOS PROCESADORES Y DEFINICION DE NIVELES */


/* COMIENZA CREACION DE LA RED Y DEFINICION DE LAS CONECCIONES */


create function CpnSetup()
    if Cts.lt.InputSize < 1
      then
           Cpn.error = INVALID_IN_SIZE;
           return;
    end;

if Cts.lt.KohonenSize < 1
  then
       Cpn.error = INVALID_KOH_SIZE;
end;

if Cts.lt.GrossbergSize < 1
  then
       Cpn. error = INVALID_GSB_SIZE;
       return;
end;

build Cpn;

connect class FromIn   of SlabKohCpn  from SlabInCpn   using full();
connect class FromMin  of SlabKohCpn  from SlabMinCpn  using full();
connect class FromMmin of SlabKohCpn  from SlabMminCpn using full();
connect class FromSum  of SlabKohCpn  from SlabSumCpn  using full();
connect class FromKoh  of SlabGsbCpn  from SlabKohCpn  using full();
connect class FromTrn  of SlabGsbCpn  from SlabTrnCpn  using one_to_one();
connect class FromKoh  of SlabMinCpn  from SlabKohCpn  using full();
connect class FromKoh  of SlabMminCpn from SlabKohCpn  using full();
connect class PromKoh  of SlabSumCpn  from SlabKohCpn  using full();
```

```
connect class PromKoh  of SlabCntCpn  from SlabKohCpn  using full();
connect class FromGsb  of SlabStatCpn from SlabGsbCpn  using full();
connect class FromTrn  of SlabStatCpn from SlabTrnCpn  using full();

fsrand(Cts.lt.RandomSeed);

for Pe_ in SlabKohCpn do
     Pe_.b = 0.0;
     Pe_.p = 1.0 / Cts.lt.KohonenSize;
     Pe_.tp = Pe_.p;
     for Icn in Pe_.FromIn do
          Icn.w = (CtS.lt.InitWtMax - Cts.It.InitWtMin) * frand() + Cts.lt.I-
nitWtMin;
     end;
end;
for Pe_ in SlabGsbCpn do
    for Icn in Pe_.FromKoh do
         Icn.u = (Cts.lt.InitWtMax - Cts.lt.InitWtMin) * frand() + Cts.lt.I-
nitWtMin;
    end;
end;

SlabStatCpn[0].Sum   = 0.0;
SlabStatCpn[0].cIter = 0.0;
SlabStatCpn[1].Sum   = 0.0;
SlabStatCpn[1].cIter = 0.0;
SlabMinCpn[0].cSame  = 0.0;
SlabMinCpn[0].cTotal = 0.0;

end CpnSetup;

/* FIN DEL BLOQUE DE CREACION DE LA RED Y DEFINICION DE LAS CONECCIONES */

/* COMIENZO DE LA FUNCION DE PROGRAMA */


operate function CpnRun()

    update SlabKohCpn(PASS1);

    if Cts.rt.LearnFlag <> 0 then
        update SlabMinCpn(PASS1);
        update SlabKohCpn(PASS2);
        update rrabGsbCpn(PASS2);
        update SiabKohCpn(PASS3):
        update SlabMinCpn(PASS2);
        update SlabKohCpn(PASS4);
    elseif Cts.rt.CountFlag <> 0 then
        update SlabCntCpn();
    elseif Cts.rt.Winners == 1 then
        update SlabMinCpn(PASS1);
        update SlabKohCpn(PASS2);
        update SlabGsbCpn(PASS1);
   else
        update SlabMminCpn();
        update SlabKohCpn(PASS5);
        update SiabSumCpn();
        update SlabKohCpn(PASS6);
        update SlabGsbCpn(PASS1):
    end;

    if Cts.rt.StatsFlag <> 0 then
```

```
        update SlabStatCpn();
    end;
end CpnRun;

/* FIN DEL BLOQUE DE LA FUNCION DE PROGRAMA */


/* COMIENZA DEFINICION DE LAS FUNCIONES DE TRANSFERENCIA */

transfer function KohXfer(tPassCpn Pass)
    int pe_index();
    real Scale;

case Pass of
     when PASS1:
         this_pe.z = 0.0;
         for Icn in this_pe FromIn do
             this_pe.z += (Icn.x - Icn.w) * (Icn.x - Icn.w);
     end;
         this_pe.z = sqrt(this_pe.z);
         this_pe.d = this_pe.z;
     when PASS2:
         for Icn in this_pe.FromMin do
             if pe_index(this_pe) == Icn.MinPeIdx then
                 this_pe.z = 1.0;
             else
                 this_pe.z = 0.0;
             end;
         end;
     when PASS3:
         if this_pe.p  Cts.rt.T then
             this_pe.z = Cts.rt.d;
         else
             this_pe.z a this_pe.d - this_pe.b;
         end;
     when PASS4:
         for Icn in this_pe.FromMin do
             if pe_index(this_pe) == Icn.MinPeIdx then
                  this_pe.p += Cts.rt.b * (1.0 - this_pe.p);
                  Scale = Cts.rt.Alpha;
             else
                  this_pe.p += Cts.rt.b * (0.0 - this_pe.p);
                  Scale = Cts.rt.Bsta;
             end;
         end;
         this_pe.b = Cts.rt.c * (this_pe.tp - this_pe.p);
         if Scale > 0.0 then
             for Icn in this_pe.FromIn do
                  Icn.w += Scale * (Icn.x - Icn.w);
             end;
         end;
     when PASS5:
         for Icn in this_pe.FromMmin do
           if this_pe.z <= Icn.Mmin.MinValue then
               if Icn.Mmin.MinDistance == 0.0 then
                   if this_pe.z == 0.0 then '
                       this_pe.z = 1.0;
                   else
```

```
                    this,pe.z = 0.0;
                end;
            else
                this_pe.z = Icn.Mmin.MinDistance / this,pe.z;
                    if Cts.rt.r <> 1.0 then
                        this_pe.z = pow(this_pe.z, Cts.rt.r);
                    end;
                end;
            else
                this,pe.z = 0.0;
            end;
        end;
     when PASS6:
        for Icn in this_pe.FromSum do
          this_pe.z = this_pe.z / Icn.Sum;
        end;
   end;
end;

transfer function GabXfer(tPassCpn Pass)
    this,pe.y = 0.0;
    for Icn in this_pe.FromKoh do
        this_pe.y += Icn.z * Icn.u;
    end;
    if Pass == PASS2 then
        for IcnTrn in this_pe.FromTrn do
            for IcnKoh in this_pe.FromKoh do
                if IcnKoh.z == 1.0 then
                  IcnKoh.u += Cts.rt.a * (IcnTrn.t - IcnKoh.u );
                end;
            end;
        end;
    end;
end;


transfer function MinXfer( tPassCpn Pass)
    tSYsCpn MinVal;
    int pe_index();
    for 6Iccn in this_pe.FromKoh do
        MinVal < Icn.z;
        this_pe.MinPefdx = pe_index(fcn.from_pe);
        break;
    end;
    for Icn in this_pe.FromKoh do
        if Icn.z  MinVal then
           MinVal = Icn.z;
           this_pe.MinPefdx = pe_index(fcn.from_pe);
        end;
    end;
    if (Cts.rt.Reject >= 0) && (MinVal > Cts.rt.Reject) then
        this_pe.MinPefdx = -1;
    end;
    if Pass == PASS1 then
        this_pe.UnbiasedWinner = this_pe.MinPefdx;
    elseif Cts.rt.HinRatioFlag > 0 then
        this,pe.cTotal++;
        if this_pe.UnbiasedWinner == this_pe.MinPefdx then
            this,pe.cSame++;
        end;
  this_pe.PercentSame = (real) this_pe.cSame / this_pe.cTotal;
    end
```

```
end;

transfer function MminXfer()
    tStaCpn     MinVal;
    tStsCpn     Floor;
    tStsCpn     NotMin;
    int cOccur;
    int cWinners;


    for Icn in this_pe.FromKoh do
        MinVal = Icn.z;
        NotMin = Icn.z;
        break;
   end;
   cOccur = 0;
   for Icn in this_pe.Fromkoh do
       if Icn.z  MinVal then
          MinVal = Icn.z;
          cOccur = 1;
       elseif fcn.z == MinVal then
          cOccur++;
      else
          NotMin = Icn.z;
      end;
  end;
  cWinners += cOccur;
  this_pe.Multittin.MlnDistance = MinVal;
  uhile cwinners < Cts.rt.Winners do
      cOccur = 0;
      Floor  = MinVal;
      MinVar = NotMin;
      for Icn in this_pe.Fromkoh do
          if (Icn.z > Floor) && (Icn.z < MinVal) then
              MinVal = Icn.z;
              cOccUr = 1;
          elseif Icn.z == MinVal then
              cOccur++;
          else
                NotMin = Icn.z;
          end;
      end;
      cWinners += cOccur;
      lf (Cts.rt.Reject >= 0) && (MinVal > Cts.rt.Reject) then
            MinVal = Cts.rt.Reject;
            break;
      end;
   end;
   this_pe.MultiMin.MinValue = MinVal;
end;


transfer function SumXfer()
   this_Pe.Sum = 0.0;
   for Icn in this_pe.Fromkoh do
        this_pe.Sum += Icn.z;
    end;
end;


transfer function StatXfer()
```

```
    real fabs();
    int pe_index();

    this_pe.cIter += 1.0;
    if pe_index(this_pe) == 0 then
        for IcnGsb in this_pe.FromGsb, IcnTrn in this_pe.FromTrn
            this_pe.Sum  += (IcnTrn.t - IcnGab.y) * (IcnTrn.t - IcnGab.y);
    end;
  else
    for IcnGsb in this_pe.FromGsb, IcnTrn in this_pe.FromTrn do
        this_pe.Sum += fabs(IcnTrn.t - IcnGsb.y);
    end;
  end;
  this_pe.Stat = this_pe.Sum  / this_pe.cIter;
end;


transfer function Cntxfer()
    this_pe.Count = 0;
    for Icn in this_pe.Fromkoh do
       if Icn.x <= Cts.rt.Reject then
          this_pe.Count++;
       end;
    end;
end;

/* FIN DEL BLOQUE DE DEFINICION DE LAS FUNCIONES DE TRANSFERENCIA */
```

3.2 CODIFICACION EN C++ DE LOS MODELOS DE REDES NEURALES.

A partir de diversos tipos de pseudocódigo con enfoque a redes neurales, como el Lenguaje AXON que se presentó en la sección anterior, se han desarrollado algoritmos codificados en varios lenguajes neurosoftware especializados para computadoras (también especializadas), en centros de investigación de Universidades (p.e. Cambridge) e institutos científicos y tecnológicos (p.e. el centro de investigación de SIEMENS, en Alemania, y la NASA , en EUA). Sin embargo, por consecuencia del nuevo auge que está tomando la investigación y, sobre todo, el potencial de aplicación de las redes neurales, se ha logrado desarrollar software mas práctico que puede usarse en equipos de cómputo convencionales, de costo menor, y por ende, de mayor uso actualmente en muchas industrias. Este tipo de neurosoftware se ha codificado en los lenguajes de programación de mayor uso en el desarrollo de sistemas, como son el **C**, el **C++** y ocasionalmente el **Pascal**.

La mayoría de estos algoritmos se han implementado de acuerdo a la aplicación en que serán utilizados. Sin embargo, algunos programadores han creado algoritmos de uso general, proporcionando al usuario el código fuente con el fin de permitirle la libertad de modificarlo de acuerdo a sus propias necesidades. Algo muy especial en este sentido, es que el código fuente se proporciona de manera gratuita en la mayoría de los casos y sin ningún derecho de autor; extraordinario en verdad, si se toma en cuenta los excelentes resultados (incluyendo los económicos, por supuesto) que se obtienen al aplicar las redes neurales.

Como ya se mencionó, el código se proporciona para que el usuario lo modifique, por lo que la generalidad de este tipo de código debe ser muy extensa. Por tal motivo, **Adam Blum** (entre otros), desarrolló los algoritmos de los principales modelos de redes neurales en el lenguaje **C++**, que como ya se vió en el capítulo II, está enfocado a la programación orientada a objetos. La OOP nos permite crear clases muy generales de objetos, a partir de las cuales se pueden obtener objetos de una clase muy particular, gracias a la propiedad de herencia. Así entonces, dichas clases las puede obtener el usuario del neurosoftware a partir de sus propias necesidades de acuerdo a su aplicación. Por tanto, este tipo de neurosoftware es muy abierto, que es una característica esperada y muy bien recibida por quienes deseen aplicar el concepto de red neural, aún para aquellos que no tengan un amplio conocimiento al respecto.

En la aplicación que se propone en el capítulo V, dichos algoritmos se modifican de acuerdo a las necesidades del usuario final. En este capítulo solo se mostrará el código correspondiente a los algoritmos de **Retropropagación** y de **Contrapropagación**, por ser los de mayor uso y que reúnen las características necesarias para el campo de la Ingeniería de Control, que es en donde se desea aplicar las redes neurales.

Además de los algoritmos presentados aquí, en el Apéndice **A** se muestran los algoritmos desarrollados por la NASA (en Lenguaje **C**), y por otros desarrolladores de neurosoftware, a fin de dar una mejor y mas amplia visión del gran interés que existe por las redes neurales. Estos algoritmos se obtuvieron vía modem de un correeo electrónico de manera gratuita.

ALGORTIMOS GENERALES PARA CUALQUIER MODELO DE RED NEURONAL.

```
/* ////////////////////////////////////////////////
//
//                VECMAT.HPP
//                CLASES VECTOR Y MATRIZ
//
//////////////////////////////////////////////// */


#include <stdlib.h>
#include <io.h>
#include <conio.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <ctype.h>
#include <math.h>
#include <time.h>
#include <float.h>
#include <sys\stat.h>

#ifdef __TURBOC__
   #include <alloc.h>

#elif defined(__ZTC__)
   #include <dos.h>

#endif

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>


/* C++ no tiene min/max */
#define max(a,b)  ( ( (a) > (b) ) ? (a) : (b) )
#define min(a,b)  ( ( (a) < (b) ) ? (a) : (b) )

/* #include "debug.h"*/

double logistic(double activation);
/* cambiará los valores mucho más grandes que estos*/
const ROWS=64;  /* longitud del primer patrón*/
const COLS=64;  /* longitud del segundo patrón */
const MAXVEC=64; /* tamaño de vectores por omisión*/

class matrix;

class vec
        {
          friend class net;
          friend class bp;
          friend ostream& operator <<(ostream& s, vec& v1);
          #ifdef __TURBOC__
                  friend ostream far& operator << (ostream far& s,vec far&
```

```
v1);
        #endif
        friend class matrix;
        friend istream& operator >>(istream& s, vec& v1);
        int n;
        float *v;

        public :
                vec(int size=MAXVEC, int val=0);   /* constructor*/
                ~vec();                            /* destructor*/
                vec(vec &v1);  /* inicializador de copia*/
                int length();
                float distance(vec& A);
                vec& normalize();
                vec& normalizeon();
                vec& scale(vec& minvec, vec& maxvec);

                /* producto punto del vector y su complemento*/

                float d_logistic();
                float maxval();
                vec& garble(float noise);
                vec& operator =(const vec& v1);
                vec operator +(const vec& v1);
                vec operator +(const float d);
                /* tarea de suma de vector*/
                vec& operator +=(const vec& v1);
               /* suministro de integridad*/
               /* pero no se usaran ahora*/
               /* multiplicacion de un vector por una constante*/
               vec& operator *=(float c);
               /* la multiplicación del vector transpuesto necesita acceso al arreglo v*/
                int operator ==(const vec& v1);
                float operator [](int x);
                int vec :: maxindex();
                vec& getstr(char *s);
                void putstr(char *s);

                vec operator -(const vec& v1);   /* substracción*/
                vec operator -(const float d);   /* substracción*/
                float operator *(const vec& v1); /* producto punto*/
                vec operator *(float c); /* multiplicación por una constante*/
                vec& sigmoid();
        }; /* clase vector*/



class vecpair;

class matrix
        {
          friend class hop;
          friend class tsp;
          friend class net;
          friend class bp;

          friend ostream& operator <<(ostream& s, matrix& m1);
          friend istream& operator >>(istream& s, matrix& m1);

          protected :
```

```
                float **m; /* la representación matricial*/
                int r,c;   /* numero de renglones y columnas*/

        public :
                /* constructores*/

                matrix(int n=ROWS, int p=COLS, float range=0);
                matrix(int n, int p, float value, float range);
                matrix(int n, int p, char *fn);
                matrix(const vecpair& vp);
                matrix(matrix& m1); /* inicializador de copia*/
                ~matrix();
                int depth();
                int width();
                matrix& operator =(const matrix& m1);
                matrix& operator +(const matrix& m1);

                vec operator *(vec& v1);
                vec colslice(int col);
                vec rowslice(int row);
                void insertcol(vec& v, int col);
                void insertrow(vec& v, int row);
                int closestcol(vec& v);
                int closestrow(vec& v);
                int closestrow(vec& v, int *wins, float scaling);
                int load(int fh);
                int save(int fh);

                matrix& operator +=(const matrix& m1);
                matrix& operator *(const float d);
                matrix& operator *=(const float d);
                void initvals(const vec& v1, const vec& v2, const float
rate=1.0, const float momentum=0.0);

        }; /* clase matriz*/


class vecpair
        {
        friend class net;
        friend class bp;
        friend class matrix;
        friend ifstream& operator >>(ifstream& s, vecpair& v1);
        friend ostream& operator <<(ostream& s, vecpair& v1);
        friend matrix :: matrix(const vecpair& vp);
                /*bandera que señala si sucede la codificación*/
        int flag;

        public :
                vec *a;
                vec *b;
                vecpair(int n=ROWS, int p=COLS);  /*constructor*/
                vecpair(vec& A, vec& B);
                vecpair(const vecpair& AB); /* inicializador de copia*/
                ~vecpair();
                vecpair& operator=(const vecpair& v1);
                int operator==(const vecpair& v1);
                vecpair& scale(vecpair& minvecs, vecpair& maxvecs);
        }; /* clase par-vector*/
```

```
/*////////////////////////////////////
//
//              VECMAT.CPP
//
// METODOS PARA LAS CLASES VECTOR Y MATRIZ */
//
#include "vecmat.hpp"
/*////////////////////////////////////////
```

```
// funciones miembro de la clase VECTOR*/

vec::vec(int size, int val)
  {
   v=new float[n=size];
   for(int i=0;i<n;i++)
       v[i]=val;
  }   /*/constructor*/

vec::~vec()
   {
    delete v;
   }   /*/ destructor*/


vec::vec(vec& v1)  /*/inicializador-copia*/
  {
    v=new float[n=v1.n];
    for(int i=0;i<n;i++)
       v[i]=v1.v[i];
  }


vec& vec::operator=(const vec& v1)
  {
    delete v;
    v= new float[n=v1.n];
    for(int i=0;i<n;i++)
       v[i]=v1.v[i];
    return *this;
  }



vec vec::operator+(const vec& v1)
  {
    vec sum(v1.n);
    for(int i=0;i<v1.n;i++)
      sum.v[i]=v1.v[i]+v[i];
    return sum;
  }

vec vec::operator+(const float d)
  {
    vec sum(n);
```

```
    for(int i=0;i<n;i++)
      sum.v[i]=v[i]+d;
    return sum;
  }

vec& vec::operator+=(const vec& v1)
  {
    for(int i=0;i<v1.n;i++)
       v[i]+=v1.v[i];
    return *this;
  }

float vec::operator*(const vec& v1) /*/ producto punto*/
  {
    float sum=0;
    for(int i=0;i<min(n,v1.n);i++)
       sum+=(v1.v[i]*v[i]);
    /*/ D(cout << " producto punto" << *this << v1 << sum << "\n";)*/
    return sum;
  }

int vec::operator==(const vec& v1)
   {
     if(v1.n!=n)
       return 0;
     for(int i=0;i<min(n,v1.n);i++)
       if(v1.v[i]!=v[i])
          return 0;
     return 1;
   }

int vec::length()
    {
     return n;
    }     /*/ metodo de longitud*/

float vec::operator[](int x)
   {
     if(x<length() && x>=0)
       return v[x];
     else
       cerr << "índice del vector fuera de rango";
     return 0;
   }


vec& vec::garble(float noise)  /*/ vector corrompido w/ruido aleatorio*/
   {
     time_t t;
     time(&t);
     srand((unsigned)t);
     for(int i=0;i<n;i++)
       if( (rand()%10)/10<noise)
         v[i]=1-v[i];
     return *this;
   }

vec& vec::normalize()   /*/ normalizado por longitud*/
   {
     for(int i=0;i<n;i++)
        v[i]/=n;
```

```
    return *this;
  }

vec& vec::normalizeon() /*/ normalizado por elementos no-cero*/
  {
    int on=0;
    for(int i=0;i<n;i++)
      if(v[i])
          on++;
    for(i=0;i<n;i++)
      v[i]/=on;
    return *this;
  }

float vec::maxval() /*/ devuelve el máximo valor absoluto*/
  {
    float mx=0;

    for(int i=0;i<n;i++)
      if(fabs(v[i])>mx)
         mx=fabs(v[i]);
    return mx;
  }

vec& vec::scale(vec& minvec,vec& maxvec)
  {
    for(int i=0;i<n;i++)
    if(v[i]<minvec.v[i])
      v[i]=0;
    else
      if(v[i]>maxvec.v[i])
           v[i]=1;
      else
           if( (maxvec.v[i] - minvec.v[i])==0)
             v[i]=1;
           else
              v[i]= (v[i]-minvec.v[i])/(maxvec.v[i]-minvec.v[i]);
    return *this;
  }


float vec::d_logistic() /*/ devuelve vec * (1-vec)*/
  {
    float sum=0.0;

    for(int i=0;i<n;i++)
      sum+=(v[i]*(1-v[i]));
    return sum;
  }

/*/ función de distancia Euclidiana || A - B ||*/

float vec::distance(vec& A)
  {
    float sum=0.0,d;

    for(int i=0;i<n;i++)
      {
      d=v[i]-A.v[i];
      if(d)
        sum+=pow(d,2);
```

```
    return sum?pow(sum,0.5): 0;
  }

/*/ índice del ítem más alto en el vector*/

int vec::maxindex()
  {
    int idx,i,mx;
    for(i=0,mx=-INT_MAX;i<n;i++)
     if(v[i]>mx)
       {
         mx=v[i];
         idx=i;
       }
    return idx;
  }


double logistic(double activation)
  {
  /* ESTOS LIMITES DE BAJOFLUJO SE COPIARON DE LA IMPLEMENTACION BP DE McCLELLAND. */

     if(activation>11.5129)
         return 0.99999;
     if(activation<-11.5129)
         return 0.00001;
     return 1.0/(1.0+exp(-activation));
  }

vec& vec::getstr(char *s)
  {
    for(int i=0;i<MAXVEC&&s[i];i++)
       if(isalpha(s[i]))
          v[toupper(s[i])-'A']=1;
    return *this;
  }

void vec::putstr(char *s)
  {
    int ct=0;
    for(int i=0;i<26;i++)
     if(v[i]>0.9)
        s[ct++]='A'+i;
  }

vec vec::operator-(const vec& v1)
  {
    vec diff(n);

    for(int i=0;i<n;i++)
       diff.v[i]=v[i]-v1.v[i];
    return diff;
  }

vec vec::operator-(const float d)  /*/ substracción de constante*/
  {
    vec diff(n);

    for(int i=0;i<n;i++)
       diff.v[i]=v[i]-d;
    return diff;
```

```
    return diff;
  }

vec vec::operator*(float c)
  {
    vec prod(length());

    for(int i=0;i<prod.n;i++)
       prod.v[i]=v[i]*c;
    return prod;
  }

vec& vec::operator*=(float c)
  {
    for(int i=0;i<n;i++)
       v[i]*=c;
    return *this;
  }  /*/ multiplicación del vector por una constante*/

vec& vec::sigmoid()
  {
    for(int i=0;i<n;i++)
       v[i]=(float)logistic((double)v[i]);
    return *this;
  }


istream& operator>>(istream& s, vec& v1)
  {        /*/ formato : lista de numeros de punto flotante seguidos por ','*/

    float d;
    int i=0,c;

    for(;;)
         {
           s>>d;
           if(s.eof())
            return s;
           if(s.fail())
           {
             s.clear();
             do
               c=s.get();
             while(c!=',');
             return s;
           }
           v1.v[i++]=d;
           if(i==v1.n)
           {
             do
               c=s.get();
             while(c!=',');
             return s;
           }
         }
  }


ostream& operator<<(ostream& s, vec& v1)
      /*/ formato: lista de numeros de punto flotante seguidos por ','*/
  {
```

```
    s.precision(2);
    for(int i=0;i<v1.n;i++)
       s<<v1[i]<<" ";
    s<<",";
    return s;
  }

/*//////////////////////////////////////
// funciones miembro de la clase MATRIZ*/

matrix::matrix(int n, int p, float range)
  {
    int i,j,rnd;
    time_t t;
    int pct,val;

    m=new float *[n];
    if(range)
      {
      time(&t);
      srand((unsigned)t);
      }
    for(i=0;i<n;i++)
     {
       m[i]=new float[p];
       for(j=0;j<p;j++)
           if(range)
           {
             rnd=rand();
             pct=(int)(range * 100.0);
             val=rnd%pct;
             m[i][j]=(float)val/100.0;
             if(range<0)
               m[i][j]=fabs(range)-(m[i][j]*2.0);
           }
           else
             m[i][j]=0;
     }
    r=n;
    c=p;
  }


matrix::matrix(int n, int p, float value, float range)
  {
    int i,j;

    m=new float *[n];
    for(i=0;i<n;i++)
     {
       m[i]=new float[p];
       for(j=0;j<p;j++)
            if(range)
            m[i][j]=value;
     }
    r=n;
    c=p;
  }


matrix::matrix(int n, int p, char *fn)
```

```
    {
      int i,j, rnd;
      time_t t;

      m= new float *[n];
      for(i=0;i<n;i++)
        m[i]=new float(p);
      r=n;
      c=p;
      ifstream in(fn, ios::in);
      in >> *this;
    }


matrix::matrix(const vecpair& vp)
    {
      r=vp.a->length();
      c=vp.b->length();
      m=new float *[r];
      for(int i=0;i<r;i++)
       {
         m[i]=new float[c];
         for(int j=0;j<c;j++)
             m[i][j]=vp.a->v[i]*vp.b->v[j];
       }
    }  /*/ constructor*/


matrix::matrix(matrix& m1)   // inicializador de copia
    {
    /*/D(cout << "inicializador de copia de la matriz \n";*/

      r=m1.r;
      c=m1.c;
      m=new float *[r];
      for(int i=0;i<r;i++)
       {
         m[i]=new float[c];
         for(int j=0;j<c;j++)
             m[i][j]=m1.m[i][j];
       }
    }


matrix::~matrix()
    {
      for(int i=0;i<r;i++)
        delete m[i];
      delete m;
    }   /*/ destructor*/

matrix& matrix::operator=(const matrix& m1)
    {
      for(int i=0;i<r;i++)
        delete m[i];
      r=m1.r;
      c=m1.c;
      m=new float*[r];
      for(i=0;i<r;i++)
       {
         m[i]=new float[c];
         for(int j=0;j<r;j++)
```

```
            m[i][j]=m1.m[i][j];
      }
     return *this;
   }


matrix& matrix::operator+(const matrix& m1)
   {
     int i,j;
     matrix sum(r,c);

     for(i=0;i<r;i++)
      for(j=0;j<r;j++)
          sum.m[i][j]=m1.m[i][j]+m[i][j];
     return sum;
   }

matrix& matrix::operator*(const float d)
   {
     int i,j;
     for(i=0;i<r;i++)
      for(j=0;j<r;j++)
         m[i][j]*=d;
     return *this;
   }

vec matrix::colslice(int col)
   {
     vec temp(r);

     for(int i=0;i<r;i++)
       temp.v[i]=m[i][col];
     return temp;
   }

vec matrix::rowslice(int row)
   {
     vec temp(c);
     for(int i=0;i<r;i++)
       temp.v[i]=m[row][i];
     return temp;
   }

void matrix::insertcol(vec& v, int col)
   {
     for(int i=0;i<v.n;i++)
      m[i][col]=v.v[i];
   }

void matrix::insertrow(vec& v, int row)
   {
     for(int i=0;i<v.n;i++)
      m[row][i]=v.v[i];
   }

int matrix::depth()
   {
     return r;
   }

int matrix::width()
   {
```

```
    return c;
  }

int matrix::closestcol(vec& v)
  {
    int mincol;
    float d;
    float mindist=INT_MAX;
    vec w(r);          /* verificar si va c ó r */

    for(int i=0;i<c;i++)
     {
       w=colslice(i);
       if( (d=v.distance(w)) < mindist)
         {
           mindist=d;
           mincol=i;
         }
     }
    return mincol;
  }

int matrix::closestrow(vec& v)
  {
    int minrow;
    float d;
    float mindist=INT_MAX;
    vec w(c);

    for(int i=0;i<r;i++)
     {
       w=rowslice(i);
       if( (d=v.distance(w)) < mindist)
         {
           mindist=d;
           minrow=i;
         }
     }
    return minrow;
  }


int matrix::closestrow(vec& v, int *wins, float scaling)
  {
    int minrow;
    float d;
    float mindist=INT_MAX;
    vec w(c);

    for(int i=0;i<r;i++)
     {
       w=rowslice(i);
       d=v.distance(w);
       d*=(1+((float)wins[i]*scaling));
       if(d<mindist)
         {
           mindist=d;
           minrow=i;
         }
     }
    return minrow;
  }
```

/*/ guarda los valores binarios de la matriz en el archivo raw especificado*/

```
int matrix::save(int fh)
   {
     int success=1;

     for(int i=0;i<r;i++)
      for(int j=0;j<c;j++)
           write(fh,&(m[i][j]),sizeof(m[0][0]));
     return success;
   }
```

/*/ carga los valores binarios de la matriz desde el archivo raw especificado*/

```
int matrix::load(int fh)
   {
     int success=1;

     for(int i=0;i<r;i++)
      for(int j=0; j<c;j++)
          if(read(fh,&(m[i][j]),sizeof(m[0][0]))<0)
             success=0;
     return success;
   }


#if MULTIWINNER

int _Cdecl intcmp(const void* i1,const void*i2);
int _Cdecl intcmp(const void* i1,const void*i2)
   {
     if(*i1>*i2)
      return 1;
     if(*i1<*i2)
      return -1;
     return 0;
   }



void matrix::closestrows(vec& v, int *rows)
   {
     int dist[r];
     vec w(c);

     for(int i=0;i<r;i++)
      {
        w=rowslice(i);
        dist[i]=v.distance(w);
      }
     qsort(dist,r,sizeof(int),intcmp);
   }

#endif


matrix& matrix::operator+=(const matrix& m1)
   {
     int i,j;
```

```
    for(i=0;i<r&&i<m1.r;i++)
     for(j=0;j<c&&j<m1.c;j++)
          m[i][j]+=(m1.m[i][j]);
    return *this;
  }


matrix& matrix::operator*=(const float d)
  {
    int i,j;
    for(i=0;i<r;i++)
     for(j=0;j<c;j++)
         m[i][j]*=d;
    return *this;
  }


vec matrix::operator*(vec& v1)
  {
    vec temp(v1.n==r?c:r), temp2(v1.n==r?r:c);

    for(int i=0;i<((v1.n==r)?c:r);i++)
     {
       if(v1.n==r)
           temp2=colslice(i);
       else
           temp2=rowslice(i);
       temp.v[i]=v1*temp2;
     }
    return temp;
  }


void matrix::initvals(const vec& v1,const vec& v2,const float rate,
                      const float momentum)
  {
    for(int i=0;i<r;i++)
     for(int j=0;j<c;j++)
          m[i][j]=(m[i][j]*momentum)+((v1.v[i]*v2.v[j])*rate);
  }



istream& operator>>(istream& s,matrix& m1)
  {
    for(int i=0;i<m1.r;i++)
     for(int j=0;j<m1.c;j++)
         s>>m1.m[i][j];
    retun  s;
  }


ostream& operator<<(ostream& s,matrix& m1)   // imprime una matriz
  {
    for(int i=0;i<m1.r;i++)
     {
       for(int j=0;j<m1.c;j++)
           s<<m1.m[i][j]<<" ";
       s<<"\n";
     }
    return s;
  }
```

```
////////////////////////////////////////
//// funciones miembro de la clase VECPAIR
////
//// constructor
////
///


vecpair::vecpair(int n, int p)
    {
      a=new vec(n);
      b=new vec(p);
      /*#else
      #endif*/
    }


vecpair::vecpair(vec& A, vec& B)
    {
      a=new vec(A.length());
      *a=A;
      b=new vec(B.length());
      *b=B;
    }


vecpair::vecpair(const vecpair& AB)
    {
      *this=vecpair(*(AB,a),*(AB,b));
    }

vecpair::~vecpair()
    {
      delete a;
      delete b;
    };              /*/ destructor*/

vecpair& vecpair::operator=(const vecpair& v1)
    {
      *a=*(v1.a);
      *b=*(v1.b);
      return *this;
    }


vecpair& vecpair::scale(vecpair& minvecs, vecpair& maxvecs)
    {
      a->scale(*(minvecs.a),*(maxvecs.a));
      b->scale(*(minvecs.b),*(maxvecs.b));
      return *this;
    }


int vecpair::operator==(const vecpair& v1)
    {
      return(*a==*(v1.a))&&(*b==*(v1.b));
    }


ifstream& operator>>(ifstream& s, vecpair &v1)    /*/entrada a un vector pair*/
    {
```

```
    s>>*(v1.a)>>*(v1.b);
    return s;
  }

ostream& operator<<(ostream& s, vecpair &v1)   /*/ imprime un vector pair*/
  {
    return s<<*(v1.a)<<*(v1.b)<<"\n";
  }
```

[redacted]

```
// NET.HPP
// ARCHIVO HEADER PARA LA CLASE BASE "RED NEURONAL GENERAL"
// PARA SER USADO COMO PADRE PARA ESPECIFICAR LAS IMPLEMENTACIONES DE UNA RED NEURONAL
// LOS METODOS PARA CODIFICADO Y LLAMAR SON DEFINIDOS COMO PURAS FUNCIONES VIRTUALES,
// HACIENDO DE ESTOS UNA CLASE ABSTRACTA QUE NUNCA PUEDE SER ...
// LOS DETALLES DEL CODIFICADO Y DEL LLAMADO PUEDEN DEPENDER DE LA PROPIA TOPOLOGIA.
// SIN EMBARGO, LOS METODOS "ENTRENAR", "PRUEBA" Y "CORRER" PUEDEN DEFINIRSE DESDE
// QUE ELLOS SON SUBSTANCIALMENTE LOS MISMOS PARA CADA UNA DE LAS CLASES. EL CONSTRUCTOR
// PUEDE SER DEFINIDO Y PUEDE SER USADO POR LAS CLASES DESCENDIENTES("HIJOS")
// EN SUS PROPIOS CONSTRUCTORES PARA...
// ELEMENTOS COMUNES DE LAS CLASES DERIVADAS.
```

[redacted]

```
#include "vecmat.cpp"

// CLASE PARAMETRO USADA PARA APUNTAR HACIA LA VARIABLE A SER INICIALIZADA
// Y UNA CADENA ESPECIFICA PARA SER USADA PARA INICIALIZAR EL ARCHIVO DEFINCION.


enum vartype {real, integer, string};
const NAMELEN=16;

class parm
        {
         char *name; /* cadena para valor inicial */
         void *var;
         vartype type;

         public :
                parm(char *s, void *v, vartype vt)
                   {
                   name=new char[strlen(s)];
                   memcpy(name, s, sizeof(name));
                   var=v;
                   type=vt;
                   }
                ~parm()
```

```
                {
                delete name;
                }
        };
```

```
// CLASE PARA LA TABLA PARAMETRO
// LA TABLA PARAMETRO ES... DESDE EL ARCHIVO DEFINICION
// (.DEF FILE) EL CUAL CONTIENE TODOS LOS NOMBRES DE PARAMETROS
// SEGUIDOS POR EL VALOR DEL PARAMETRO GENERALMENTE UNO A LA VEZ....


class parmtable
             {
            friend istream& operator >> (istream& s, parmtable& p);
            parm **entry;
            int noparms;

            public :
                   parmtable(int n, parm *p);
                   parmtable(int n, parm *p, char *name);
                   ~parmtable()
                              {
                              for(int i=0;i<noparms;i++)
                                    delete entry[i];
                              delete entry;
                              }
             };



////////////////////////////////////////////////
///
///         CLASE NET
///
////////////////////////////////////////////////

class net
        {
          friend class vec;
          friend class matrix;
          friend class vecpair;

          protected :
                  char *name; // cadena usada como nombre-base para archivos
                  int n;                                  // tamaño del nivel de entrada
                  int p;                                  // tamaño del nivel de salida
                  int q;
                              // razon de aprendizaje ( definido como 1 donde no es gradual)
                  float learnrate;
                              // decaimiento ( por omision construido a cero si no es aplicable)
                  float decayrate;
                  int iters;
                  int cycleno;
                  vecpair *minvecs;
                  vecpair *maxvecs;


                              // los metodos privados, puesto que se desconoce la topología
                              // deben ser virtuales puros
                  virtual int saveweights()=0;
```

```
            virtual int loadweights()=0;
            int skipcmt(ifstream& s);

    public :
            enum parmtype
                    {
                     inputs, outputs, learn, decay
                    };
            net() {};
            net(char *s);
            net(char *s,int noparms, parm *p);
            ~net();

                        // la codificación y el llamado y la "virtual pura" los cuales
                        // hacen la clase abstracta net
            virtual int encode(vecpair& v)=0;
            virtual vec recall(vec& v)=0;
            virtual float cycle(ifstream& s);
            virtual int train();

                    // el valor de punto flotante indica el porcentaje correcto de la prueba
            virtual float test();
            virtual int run();
    };
```

// NET.CPP
// CODIGO FUENTE PARA LA CLASE BASE DE LA RED NEURONAL ABSTRACTA

```
#include "net.hpp"
#include <string.h>


////////////////////////////////////////////
//
//          CLASE TABLA DE PARAMETROS
//
//          METODOS DE LA TABLA DE PARAMETROS
//
////////////////////////////////////////////



parmtable::parmtable(int n, parm *p)
   {

     noparms=n;
     for(int i=0;i<noparms;i++)
         entry[i]=p++;
   }


parmtable::parmtable(int n, parm *p, char *name)
   {
     char fn[16];
```

```
    sprintf(fn,"%s.DEF",name);
    ifstream def(fn,ios::in);
    if(!def)
     {
       cerr << "\nError, no se encuentra el archivo definido";
       return;
     }
    while(def.read(p,sizeof(p)))
  }


~parmtable()
          {
            for(int i=0;i<noparms;i++)
            delete entry[i];
            delete entry;
          }*/


istream& operator >> (istream& s, parmtable& p)
  {
    char keyword[NAMELEN];
    s >> keyword;
    for(int i=0;i<p.noparms;i++)
     if( !strcmp(keyword,p[i].name) )
         break;
    if(i==noparms)
      {
      cerr << "\n Palabra clave errónea";
      return s;
      }
    switch(p[i].type)
      {
      case string  :  s >> ((char *)p[i].var);    break;
      case integer :  s >> * ((int *)p[i].var);   break;
      case real    :  s >> * ((float *)p[i].var); break;
      }
    return s;
  }



////////////////////////////////////////////////////
//
//              CLASE RED
//
//       METODOS DE LA CLASE RED NEURAL GENERAL
//
////////////////////////////////////////////////////


net::net(char *s)
  {
    char fn[16];
    name = new char[strlen(s)+1];
    strcpy(name,s);
    const NOPARMS=5;
    parm a[NOPARMS] =
                  {
                  parm("ENTRADAS",    (void *)&n,integer),
```

```
                    parm("SALIDAS" ,     (void *)&p,integer),
                    parm("RAZON"   ,     (void *)&learnrate,real),
                    parm("DESCENSO",     (void *)decay,real),
                    parm("ITERACIONES", (void *)&iters,integer)
                    };
    parmtable p(NOPARMS,a,name);
    return;
  }


int net::train()
  {

    char fn[16];
    ifstream *s;

    if(loadweights())
      cout << "\nEntrenamiento desde los pesos almacenados";
    sprintf(fn,"%s.FCT",name);
    cout << "\nEntrenamiento desde " << fn  << ". Presione una tecla para
terminar...";
    for(;;)
      {
      s=new ifstream(fn,ios::in);
      if(!*s)
        {
          cout << "\nError al abrir el archivo real.";
          return 0;
        }
      cout << "\nCiclo " << ++cycleno << ": ";
      float ret=cycle(*s);
      delete s;

      if(ret>=1.0 || kbhit() )
         {
           cout << "\nEntrenemiento suspendido en " << cycleno << " ciclos.";
           break;
         }
      }
    saveweights();
  }


float net::cycle(ifstream& s)
  {
    vecpair v(n,q);
    float good, total;

    s >> *minvecs;
    s >> *maxvecs;

    skipcmt(s);
    for(;;)
      {
      s >> v;

      if(s.eof() || s.fail()) break;
      v.scale(*minvecs, *maxvecs);
      if(encode(v))
        {
          good++;
          /*if(!trace)  cout << '.';*/
```

```
        }
      else
          /*if(!trace)  cout << 'x';*/

      total++;
      if(kbhit()) {return 1.0;}
      }

    return good/total;
  }

int net:: skipcmt(ifstream& inf)
  {
    int c;

    inf.unsetf(inf.skipws);
    if(inf.peek()==':')
      {
      do
        {
          c=inf.get();
          if(c<0)  return 0;
        }
      while( (c!=0xd) && (c!=0xa) );
      inf.setf(inf.skipws);
      return 1;
      }
    else
      {
        inf.setf(inf.skipws);
        return 0;
      }
  }
```

3.2.2 [redacted]

ALGORTIMO DE RETROPROPAGACION.

```
// BP.HPP
// HEADER PARA LA IMPLEMENTACION DE LA RETROPROPAGACION


#include "net.cpp"


class bp : public net
                {                                   // red derivada de la retropropagación

                  friend parmtable::parmtable(int n, parm *p, char *name);

                  friend class net;

                  private :
                        int q;                      // tamaño de el nivel oculto
                        matrix *W1, *W2;            // matrices de pesos de la sinapsis
                        matrix *dW1, *dW2;          // usadas para calcular los cambios en
                                                    las matrices
                        vec *h, *o, *d, *e, *thresh1, *thresh2;
```

```
                int epoch;
                vec *totd, *tote;
                vecpair *minvecs, *maxvecs;
                float momentum, initrange;

                                                // funciones privadas
                                                // estas son funciones de ayuda

                void initvals(matrix& m, const vec& v1, const vec&
v2,
                              const float rate=1.0, const float
momentum=0.0);
                int bp::saveweights();
                int bp::loadweights();

        public :                                // funciones publicas
                bp(char *s);        // constructor basado en el archivo <nombre>.DEF
                ~bp();                          // destructor

                                        // prevalecimiento de funciones virtuales puras

                int encode(vecpair& v); // almacenamiento de un par patrón

                                // rellamamiento y patrón de salida dada una entrada

                vec recall(vec& v);

                float cycle(ifstream& s);
                float test();
                int run();

        };
```

// BP.CPP
// IMPLEMENTACION DE UNA RED CON RETROPROPAGACION

```
#include "bp.hpp"

extern int trace;

bp::bp(char *s):net(s)                                  // CONSTRUCTOR
{
        const NOPARMS=4;
        parm parms[NOPARMS] = {
                        parm("HIDDEN", (void *)&q, integer),
                        parm("MOMEMTUM", (void *)&momentum, real),
                        parm("INITRANGE", (void *)&initrange, real),
                        parm("EPOCH", (void *)&epoch, integer)
                        };
```

```
            parmtable ptbl(NOPARMS,parms,name);

                    // INICIALIZA AMBOS PESOS DE MATRICES A LOS VALORES ALEATORIOS DESDE -1 A +1

                W1=new matrix(n,p,-initrange);
                W2=new matrix(p,q,-initrange);
                dW1=new matrix(n,p);
                dW2=new matrix(p,q);

                h=new vec(p);
                o=new vec(q);
                d=new vec(q);
                e=new vec(p);

                thresh1=new vec(p);

                                                        //thresh1->randomize(initrange);
                thresh2=new vec(q);

                                                        //thresh2=randomize(initrange);

                if (epoch) {
                        totd=new vec(q);
                        tote=new vec(p);
                        }

                minvecs=new vecpair(n,q);
                maxvecs=new vecpair(n,q);

                cycleno=0;
}

bp::~bp()
{
                delete W1;
                delete W2;

                delete dW1;
                delete dW2;

                delete h;
                delete o;
                delete d;
                delete e;

                if (epoch){
                        delete totd;
                        delete tote;
                      }

                delete minvecs;
                delete maxvecs;
}

//////////////////////////////////////////////////////////////////
//
// algoritmos de los metodos de retropropagación - codificado y llamado
//
//////////////////////////////////////////////////////////////////
```

```
int bp::encode(vecpair& v)
{
        float maxdiff,tolerance=0.0001;
                                                        // paso 1)  desde el texto: h=F (W1 i)
       *h = (*W1) * (*(v.a));
                // encuentra el vector que es el producto punto de la entrada y la matrix peso
                                                        // aplica la activaciónsigmoidal al resultado
       h->sigmoid();

                                                        // paso 2)  desde el texto: o=F (W2 h)
       *o=(*W2)*(*h);

       if (trace)
            {
              cout << " encuadramiento Guess: " << *o;
            }

       o->sigmoid();

       if (epoch)                                       // ajuste de pesos al final del ciclo
            {
                                        //paso 3) desde el texto: d = 0 (1-o) (o-t)
                                        // algo que los codigos de circuitos es que se usa
                                        // existiendo sobre carga de los operadores desde el
                                        // el vector clase

            *d = (*(v.b) - *o);

            if (trace)
            cout.precision(2);
                  cout << "\nSalida: " << *(v.b) << " Guess " << *o;

            maxdiff=d->maxval();
            *d = *d * o->d_logistic();

                                                        // paso 4 desde el texto: e = b (1-b) W d
            *e = ((*W2) * *d) *
                                                        // returns el producto punto de vec & complement
                          h->d_logistic();

                                        // los pesos podrian ser ajustados al final del ciclo
                                        // con los seguimientos totales
            *totd += *d;
            *tote += *e;

                                        // entrenamiento patrón-patrón
       }

       else
         {
                                        // paso 3 desde el texto: d = o (1-o) (o-t)
                                        // algo que los codigos de circuitos es que se usa
                                        // existiendo sobre carga de los operadores desde el
                                        // el vector clase

            *d = (*(v.b) - *o);

            cout.precision(2);
                  if (trace)
                  cout << "\nSalida: " << *(v.b) << "Guess: " << *o;
            maxdiff=d->maxval();
```

```
            *d = *d * o->d_logistic();

                                                    // paso 4) desde el texto: e = b (1-b) W2 d

            *e = ((*W2) * *d) *                     // returna el producto punto de el vec & complement

                        h->d_logistic();

                                                    // paso 5) W2 = W2 +  h d +  W2(i-1)
                                                    // " h d " part

            initvals(*dW2, (*h), *d,learnrate,momentum);

            (*W2) += *dW2;

            *thresh2 += ( (*d) * learnrate);
                                                    // Paso 6) W1 = W1 +  i e + W2(i-1)
            initvals(*dW1, *(v.a), *e,learnrate,momentum);
            (*W1) += *dW1;

            *thresh1 += ( (*e) * learnrate);
      }
    if (maxdiff < tolerance)
            {
              return 1;
            }
    else
            {
               return 0;
            }
}

void bp::initvals(matrix& m, const vec& v1, const vec& v2, const float rate,
const float momentum)

                                    // usada al inicializarse una matrix al producto de vectores
                                    // de v1 y v2
                                    // tambien adiciona en los anteriores contenidos de la matrix
                                    // multiplicado por un termino momentum.

{
        for(int i=0; i<m.r; i++)
              for(int j=0; j<m.c; j++)
                    m.m[i][j]=( (m.m[i][j]*momentum)+(v1.v[i]*v2.v[j])*ra-
te);
}

vec bp::recall(vec& v)
{
                    // paso 1) h = F(W1 i)
                    // busca el vector que es el producto punto de la entrada y la matrix peso
// aplicando la activación de la función sigmoide al resultado
        *h=(*W1)*v;
        h->sigmoid();

                                    // paso 2) o = F (W2 h)

        vec out(this->q);
        out=(*W2)*(*h);
        out.sigmoid();
```

```
        return out;
}


float bp::cycle(ifstream& s)
   {
            vecpair v(n,q);
            float good,total;

            s >> *minvecs;
            s >> *maxvecs;

            skipcmt(s);
            for(;;)
            {
                 s >> v;

                 if(s.eof() || s.fail())
                     break;
                 v.scale(*minvecs,*maxvecs);
                 if(encode(v))
                       {
                           good++;
                           if(!trace)
                                 cout << '.';
                       }
                 else
                       if(!trace)
                             cout << 'x';
                 total++;
                 if(kbhit())
                    {
                    return 1.0;
                    }
            }
            if (epoch)                                  // ajuste de pesos al final del ciclo

             {   // W2=W2 + α h d (total)

                 dW2->initvals( (*h),*totd,learnrate);        //" α h d part
                 (*W2) += *dW2;

                 *thresh2 += ( (*totd) * learnrate);

                                                              //W1=W1+α i e (total)

                 dW1->initvals(*(v.a),*tote,learnrate);
                 (*W1)+=*dW1;

                 *thresh1+=( (*tote) * learnrate);
             }
            cout << "\n" << good/total*100 <<"porciento correcto. \n";
            return good/total;
    }



float bp::test()
   {
```

```
    float good=0,total=0,tolerance=0.0001;
    char tstfn[32];
    vecpair v;
    vec out;


    if(!loadweights())
      {
      cout << "No existe red almacenada para prueba";
      return 0;
      }

    sprintf(tstfn,"%s.TST",name);
    ifstream tstf(tstfn,ios::in);

                                                        // comentario skip
    skipcmt(tstf);
    for(;;)
     {
     if(!(tstf>>v))
        break;
     out=recall(*(v.a));
     if( (*(v.b)-out).maxval()<tolerance)
         good++;
     total++;
     }
    cout << good/total << "porciento correcto. \n";
    return good/total;
  }

int bp::run()
  {
    char ifn[16],ofn[16];
    int c;
    vec in(n),out(q);

    if(!loadweights())
      {
      cout << "No existe red almacenada para ejecutar(correr). \n";
      return 0;
      }
    sprintf(ifn,"%s.IN",name);
    sprintf(ofn,"%s.OUT",name);
    cout << "Corriendo desde " << ifn << "\n";
    cout << "Salida hacia " << ofn << "\n";
    ifstream inf(ifn,ios::in);
    ofstream outf(ofn,ios::out);

    skipcmt(inf);
    for(;;)
      {
      if(!(inf>>in))
        break;
      if(!inf || inf.eof() || inf.fail())
          break;
      outf << recall(in);
      }
    return 1;
  }
```

```
////////////////////////////////////////////////
///
/// METODOS DE RETROPROPAGACION A NIVEL ENTRADA/SALIDA
///
/// Salvando y cargando pesos, skipping comentarios
///
////////////////////////////////////////////////

int bp::saveweights()
   {
     int fh;
     char fn[32];

     sprintf(fn,"%s.WTS",name);
     #ifdef __TURBOC__
           chmod(fn,S_IREAD|S_IWRITE);
     #elif defined(__ZTC__)
           dos_setfileattr(fn,FA_NORMAL);
     #endif

     fh=open(fn,O_CREAT|O_TRUNC|O_BINARY);
     if(!fh)
       return 0;
     write(fh,&cycleno,sizeof(int));
     if(!(W1->save(fh)))
        return 0;
     if(!(W2->save(fh)))
        return 0;
     close(fh);

                                          //pone las matrices dentro de ".MAT" en forma legible

     sprintf(fn,"%s.MAT",name);
     ofstream matf(fn,ios::out);
     matf << "\n La primera matriz contiene: ";
     matf << *W1;
     matf << "\n La segunda matriz contiene: ";
     matf << *W2;
     return 1;
   }

int bp::loadweights()
   {
     int fh;
     char fn[32];

     sprintf(fn,"%s.WTS",name);
     fh=open(fn,O_RDONLY|O_BINARY);
     if(!fh)
      return 0;
     if( !(W2->load(fh)))
      return 0;
     close(fh);
     return 1;
   }
```

3.2.3 [redacted]

ALGORITMO DE CONTRAPROPAGACION.

// CPN.HPP
// Incluye la implementación de redes con contrapropagación
// Incluye las clases matrix y vec.

```
#include "net.hpp"

class layer: public matrix, public net
{
 public:
        void chgwts-koh(int c,vec& input);
        void chgwts-gross(int r,double activation,vec& output);
        layer(int n,int p,double randlimit,double r,double d):
                matrix(n,p,randlimit){learnrate=r;decay=d;}
        -layer(){};
};

class cpn: public net
{
 protected:
        int q;
        layer *koh;
        layer *gross;
        double noise;
        double range;
        int *wins.
        double scaling;
        int winners;


 public
        cpn(char *s)'
        cpn(){};
        void encode (const vecpair& AC);
        vec recall(vec& A);
};
```

[redacted]

// CPN.CPP
//Implementación de Hecht-Nielsen´s en el modelo de red con
//Contrapropagación

[redacted]

```
#include "cpn.hpp"
extern int trace;

void layer::chgwts_kob (int r, vec& input)
```

```
{
        vec v=rowslice  (r);
        vec w=input-v;
        v+= (w*learnrate);
        learnrate/=2;
        insertrow(v,r);
}

void layer::chgwts_gross(int c,double activatíon,vec& output)
{
        vec v=colslice(c),
        v*=(i-decay);

        vec tap=(output*activation);
        tmp*=learnrate,
        v+=tmp;
        if(trace)
                cout « "New Grossberg leyer column "
                « c « " " « v « "\n";
        insertcol(v,c);
}


cpn::cpn(char *s):net(s)

      const NOPARMS=5;
      parm  parms{NOPARMS]=
      {
              parm("HIDDEN",(void *)&9.integer);
              parm("RANGE",(voíd *)&range,real);
              parm("NOISE",(voíd *)&noise,real);
              psrm("SCALING",(void *)&sceiinB;real);
              psrm("WINNERS",(void *)&winners,integer);
      };
      parmtbl(NOPARMS,parms,name);

  koh=new layer(p,n,range,learnrate,0);
       wins=new int[p];
  for(int i=0;i<p,i++)
       Wins[i]=0; //el valor es inicializado con valores aleatorios entre 0 y
1
  cycle=o;
  gross=new layer(q,p,o,learnrate,decay);
}

void cpn::encode(const vecpair& AC)
{
       double hidactivation,
       (AC.a)->garble(noise);
       if(winners<2)
         {
              AC.a->normalixeon();
              if(trace)
                    cout « "vector de entrada normalizado " « * (AC.a);
              int  g;
              g=koh->closestrow(*(Ac.a),wins,scaling);
              wins[(int)g]++;
              if(trace){
                    cout « "nodo ganador de Kohonen : " « g
                         « " (" « wins[g] « "-th win)\n"
                    cout « " (output vector) " « *(AC.b) << "\n";
```

```
                    }
            koh->chgwts_koh(g,*(AC.a));
            hidactivation=(koh->rowslice(g)*(*(Ac.a)));
            gross->chgwts_gross(g,hidactivation,*(AC.b));
        }
        else{
            vec gs=koh->closestrows(*(AC.a),winners);
            for(int i=0;i<winners;i++){
                    koh->chgwts_koh((int)(gs[i]),*(Ac.a));
                    hidactivation=
                      (koh->rowslice((int)(gs[i]))*(*(AC.a)));
                    gross->chgwts_gross((int)(gs[i]),
                      hida<tivation,*  AC b));
                }
        }
}

vec cpn::recall(vec& A)
{
        A.nornalizeon();
#if MAXACTIY
        vec b=(*koh)*A;
        int g=b.maxindex();                                    /
#else
        int g=koh->closestrow(A,wins,scaling);
#endif
        if(trace)
            cout « "Chose column " « g « "\n";
        return (gross->colslice (g));
}
```

CAPITULO 4

4.1 USO DEL MODO GRAFICO

4.2 COMO SELECCIONAR LOS ELEMENTOS DE UN PROGRAMA QUE DEBEN SER REPRESENTADOS COMO OBJETOS

4.3 CONSTRUCCION DE AMBIENTES GRAFICOS CON PROGRAMACION ORIENTADA A OBJETOS.

4.3.1 CONTROL DE OBJETOS GRAFICOS.

4.3.1.1 BOTON Y RADIOBOTON.

4.3.1.2 BARRA DE DESPLAZAMIENTO.

4.3.1.3 ICONO.

4.3.2 MANEJO DE VENTANAS

4.3.2.1 TRABAJANDO CON UNA PILA.

4.4 AMBIENTES GRAFICOS CON TECNOLOGIA OOP PARA REDES NEURALES.

CAPITULO 4

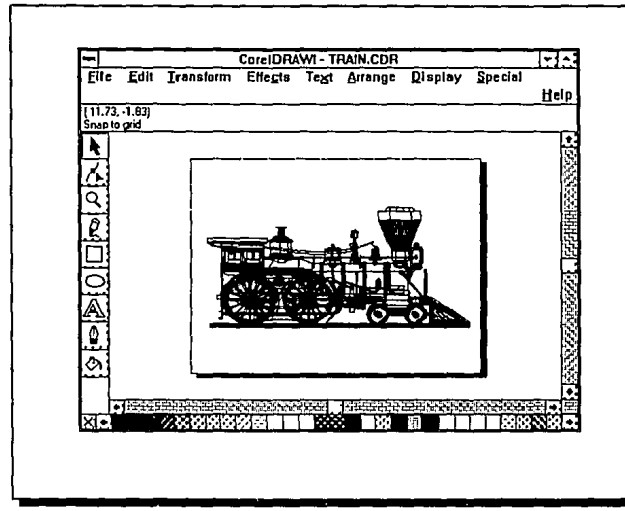
Al paso del tiempo, los gráficos han tomado un papel muy importante en el mundo de las computadoras. Expertos en la industria opinan que esta es la década en que los gráficos serán vistos por todas partes. Por ejemplo, puede ser que los gráficos que se ven en los comerciales sean parte de un programa realizado en una computadora. La programación gráfica ha tomado un papel muy importante en los productos de software hoy en día, por lo que los usuarios de PC's demandan que los programas y herramientas sean más agradables y fáciles de usar.

En la programación de gráficos se realizan diferentes disciplinas, tales como modelados en 3 dimensiones, simulación , animación, análisis, procesamiento de imágenes, anuncios publicitarios, creación de juegos, programas tutoriales y mucho más.

Simulación con la programación de graficos

Además de que los programas de gráficos engrandecen nuestra habilidad para resolver problemas, comprender un amplio rango de datos, y expresar ideas creativas, también cambia de manera muy notoria la imagen y el diseño de los programas. Ayuda de gran manera en areas como ingeniería, arquitectura, industria, matemáticas, medicina, psicología y biología, Además ayuda al entretenimiento y a la publicidad.

Algo que se espera de los ambientes gráficos es que permita el manejo directo de los elementos que aparecen en pantalla y que sea consistente. Esto es, que cuente con acciones y elementos comunes en todas las aplicaciones que se utilizan. Lo anterior es posible si se utiliza la programación orientada a objetos en la construcción de los ambientes gráficos. Sin embargo, una de las principales críticas que se hace a los ambientes gráficos es su costo, no porque un sistema operativo gráfico cueste más que otro, sino a los requerimientos que debe cumplir el equipo que se necesita para usarlos (monitor de alta resolución, excesiva memoria RAM, disco duro de gran capacidad de almacenamiento,etc.).



El mundo de la animación podría relacionarse con dibujos animados como los que salen en la televisión o podríamos relacionarlos con una máquina de video-juegos como las nintendo o atari.

En un principio se utilizó un cinescopio combinado con una serie de figuras hechas a mano para crear el movimiento de una animación, como en las caricaturas de la Warner Brothers o las de Hanna-Barbera. En la actualidad la mayoría de animaciones han sido creadas en su totalidad en una computadora.

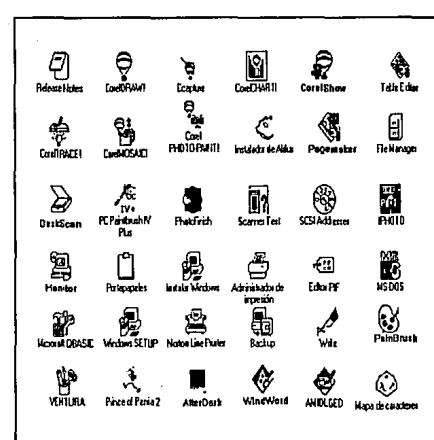
En el pasado, la programación con gráficos era muy difícil de realizar y de gran consumo de tiempo. Afortunadamente, con la aparición de la programación orientado a objetos, la simplificación de estos trabajos ha sido notoria; como en la creación de gráficos de 2 ó 3 dimensiones en computadoras personales. Sin embargo, no se ha aprovechado al máximo la Programación Orientada a Objetos.

La programación orientada a objetos ofrece de forma inmediata y significativa, ventajas para los programas gráficos dentro de los conceptos de polimorfismo, inherencia y encapsulación. Esto es, transforma inmediatamente sus programas en trabajos capases de desplegar, mover, borrar, salvar o cargar imágenes. Y lo mejor de todas estas características, sólo se necesita escribirlo una sola vez para poder realizar todas estas tareas. Además de poder poner gráficos en diferentes colores, tamaños y formas, C++ puede ayudar en el desempeño, realización e incremento de su productividad.

Un programa orientado a objetos no es ya una serie de líneas numeradas que contienen instrucciones y que se ejecutan secuencialmente; tampoco es una serie de "procedimientos" que pueden ser llamados desde un "programa principal". En un lenguaje orientado a objetos, un objeto es un paquete de datos y procedimientos que permanecen juntos. Específicamente todas las constantes y el contenido de todas las variables involucradas forman un objeto. Se pueden crear categorías de objetos y se cuenta con un "enlazamiento" dinámico que permite agregar nuevas categorías sin tener que modificar el código existente.

Las imágenes son un gran recurso para la programación orientada a objetos, siendo los iconos, las gráficas y los diagramas los elementos principales para la creación de ambientes gráficos. La visualización se presenta como una nueva forma de interfaz entre el usuario y la computadora, ofreciendo al primero mejores maneras para expresarse, entender, resolver y documentar los diversos problemas.

Los símbolos pictóricos, gráficas y diagramas son manifestaciones visuales que tienen por objetivo: representar, ilustrar y comunicar objetos e ideas. Los datos como **objetos** se pueden representar y describir en gráficas, mostrándose en estas mismas nuevas ideas obtenidas de las **relaciones funcionales** que se describen en la gráfica anterior. Por un lado, las ideas concebidas por el comunicador son sintetizadas en diagramas y símbolos gráficos (íconos), ofreciendo una atención atractiva al lector que le facilita el recordar conceptos y relaciones, y por otra parte, el concepto de gráfica se aplica a lo que se presenta por medio de figuras o signos.



La programación orientada a objetos se utiliza en una gran variedad de aplicaciones, desde gráficos para un negocio, hasta la animación de demostraciones. Los ambientes gráficos se desarrollan con objetos gráficos que pueden utilizarse en otras aplicaciones futuras, por ejemplo, el objeto ratón.

Conforme las microcomputadoras evolucionaron con rapidez, un pequeño grupo de investigadores desarrolló nuevos métodos para controlar computadoras basándose en la idea de una interfaz gráfica con el usuario. La idea básica de esta concepción fue crear un ambiente visual de fácil comprensión y uso para el usuario de la computadora. En éste, todas las operaciones se inician por la selección de símbolos, llamados íconos, que representan una operación específica.

4.1 USO DEL MODO GRÁFICO

Desde que el Massachussets Institute of Technology consiguió en 1952 que una computadora dibujará unas sencillas figuras en un cinescopio de TV, la utilización de las computadoras, tanto en el campo del dibujo y diseño asistido por computadora (CAD), como en el de la fabricación asistida por computadora (CAM), ha experimentado un espectacular desarrollo, que apunta hacia límites inimaginables.

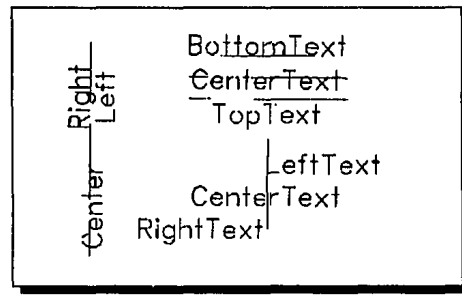
Los gráficos se diseñaron para proporcionar al mundo de las computadoras una alta calidad, la mayoría de estos gráficos siguen una cierta filosofía de apariencia y de diseño cuyo resultado es un ambiente amable para el usuario.

El diseño de gráficos requiere de microprocesadores más rápidos, unidades de discos rápidas, demasiada memoria y monitores de alta resolución (1024 x 780 pixeles) para imágenes de alta calidad.

A mediados de la década de los ochenta, la serie de computadores Apple Macintosh rompieron la tradición al usar los gráficos. Hoy en día están disponibles en el mercado varios productos con ambientes gráficos para las computadoras.

En modo gráfico, las coordenadas comunes (de 80 x 25) son reemplazadas por las coordenadas de tipo pixel, y estas varían dependiendo del tipo de interfaz a usar, desde los 320 x 200 pixeles (en un monitor CGA), hasta más de 1024 x 768 pixeles (como en los monitores VGA, SVGA y multisincrónicos).

Debido a la gran variedad de resoluciones de pantalla, la mayor parte de los programas checan el tipo de hardware, para un apropiado controlador de video. Ellos determinan el tipo de video y el tamaño de pantalla que se va a utilizar, ajustando las operaciones del mismo.



Las líneas y las curvas se utilizan para muchas aplicaciones de dibujo, algunas imágenes pueden solamente ser manipuladas individualmente por pixeles. Imágenes enteras pueden ser salvadas, reescritas y borradas o ser combinadas con imágenes existentes en la pantalla.

El texto en modo gráfico opera completamente diferente del texto convencional. Por ejemplo, en el modo convencional el texto se posiciona por filas y columnas, en el modo gráfico se determina por la posición del pixel en la pantalla.

Debido a que el tamaño del caracter varía, y que el texto puede ser desplegado tanto vertical como horizontalmente, la línea del caracter y la posición pueden llegar a confundirse; por lo que se utilizan varias funciones para hacer más cómodo el despliegue del texto, el tamaño y el ancho del tipo de letra, llamada **font** en el modo gráfico.

4.2 COMO SELECCIONAR LOS ELEMENTOS DE UN PROGRAMA QUE DEBEN SER REPRESENTADOS COMO OBJETOS

La selección de objetos es un problema importante que se tiene cuando se programa con C++ (OOP). Aunque el proceso de determinar los componentes de un programa que deben ser representados como objetos parezca mágico a primera vista (como una caja negra), se tienen algunas reglas generales que se pueden seguir como una ayuda en el diseño de aplicaciones orientadas a objetos. La idea principal en que se basan estas reglas consiste en diseñar el código de tal manera que se pueda tomar ventaja de las tres propiedades clave de la OOP : **Encapsulación, Herencia** y **Polimorfismo**. A continuación se enlistan las reglas que se pueden tomar como una guía a seguir :

1.- La mayoría de las estructuras de datos tradicionales, tanto las dinámicas (listas ligadas y árboles) como las estáticas (pilas y colas) y demás por el estilo, son excelentes elementos que pueden representarse como objetos. La razón es que la estructura **objeto** permite la encapsulación de las operaciones que se necesitan para procesar la estructura de datos junto con los datos que almacena la estructura misma. Aquí la ventaja es que la estructura de datos (objeto) es usada y manipulada más fácilmente debido a que todas sus características se almacenan en una sola entidad.

2.- Los componentes de la interfaz con el usuario, como lo son las ventanas, los menús, los botones, las cajas de diálogo, y demás por el estilo, también son buenos candidatos para representarse como objetos. Esta clase de componentes se benefician principalmente de la propiedad de **Herencia**. Por ejemplo, se puede definir el objeto **ventana_texto** para trabajar en el modo texto y después utilizar éste mismo para derivar un objeto **ventana_graf** que trabaje en modo gráfico. El objeto base **ventana** también puede ser usado para derivar el objeto **menú** o el objeto **botón**.

3.- Los componentes que se utilizan en un programa para modelar objetos o eventos del mundo real, como las mediciones, dimensiones, indicadores, gráficas, gente, automóviles, bosques, conferencias, etc, pueden ser representados como objetos.

4.- Los componentes **Hardware**, como el ratón, el teclado, y la pantalla también pueden ser usados como objetos. Cuando se trabaja con este tipo de objetos lo deseable es definir los objetos generales **dispositivo de entrada** y **dispositivo de salida** y a partir de estos definir objetos de entrada/salida más específicos para soportar el ratón, el teclado y la pantalla.

5.- Existen situaciones en donde el código del **programa principal** puede representarse como un objeto.

6.- También se tienen situaciones en donde los algoritmos se pueden representar como objetos. Por ejemplo, se puede crear el objeto **Ordenación** para encapsular diferentes algorítmos de ordenación (como el Quick Sort o el método de la burbuja) ó el objeto **Imagen** para encapsular los diferentes métodos de compresión y almacenamiento de imágenes (como PCX, GIFF, TIFF,BMP,etc). Se puede representar un compilador general como un objeto y derivar diferentes compiladores a partir del objeto **compilador general** para cada uno de los diferentes lenguajes de programación.

4.3 CONSTRUCCIÓN DE AMBIENTES GRÁFICOS CON PROGRAMACIÓN ORIENTADA A OBJETOS.

Anteriormente los usuarios de computadoras encontraban incómodo crear una atractiva presentación gráfica. Afortunadamente la situación ha cambiado. Actualmente las computadoras tienen una mejor y muy alta resolución gráfica. Una natural consecuencia de la programación con gráficos es la de proporcionar una aplicación que contenga controladores gráficos.

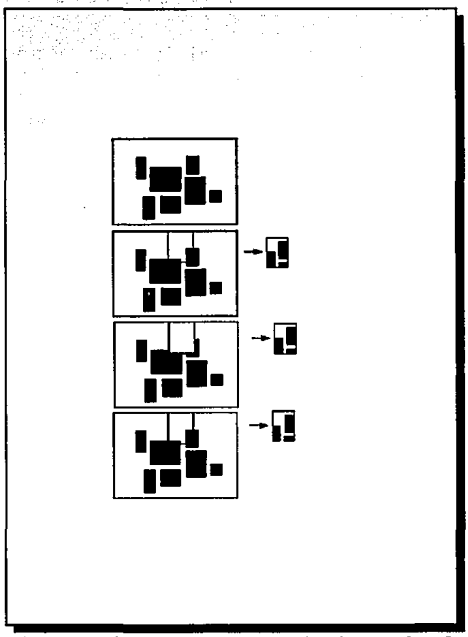
4.3.1 Control de Objetos Gráficos.

La programación orientada a objetos ofrece una variedad de comodidades para el programador. Un área donde la comodidad es más impresionante es la creación del control de los objetos gráficos.

Para aplicaciones gráficas, un control de objetos tiene que cumplir con los siguientes criterios :

1) Creación y Mantenimiento de la Pantalla.
2) Modificar Tamaño y Posición de la Pantalla.
3) Cambiar el Aspecto y Presentación de la Pantalla.
4) Definir la Función Asociada a cada Objeto Gráfico.
5) Eliminar la Pantalla de la Memoria.
6) Control total del Ambiente Gráfico con el Objeto Ratón.

Para la creación del control de objetos es necesario utilizar los objetos gráficos: Botón, RadioBotón, Barra de Desplazamiento e Icono. El código en C++ de este tipo de objetos gráficos se presenta en el apéndice B.



Algunas de las características de la programación orientado a objetos: borrado, desplegado, movimiento, cargado y salvado de imagenes

4.3.1.1 Botón Y RadioBotón.

El Botón es básicamente una representación gráfica en pantalla de un Botón físico. Existen tres tipos de Botón de acuerdo a su forma, que son :

1) Circular.
2) Cuadrado.
3) Tridimensional.

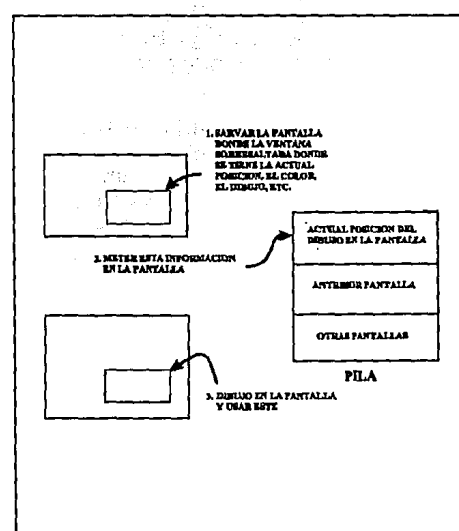
Hay varias funciones que se usan para construir un botón, por ejemplo, una que describa que tipo se tiene, dos funciones para dar el ancho y el tamaño; otra donde se define el nombre; se cuenta con otra función que dice si el Botón ha sido seleccionado o no (cambia de color cuando es seleccionado), y por último otra función para aceptar el doble click del ratón como selección del Botón.

El RadioBotón es un Botón circular contenido en un grupo de botones del mismo tipo, en donde solamente uno a la vez puede ser seleccionado.

El RadioBotón es una variante del Botón que utiliza las mismas variables de éste y adicionalmente tiene otras dos; una que le da el color y otra que le da el radio al Botón.

4.3.1.2 Barra de Desplazamiento.

La mayoría de los programas de aplicación gráfica requiere de una página gráfica extendida, pero la pantalla física esta limitada en tamaño, por lo que se requiere de una herramienta que permita un desplazamiento de la pantalla para poder ver toda la página gráfica. A esta herramienta se le denomina Barra de Desplazamiento.

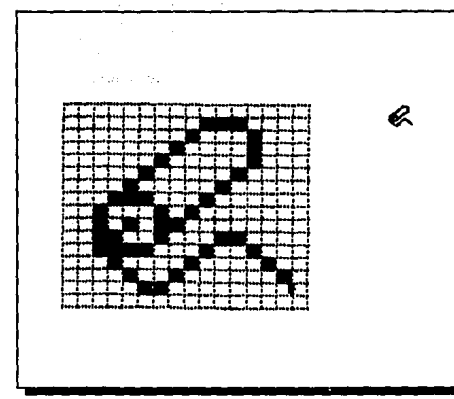


La barra de desplazamiento contiene dos elementos principales: las flechas y el cuadro de desplazamiento. Las flechas indican la dirección del desplazamiento, actúan como botones que pueden oprimirse para mover la información que se tiene en pantalla; hacia arriba o hacia abajo, cuando se tiene una barra de desplazamiento vertical, y hacia la derecha o hacia la izquierda, cuando se trata de una barra de desplazamiento horizontal. El cuadro indica la localización aproximada de la pantalla dentro de la página gráfica completa.

La barra de desplazamiento es una variante del objeto Botón, es decir, se declara como un descendiente de este último, por tal hereda las funciones del padre, pero además se declaran para éste otras funciones. Una función determina la orientación, ya sea horizontal o vertical; una que determina el color; otra que maneja la posición del cuadro y una última que incrementa el valor de la posición del cuadro.

4.3.1.3 Icono.

Con el incremento de las aplicaciones gráficas, muchos programas, y por consecuencia los programadores, han abandonado los menús de texto para basarse en las aplicaciones con íconos. Sin embargo, se ha abusado tanto en su uso, que muchos los consideran más un estorbo que una ayuda. Para el agrado de unos o el disgusto de otros, los íconos siguen siendo elementos principales en la programación de gráficos.

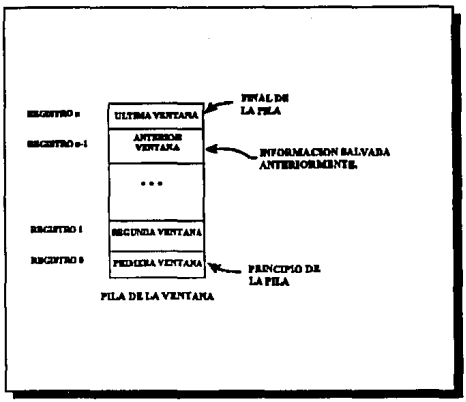


Un ícono es una idea gráfica (ideograma) de una función o un proceso. Los íconos sirven como un símbolo en pantalla y como un Botón al asumir el control de aplicaciones y procedimientos.

4.3.2 Manejo de Ventanas

Las ventanas gráficas son muy atractivas, debido a que puede ser ocupada para mensajes de títulos, crear menús pull-down (desplegables), y enlazar otros programas de entrada y salida.

Para el manejo de ventanas, se necesita un conjunto de funciones para la creación, desplegado y removimiento de la ventana y una pila donde se almacenan las ventanas, para que realice el desplegado y limpiado de las mismas.



El proceso de creación de ventanas se muestra en la figura 4.1. Como se observa la idea básica es la de salvar la región de la pantalla en donde estará sobrepuesta la ventana resaltada (Pop-Up), así como cualquier parámetro de la pantalla que pudieran ser cambiados y que más tarde esta información pueda ser usada para restaurar la pantalla, al estado en que se encontraba antes de que la ventana fuera creada. Por tanto, nosotros no podremos arbitrariamente brincar a una ventana que este cubierta por otras varias ventanas que se encuentran hacia el frente de la pantalla. Actualmente esta característica se puede implementar (p.e. en MS-Windows).

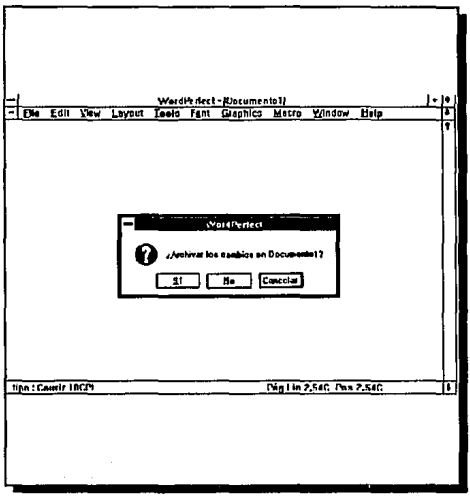
Al remover una ventana resaltada desde la pantalla, necesitaremos 2 pasos:

1.- Sobregrabar la ventana resaltada con una imagen almacenada de la pantalla en una pila.
2.- Restablecer los parámetros de la pantalla que son salvados en la pila.

Los parámetros de la pantalla que deseamos incluir como atributos del dibujo, posición y el color, se necesitan salvar para que después sean ocupados.

4.3.2.1 Trabajando con una pila.

El modo de empleo es muy simple, nuestra pila se implementa como un simple arreglo bidimensional, como se muestra en la figura 4.2, donde cada elemento en el arreglo almacena la información discutida anteriormente por la ventana desplegable(Pop-Up). Note que la pila esencialmente contiene el índice del arreglo y normalmente el apuntador a la siguiente localización en la pila. Este se incrementa de uno en uno al mismo tiempo que la ventana se crea en la pila. De este modo, si hay 4 ventanas en la pila, el apuntador de la pila apuntará hacia la cuarta posición.



Cada elemento en el arreglo es definido como un apuntador hacia la estructura. La estructura es usada para salvar la posición, el status y los parámetros de la región que se encuentra por debajo de la ventana actual.

La estructura es usada para salvar cualquier cosa que pueda ser cambiada cuando se despliega la ventana (Pop-Up). Estos atributos son usados para restaurar el status de la pantalla a su estado original cuando la ventana sea removida de la pantalla.

4.4 AMBIENTES GRÁFICOS CON TECNOLOGÍA OOP PARA REDES NEURALES.

La programación orientada a objetos, como se describió en el capítulo 2, es un nuevo paradigma en la ingeniería de software. Con esta nueva técnica se intenta resolver los problemas en los cuales la programación estructurada ya no es funcional.

La OOP reúne lo mejor de los demás paradigmas de programación resultando en una combinación con mejores características y un mayor potencial, cambiando hacia un nuevo punto de vista en las técnicas de programación.
2
La OOP es casi ilimitada para las aplicaciones gráficas, siendo los gráficos el principal campo para la orientación a objetos. Los objetos proporcionan capacidades gráficas poderosas que usando la programación convencional (programación estructurada) resultaría ser muy complejo para la implementación en la mayoría de los programas.

El uso de los ambientes gráficos ha demostrado ser una herramienta poderosa en diversas aplicaciones, aumentando aún más su capacidad al implementarlos en base a la programación orientada a objetos. Por tanto, se puede observar que el desarrollo de redes neuronales con ambientes gráficos presenta grandes ventajas, siendo algunos ejemplos :

a) facilidad de crear una red neuronal con sólo conectar íconos y asociarles un peso por medio del ratón.

b) Tener una interfaz amigable para un usuario que no sea experto en el área de aplicación y darle una poderosa herramienta para que el mismo construya aplicaciones en base a una red neuronal ya implementada siendo ésta totalmente transparente para el usuario.

c) Presentación de resultados en forma visual por parte de la red neuronal hacia el usuario de manera que le permita comprender fácilmente el estado actual de su aplicación.

Dos aplicaciones que presentan ventajas en la utilización de ambientes gráficos en el desarrollo de redes neuronales son :

1) Un ambiente gráfico puede ser una potencial herramienta para diseñar una
red neuronal :

* A través de íconos se puede hacer la interconexión de las neuronas y
definirles un peso asociado.

* El mismo ambiente gráfico puede ser una red neuronal; así se tendría
una red generadora de redes.

2) Cada ícono es una neurona altamente conectada (Sinápsis alta) con una
aplicación específica.
La conexión se da en el ideograma que define al ícono o por el texto
gráfico que acompaña a cada ícono.
El objeto "ratón" puede seleccionar un ícono y la red neuronal deberá
ejecutar el programa o grupo de programas que se asocian con ese ícono.

A partir del desarrollo de aplicaciones como las que se describen en el punto **1** se pueden desarrollar las aplicaciones indicadas en el punto **2**. Ejemplos de esto se pueden ver en algunos paquetes de software para construcción de redes neurales, por ejemplo, el **NeuroWindows**.

CAPITULO 5

5.1 COMO APLICAR LA NEUROCOMPUTACION

5.1.1 SOLUCION DE PROBLEMAS CON NEUROCOMPUTACION.

5.1.2 DESARROLLO DE LAS ESPECIFICACIONES FUNCIONALES.

5.2 REDES NEURALES E INGENIERIA DE CONTROL.

5.2.1 IDENTIFICACION DEL PROCESO.

5.2.2 ARQUITECTURAS BASICAS DE CONTROL CON APRENDIZAJE NO-DINAMICO.

5.2.3 APRENDIZAJE ESTRUCTURAL Y PARAMETRICO

5.2.4 OBTENCION DE INFORMACION PARA EL ENTRENAMIENTO

5.2.4.1 COPIANDO UN CONTROLADOR EXISTENTE

5.2.4.2 PREDICCION ADAPTIVA

5.2.4.3 IDENTIFICACION DEL SISTEMA

5.2.4.4 IDENTIFICACION DEL SISTEMA INVERSO

5.2.4.5 DIFERENCIACION DEL MODELO.

5.2.5 APRENDIZAJE REFORZADO

5.2.5.1 EL PAPEL DEL APRENDIZAJE REFORZADO EN CONTROL

5.2.6 EXCITACION PERSISTENTE

5.2.7 CONTROL ADAPTIVO

5.2.7.1 CONTROL DIRECTO

5.2.7.2 CONTROL INDIRECTO

5.2.7.3 INFORMACION PRELIMINAR

5.2.7.4 EL MODELO DE REFERENCIA

5.2.8 REDES NEURALES COMO CONTROLADORES ADAPTIVOS

5.2.9 REDES NEURALES ADAPTIVAS EN CONTROL DE PROCESOS QUÍMICOS

5.2.9.1 MOTIVACIONES PARA EL CONTROL DE PROCESOS QUÍMICOS

5.2.9.2 COMPARACIÓN ENTRE EL CONTROL DE PROCESOS QUÍMICOS Y EL CONTROL DE ROBOTS.

5.2.9.3 CRITERIOS DE DESEMPEÑO EN EL CONTROL DE UN BIOREACTOR.

5.2.9.3.1 CONTROL DEL BIOREACTOR UTILIZANDO UNA RED NEURAL ADAPTIVA.

5.2.9.3.2 VERSIONES MAS COMPLEJAS DEL CONTROL DEL BIOREACTOR.

5.2.10 SISTEMAS DE CONTROL DISTRIBUIDO BASADOS EN PLC'S

5.2.10.1 EL PLC.

5.2.10.2 LOS SISTEMAS DE CONTROL DISTRIBUÍDO (SCD).

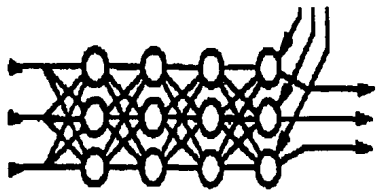
5.2.10.3 COMBINANDO PLC'S Y SCD'S.

5.2.10.4 IMPLEMENTACION DE UN SCD BASADO EN PLC'S.

5.2.10.5 EL FUTURO: MAS RAPIDO, MAS ECONOMICO, MAS PEQUEÑO.

5.2.11 OTRAS APLICACIONES

CAPITULO 5



A diferencia de las matemáticas y de la ciencia, las cuales, respectivamente, persiguen una belleza pura y un entendimiento de la naturaleza, la tecnología persigue el desarrollo de la utilidad de las cosas. Así, el verdadero valor o mérito de la tecnología en neurocomputación deberá ser medido en cuanto a su aplicación a problemas del mundo real.

Como en la mayoría de las nuevas tecnologías, la neurocomputación es sólo una nueva herramienta en una ya bien equipada caja de herramientas. Para que la neurocomputación realice todo su potencial, será necesario mezclar las técnicas en neurocomputación con otras técnicas existentes. Por tanto, se deberá constantemente responder a las interrogantes de que como se podrá efectivamente combinar la neurocomputación con otros ingredientes tecnológicos.

5.1 COMO APLICAR LA NEUROCOMPUTACION

Dos resultados importantes que se tienen al aplicar la neurocomputación a la ingeniería son, primero, el problema de las metodologías de solución y el desarrollo de las especificaciones

funcionales. Estos resultados principalmente se relacionan con el proceso inicial de definir la aplicación de la neurocomputación. Si este proceso inicial se realiza con éxito, entonces las metodologías y técnicas en redes neurales pueden ser empleadas para analizar la aplicación propuesta, desarrollar un plan de trabajo para el proyecto, y (si el plan de trabajo demuestra obtener lo esperado en el proyecto y dirige la distribución de los recursos necesarios) planear y administrar el desarrollo del proceso resultante.

5.1.1 SOLUCIÓN DE PROBLEMAS CON NEUROCOMPUTACION.

El principal problema de la metodología de solución en neurocomputación es la de **"trabajar en reversa"**. Se comenzará construyendo una lista de redes neurales que estén suficiente entendidas para ser utilizadas en la solución de problemas del mundo real. Esta lista describe las capacidades funcionales de cada red y especifica el tipo y cantidad de datos que deberán estar disponibles para el entrenamiento y prueba de la red. También se deberá construir una lista exhaustiva de las funciones operacionales implicadas en el dominio de la aplicación. Por ejemplo, si el dominio de la aplicación es el ensamblaje de computadoras, esta lista deberá contener los siguientes puntos:

* Venta y Mercadotecnia.
* Orden de compra y recepción de componentes de un equipo de cómputo (teclados, periféricos, monitores, etc.).
* Almacenaje y transporte interno de los distintos materiales de cómputo.
* Prueba de cada componente por separado, (Primera etapa del control de calidad).
* Control de ensamblado.
* Prueba del equipo completo (Segunda etapa del control de calidad).
* Empacado del producto final.
* Distribución del equipo.
* Planeación y Administración.

En este ejemplo, como en muchos otros, se tienen como candidatos probables para aplicaciones de redes neurales a cada área operacional de la empresa de ensamblaje de equipo de cómputo. Se determinarán estas aplicaciones comparando los problemas e identificación de oportunidades de la lista de funciones operacionales con las capacidades ofrecidas en la lista de capacidades funcionales de las redes neurales.

En resumen, la principal metodología para aplicar neurocomputación es la de trabajar en reversa. Se construye una lista de redes neurales, la cual se describe para cada red sus capacidades en la resolución de problemas y sus requerimientos de entrenamiento. También se desarrolla una lista de funciones operacionales (ó categorías de producción), y serán consideradas las aplicaciones en cada una de estas áreas (con una fuerte intervención del usuario final a través de este proceso). Las posibles aplicaciones que surgen de este proceso se evaluarán durante el desarrollo de las especificaciones funcionales.

5.1.2 Desarrollo de las especificaciones funcionales.

Una vez que se ha determinado la posible aplicación de neurocomputación, el siguiente paso es definirla con exactitud. Este paso es esencial, a partir de que tratar de resolver un problema que no ha sido definido con exactitud es como tratar de leer la información de una tarjeta perforada con una computadora notebook 486.

El medio por el cual un problema es definido, es a través de un documento corto, pero formal, es llamado **"Especificación Funcional"**. La especificación funcional es un mecanismo que permite trabajar en un proyecto a aquellas personas que no son expertas en el área en donde se realiza la aplicación. Se diseñará y se desarrollará un sistema de trabajo que satisfaga los requisitos indicados en la especificación funcional. Por lo tanto, la especificación funcional son tal vez el documento más importante de cualquier proyecto de aplicación.

Después de obtener la especificación funcional, iterativamente se mejorará y se discutirá entre las personas que desarrollan el proyecto y el grupo de usuarios finales. También serán importantes los comentarios y críticas de gente altamente especializada en redes neurales, ya que ellos ayudaran a validar la concepción de la aplicación desde una perspectiva técnica, así como a evaluar la posibilidad y factibilidad de poder llevar a cabo el proyecto de aplicación. Profesionistas con muchos años de experiencia en un área particular frecuentemente tienen una misteriosa habilidad de visualizar el proyecto propuesto y exactamente estimar el esfuerzo, los materiales y el tiempo que serán requeridos para realizar el proyecto.

Una especificación funcional debe escribirse de acuerdo a los siguientes puntos :

1.- Descripción : una breve descripción de la capacidad o el

producto que será desarrollado.

2.- Requerimientos de interfaz con el sistema : Requerimientos para intercomunicarse con otros sistemas.

3.- Requerimientos de interfaz con el usuario : Requerimientos o restricciones con respecto a la interfaz entre el sistema desarrollado y el usuario (Ambientes Gráficos, Ver capítulo 4), incluyendo cualquier requerimiento de seguridad.

4.- Requerimientos de funcionamiento : Requerimientos sobre todo el desempeño del sistema, estos requerimientos pueden incluir,

a) Precisión.
b) Velocidad.
c) Seguridad.
d) Magnitud.

Estos requerimientos se obtienen fácilmente por medio de técnicas OOP (ver capitulo 2).

5.- Requerimientos económicos: Requerimientos con respecto a los costos recurrentes, presentación, semejanza con otros productos o sistemas, y demás.

6.- Requerimientos de desarrollo del proyecto : Restricciones que pueden incluir limitados costos no recurrentes, restricciones no usuales del medio ambiente, y otras restricciones en suministro de materiales, seguridad , propietarias y programadas.

Se deberá construir un bosquejo inicial de la especificación funcional, la cual será revisado y mejorado por los usuarios finales y por el líder del proyecto.

Una vez que se ha completado la especificación funcional, se deberá tomar la decisión de seguir o no adelante. Esta decisión debe hacerse en dos pasos. Primero, reunir a toda la gente que estará involucrada en el proyecto y tomar conjuntamente la decisión de seguir o no. Si la decisión es de seguir , entonces el siguiente paso será trazar un plan de trabajo para el proyecto. Un plan de trabajo es una herramienta para evaluar cualquier posible proyecto. En este sentido, la especificación funcional sirve como un filtro inicial para la concepción de posibles aplicaciones. Típicamente, solo una pequeña fracción de especificaciones funcionales que fueron escritas se encuentran en el proyecto final. El proceso de describir la especificación funcional usualmente identifica problemas que hacen la concepción inicial no interesante o imposible.

En resumen, las aplicaciones de neurocomputación son como otras aplicaciones técnicas. Sin embargo, debido a las nuevas capacidades que ofrece la neurocomputación, ésta se vuelve a veces el elemento que dirige el diseño del proyecto.

5.2 REDES NEURALES E INGENIERÍA DE CONTROL.

El uso de las redes neurales en aplicaciones a la Ingeniería de Control, en donde se incluye el control de procesos, robótica, manufactura industrial y aplicaciones aeroespaciales, entre otras, ha experimentado un reciente crecimiento rápido. El objetivo básico de control es la de proporcionar la señal de entrada apropiada a un proceso físico dado para producir su respuesta deseada. En la teoría de sistemas de control el proceso físico a ser controlado generalmente es referido como **la planta.** Las señales de entrada a la planta, llamadas señales actuadoras, son típicamente necesarias para el control de motores, conmutadores, u otros dispositivos actuadores. Si la señal actuadora en la entrada de la planta (mostrada en la figura 5.1) es generada por el controlador basado en una red neuronal, al caso se le puede dar el termino de **control neural**, o brevemente **neurocontrol.**

Las soluciones a los problemas en control dependen directamente de la precisión con que se conocen las características del proceso. En un problema de control de un sólo canal cuando la entrada de la planta es una variable escalar, por ejemplo, el proceso, si es conocido, puede ser descrito por una relación entre la señal actuadora de la planta y la respuesta de la planta. en general esta relación se expresa como **P**, donde **P** es la función de transferencia del proceso de la forma **P(s)**, y **s** es una variable compleja. Se consideran las características de la planta invariantes en el tiempo. En tales casos, **P(s)**

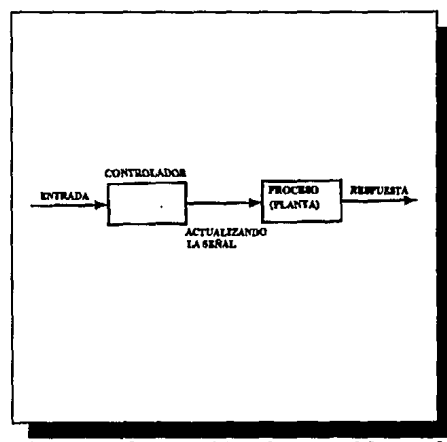


Fig.5.1 Ilustración del problema de control.

incluye las propiedades dinámicas que relacionan las señales de entrada y salida de la planta. Para plantas que puedan ser descritas en el dominio del tiempo sin una variable dependiente del tiempo, **P(s)** es independiente de **s** y se reduce a una simple función relacional entre la señal actuadora y la señal de respuesta. Dichas plantas son llamadas No-dinámicas, de poca memoria, o simplemente estáticas. Para plantas estáticas la función de transferencia **P(s)** se convierte en **P** y es denominada la **característica de transferencia.**

Considérese un ejemplo simple de plantas dinámicas y estáticas. Supongase que la corriente a través de un elemento eléctrico de un puerto es la salida de la planta, la cual necesita ser controlada. El voltaje a través de este elemento de un puerto es considerado la entrada a la planta, o señal actuadora. La planta se muestra en la figura 5.2a. La figura también ilustra la equivalencia entre una especificación eléctrica del elemento y la comúnmente usada representación general de diagrama a bloques de la planta.

Si se considera que la planta es simplemente un resistor (como se muestra en la figura 5.2b), la característica de transferencia de la planta se reduce a la siguiente constante:

$$P = \frac{i(t)}{v(t)} = G$$

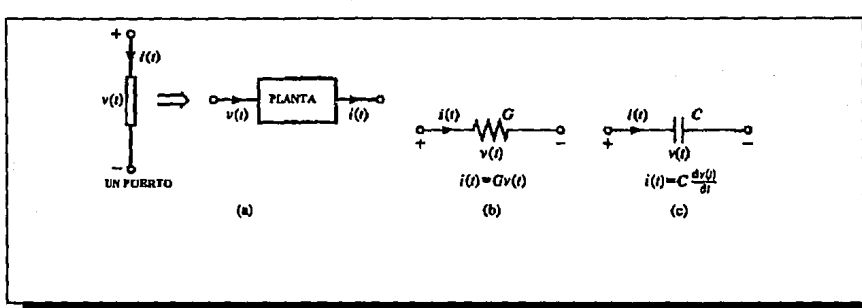


Fig. 5.2 Ilustración de la función de transferencia de una planta estática y una dinámica: (a) elemento eléctrico de un puerto como planta, (b) resistor, y (c) capacitor.

Así entonces, una planta estática puede ser descrita por la característica de transferencia **P**. En este caso, que es un proceso lineal y estático (la resistencia), la característica de transferencia es igual a la conductancia **G** del resistor.

Una planta dinámica, como lo es el capacitor mostrado en la figura 5.2c no puede ser caracterizada por constantes, no siempre por funciones complejas invariantes en el tiempo, debido a sus respuestas, siendo soluciones de las ecuaciones diferencial o integrodiferencial, funciones en el tiempo. Así entonces, su función de transferencia será:

$$P(s) = \frac{I(s)}{V(s)} = sC$$

El significado de las características de transferencia en la teoría de control de sistemas está basado en la factibilidad de que la respuesta de un sistema estático a una excitación arbitraria, puede ser calculada como el producto de su característica de transferencia y de la excitación. Similarmente, la transformada de la respuesta de un sistema dinámico puede expresarse como el producto de la función de transferencia y la transformada de la excitación.

5.2.1 IDENTIFICACIÓN DEL PROCESO.

En general la identificación de la planta debe resultar en términos de los coeficientes de la ecuación diferencial de la planta o coeficientes de su función de transferencia. El principio de la identificación es tal vez de una gran importancia en el campo de los sistemas de control adaptivo. A partir de que la planta en un sistema de control adaptivo varía en su operación en el tiempo, el control adaptivo debe ser ajustado para estimar las variaciones de la planta.

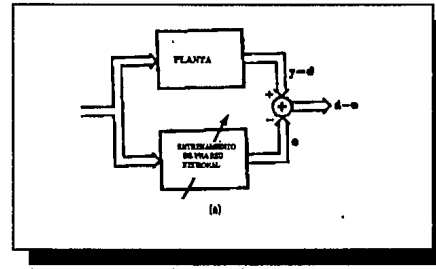


Fig. 5.3 Configuración de la red neural para la identificación de la planta: (a) Identificación de planta adelantada.

La configuración básica para la identificación de planta adelantada se muestra en la figura 5.3a. Una red neural no-lineal recibe la misma señal **x** de la planta y la salida de la planta es la respuesta deseada **d** durante el entrenamiento. El propósito de la identificación es encontrar la red neural o la respuesta **o** que iguala la respuesta **y** de la planta para un conjunto dado de entradas **x**. Durante la identificación la norma del vector de error, $\|\mathbf{d}-\mathbf{o}\|$, es minimizado a través de un número de ajuste de pesos. La minimización se realiza usando técnicas de aprendizaje como las vistas en el capitulo 1.

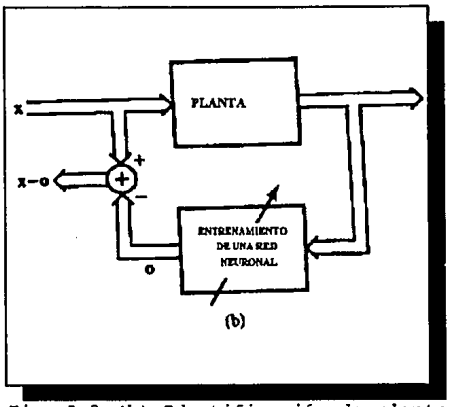


Fig. 5.3 (b) Identificación de planta inversa.

La figura 5.3a muestra el caso para el cual la red intenta modelar el mapeo de la entrada a salida de la planta, midiendo tanto la entrada como la salida al mismo tiempo. esto es comprensible a partir de que el entrenamiento de la red neural es de alimentación adelantada (feedforward) e instantáneo; así entonces se tiene que **o(t)=x(t)**. Sin embargo tipos más complejos de modelos para identificación de la pla nta pueden ser también empleados. Para calcular la entrada de la planta dinámica, la red neural debe consistir de varias entradas pasadas de la planta. Estas se pueden producir insertando elementos de retardo a la entrada de la planta.

La identificación de la planta inversa puede ofrecer otra alternativa viable para el diseño de sistemas de control. En contraste a la identificación de las características de la planta adelantada, ahora la salida **y** de la planta es usada como entrada a la red neural (figura 5.3b). El vector de error para el entrenamiento de la red se calcula como **x-o**, donde **x** es la entrada. Las normas del vector de error a ser minimizado por medio del aprendizaje es entonces $\|\mathbf{x}-\mathbf{o}\|$. La red neural en-

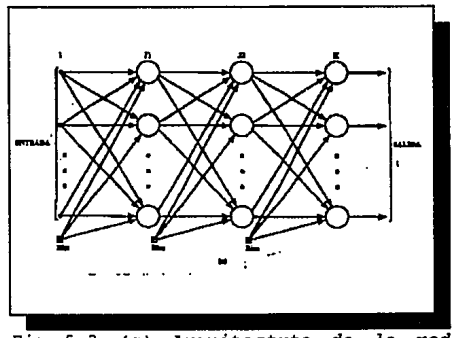


Fig 5.3 (c) Arquitectuta de la red neural entrenada.

trenada de esta forma implementará el mapeo de la planta inversa. Una vez que la red ha sido totalmente entrenada para imitar a la planta inversa, ésta puede ser usada directamente para el control inverso adelantado, como se ilustra en la figura 5.4a. El entrenamiento de la red neural propiamente actúa como un controlador en esta configuración, y la señal actuadora apropiada para la planta está directamente disponible a la salida de la red.

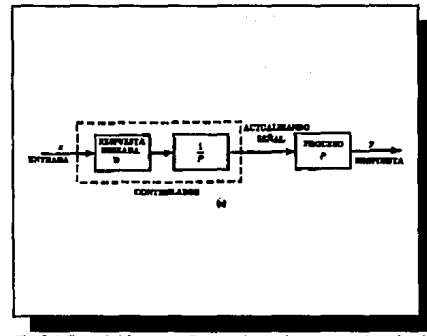


Fig 5.4 Control del proceso P para obtener la respuesta deseada: (a) Control con alimentación inversa por adelanto .

En ambos casos de identificación, la red neural es entrenada para copiar las características de la planta tanto de la adelantada como de la inversa. Una red entrenada como la que se muestra en la figura 5.3c puede modelar el comportamiento de la planta actual en operación en modo adelantado o inverso. Cada uno de los modos de identificación tiene sus ventajas y desventajas. La identificación de la planta adelantada siempre es posible, sin embargo, ésta no permite inmediatamente la construcción del controlador de la planta. En cambio la identificación de la planta inversa facilita el control simple de la planta, no obstante, la identificación por si misma no siempre es posible.

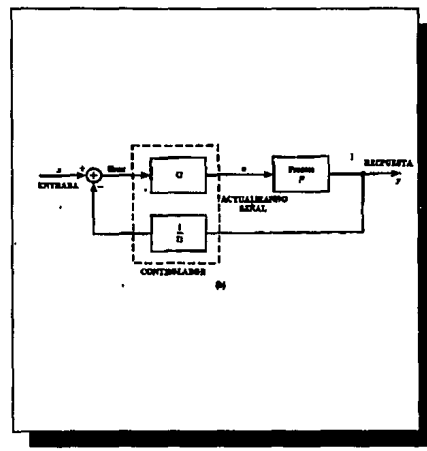


Fig.5.4 (b) Control con retropropagación.

5.2.2 ARQUITECTURAS BÁSICAS DE CONTROL CON APRENDIZAJE NO-DINÁMICO.

Existe un método específico de control en el cual las redes neurales multinivel se utilizan como controladores de plantas no-dinámicas. En el entrenamiento del controlador basado en redes neurales se demuestra la factibilidad de la arquitectura propuesta.

En la figura 5.5 se muestra la implementación de un controlador con alimentación adelantada (feedforward) utilizando para ello una red neural. El neurocontrolador B es una réplica de la red neural A, la cual esta bajo proceso de entrenamiento. La red A esta conectada de manera tal que aprende gradualmente a desempeñarse como una planta inversa no conocida. Así, esta configuración implementa el inverso del control con alimentación adelantada similar al de la figura 5.4a. **d** es la entrada del controlador, siendo ésta la respuesta deseada (esperada) de la planta. La respuesta actual de la planta, debido a su entrada **x** (que es producto del neurocontrolador B), es **y**. El error usado para el entrenamiento de la red neural es la diferencia entre las señales de salida de las redes A y B. Aunque la red B monitorea a la red A después de cada paso del entrenamiento, **y** debe ser exactamente igual a **d** para que dicho error se reduzca a cero.

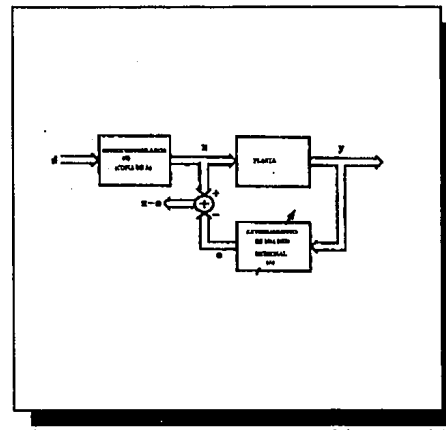


Fig 5.5 Controlador con alimentación por adelanto construído con una red neural.

Esta arquitectura de control es también llamada una **arquitectura de aprendizaje indirecto.** Las indudables ventajas de esta configuración son obvias. La red controladora puede ser entrenada en línea a partir de que esta bajo proceso de entrenamiento mientras su réplica realiza el trabajo de la función de control. Por tanto, a la larga no serán necesarias sesiones separadas de entrenamiento. Por otra parte, las entradas al neurocontrolador son las salidas esperadas de la planta. Así entonces, el entrenamiento del controlador puede realizarse fácilmente dentro de

la región de interés del dominio del vector de salida. Dicho entrenamiento en la región de interés puede ser nombrado **entrenamiento especializado**. Además, la red neural aprende continuamente, por lo tanto es **adaptiva**.

Una arquitectura de control apropiada para control y a la vez para un aprendizaje especializado en el dominio de la salida de una planta estática se muestra en la figura 5.6a. Este modo de control posee todas las ventajas del modo de control presentado en la figura 5.5, pero éste usa únicamente una sola copia de la red neural en lugar de las requeridas por el sistema de control de la figura 5.5.

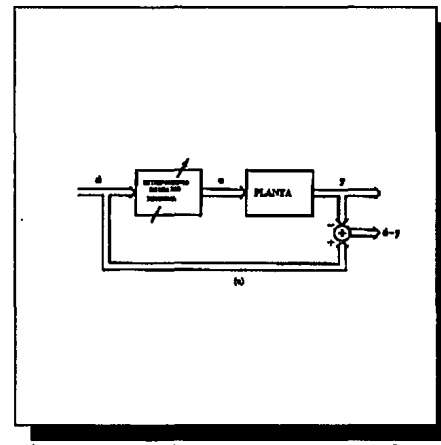


Fig 5.6 Arquitectura de control con aprendizaje especializado en línea: (a) Planta estática.

El objetivo del control de la arquitectura con aprendizaje especializado (figura 5.6a) es la obtención de la salida deseada **d** de la planta bajo la condición de que sea desconocida la entrada de la planta que produce esta salida. Para una planta dinámica, la red neural entrenada necesita que se le proporcione la secuencia de los recientes vectores de salida de la planta, además de que se le presente la salida de la planta. Esto permitirá la reconstrucción del estado actual de la planta basándose en su historial de salida más reciente. La figura 5.6b muestra una arquitectura de control con aprendizaje especializado de una planta dinámica de orden **k**. En este caso, los últimos **k** vectores de salida de la planta controlada incrementan la presente entrada de la red neural entrenada. Se debe de entender que las unidades de retardo ⌂, ubicadas en el lazo de retroalimentación, denotan el lapso de tiempo entre el muestreo de las

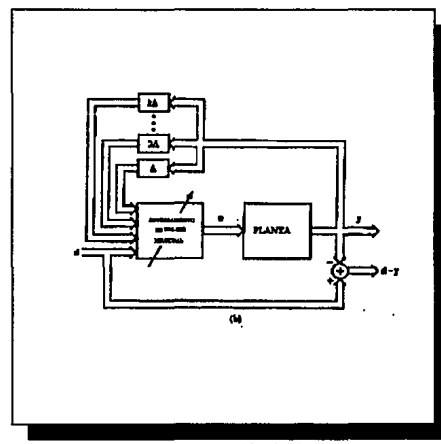


Fig 5.6 (b) Planta dinámica.

entradas, utilizadas para el entrenamiento, y que la planta sea controlada en tiempo real. Bajo dichas condiciones, **k** puede ser referido como el orden de la planta dinámica.

La arquitectura de control con aprendizaje especializado en línea que se presenta en la figura 5.6 tiene una gran deficiencia, la cual puede ocasionar un entrenamiento muy lento u otras características indeseables si la planta solo se conoce aproximadamente. Esta deficiencia radica en que no se puede aplicar el entrenamiento por error de retropropagación directamente a esta configuración debido a que el error entre la salida accesible **y** de la planta y su salida esperada **d** únicamente los conoce el usuario. El error **d-y** primero deberá reducirse para la entrada de la planta.

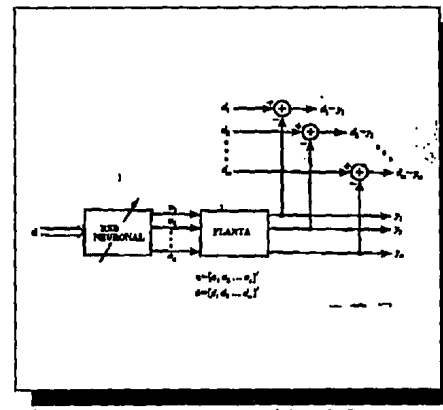


Fig 5.7 Retropropagación del error a través de la planta.

La red neural entrenada utilizando el error disponible a la salida de la planta, como se muestra en la figura 5.6, se ha rediseñado con más detalle como se ilustra en la figura 5.7.

5.2.3 APRENDIZAJE ESTRUCTURAL Y PARAMÉTRICO

Los métodos de aprendizaje paramétrico están basados en la suposición de que el modelo deseado es un miembro de una clase específica de modelos parametrizados. El término **aprendizaje estructural** se refiere al proceso de determinar que clase de modelos son apropiados para un conjunto de aplicaciones en particular. Al aplicar el modelo de retropropagación, por ejemplo, el aprendizaje estructural involucra cuántos niveles, cuántas unidades ocultas, y que clase de restricciones deben utilizarse para un problema en particular.

A una clase de modelos con un número muy grande de parámetros se le pueden incluir muchas clases diferentes de estructuras. Sin embargo, estimando que se tiene un número muy grande de paráme-

tros no la califica automáticamente como una técnica de aprendizaje estructural, debido a que no se garantiza que una estructura, aunque sea la mejor en cualquier sentido, será seleccionada por el proceso de estimación de parámetros, ni que la generalización (por ejemplo, la extrapolación de casos nuevos) será efectiva. Ajustando los parámetros para maximizar un criterio de desempeño que sólo esté definido en términos de encontrar los datos de entrenamiento que no tienen implicaciones automáticas para la selección de las estructuras apropiadas.

Si el número de parámetros se incrementa, la cantidad de datos necesarios para asegurar su estimación también se incrementará.

Cada uno de los diversos modelos implica un modo diferente de generalización; un error cuadrático medio en el conjunto de entrenamiento no implicará que la generalización se logre con éxito. Asegurar la generalización requiere que el número de ejemplos de entrenamiento sea varias veces más grande que el número de parámetros, ó que la clase de modelos parametrizados de alguna manera se asemeje a la función con la que se generó el conjunto de datos.

5.2.4 OBTENCIÓN DE INFORMACIÓN PARA EL ENTRENAMIENTO

Todos lo métodos de estimación de parámetros requieren de un procedimiento de entrenamiento en el cual las salidas deseadas, o esperadas, de la red están disponibles para un conjunto suficientemente variado de casos, por tanto, estos son métodos para aprendizaje supervisado. Pero, ¿ de dónde proviene esta información para el entrenamiento ?. La figura 5.8 ilustra el problema general; este tipo de figura se debe interpretar simplemente como íconos que representan cualquier método para el aprendizaje de funciones complejas por medio de aprendizaje supervisado. Sí se hace la suposición de que se quiere una red neural que aprenda a controlar una planta demasiado

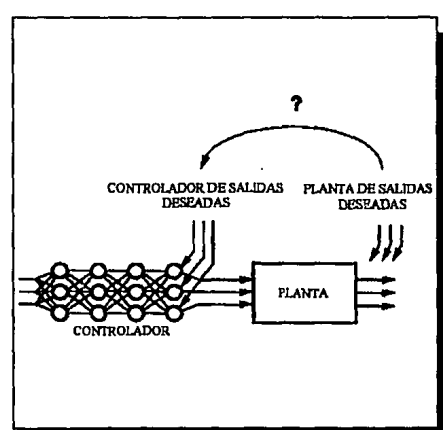


Fig 5.8 El problema de obtener datos de entrenamiento: Las salidas deseadas de la planta pueden conocerse, pero no las señales de control que las producen.

compleja, o de la cual se sabe muy poco, y que permita el uso de técnicas convencionales para el diseño de controladores. En un problema de control típico, se pueden tener las salidas esperadas de la planta, pero no las salidas esperadas de la red.

Una dificultad análoga al problema del entrenamiento de las señales que se le proporcionan a una red se presenta en la tarea de entrenar a las unidades ocultas de una red multinivel cuando las respuestas esperadas sólo se conocen para las unidades visibles (fig. 5.9). Para cualquier unidad oculta (la parte de la red que se encuentra entre la unidad oculta y las unidades de salida, por ejemplo) la parte de la red que determina como la actividad de una unidad oculta dada ejerce influencia sobre las unidades de salida, es similar a la planta de la figura 5.8. El problema de determinar las respuestas esperadas para las unidades ocultas es, por consiguiente, similar al problema de entrenar un controlador en base al comportamiento esperado de la planta, y los métodos que se aplican en un caso (tales como la retropropagación o el aprendizaje reforzado) también se aplican al otro.

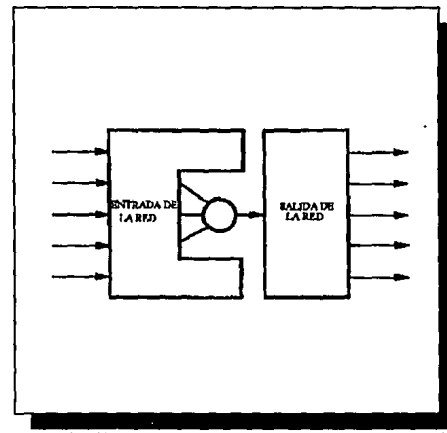


Fig 5.9 El problema de entrenar a las unidades ocultas en una red neural: las salidas deseadas no están directamente disponibles para una unidad oculta.

La información requerida para el entrenamiento en el aprendizaje supervisado, se puede obtener de las tareas propias de control. Ningún intento se ha hecho para describir en detalle los tipos de plantas y los problemas de control que son de interés para los teóricos en el área de control, ni tampoco se ha hecho justicia a los métodos altamente desarrollados que existen para la predicción, la identificación del sistema, y el control adaptivo. Si se piensa en la planta como un simple mapeo Entrada/Salida y al controlador como uno con alimentación adelantada, es suficiente para la mayoría de los métodos sugeridos que a continuación se describirán; si dichos métodos se modifican apropiadamente, también son aplicables a tipos más complejos de plantas y pueden ser integrados a métodos de control por retroalimentación. Hay que hacer la aclaración de que las redes neurales no son sólo apropiadas para control por alimentación adelantada, y que no sólo se deban usar en ausencia de lazos de control por retroalimentación más convencionales.

5.2.4.1 COPIANDO UN CONTROLADOR EXISTENTE

Si ya existe un controlador eficiente para una planta, entonces la información requerida para entrenar a una red neural se puede obtener de este controlador, como se muestra en la figura 5.10. La salida esperada de la red para una entrada determinada, es la salida del controlador existente para esta entrada. La red aprende a imitar (copiar) al controlador existente. Es posible que sea cuestionable la utilidad de este método, ya que si existe un controlador efectivo, ¿ porqué sería útil tener otro en la forma de una red neural ?. Para esta interrogativa se tienen dos indudables respuestas; la primera, el controlador existente puede ser un dispositivo imprácticó de usar (como lo es una persona), y, la segunda, la red adaptiva puede ser capaz de formar un método de control efectivo en base a la representación del estado del sistema que es más fácil de medir que la representación requerida por el controlador existente.

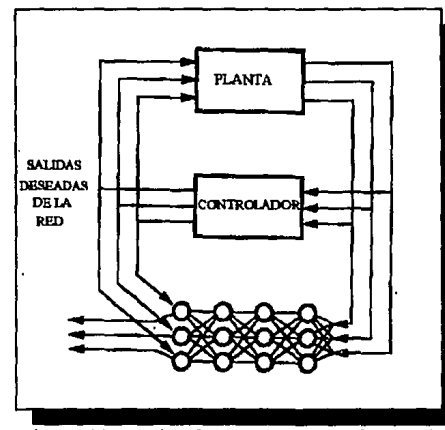


Fig 5.10 Copiando un controlador existente con una red neural.

5.2.4.2 PREDICCIÓN ADAPTIVA

La figura 5.11 ilustra la idea básica para entrenar a una red neural como predictor. Supongase que las líneas a través de la parte superior de la figura transportan las señales cuyos valores son los que se desean predecir. Para simplificar, también se hace la suposición que estas mismas señales proporcionan la información desde la cual la predicción será hecha. Para propósitos de entrenamiento, la entrada a la red neural consiste de valores con retardo de las señales, y la salida esperada consiste de los valores actuales de las señales. Por tanto, la red trata de igualar los valores de la señal actual ajustándose en función de sus valores anteriores. Entonces cuando la entrada a la red pasa por las unidades de retardo, la salida de la red es una predicción de los valores que tendrán las señales en el futuro.

Suponiendo que cada unidad de retardo (mostradas en la figura 5.11) retrasa su entrada por τ intervalos de tiempo, y que la red requiere una cantidad insignificante de tiempo para calcular su salida a partir de su entrada, entonces la red entrenada proporciona la estimación de los valores de las señales que se tendrán τ lapsos de tiempo después. Esta aproximación a la predicción adaptiva se apoya en una supuesta clase parametrizada de modelos para la relación funcional entre los valores actuales y los pasados de las señales y sus valores con retardo, o de manera equivalente entre los valores próximos de las señales y sus valores actuales.

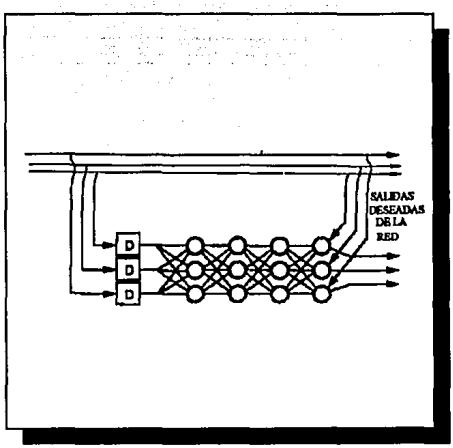


Fig 5.11 Usando una red neural para predicción adaptiva.

El esquema ilustrado en la figura 5.11 es sólo un caso especial muy simple de esquemas de predicción más generales donde las unidades de retardo en cascada (conexión de líneas de retardo, CLR) se utilizan como entradas al modelo, ó en donde se usan modelos recursivos, ó autoregresivos, en cuyos valores pasados de las salidas del modelo forman parte de la entrada.

5.2.4.3 IDENTIFICACIÓN DEL SISTEMA

La información para el entrenamiento se puede obtener de la observación del comportamiento Entrada/Salida de la planta, como se sugiere en la figura 5.12, en donde la red recibe la misma entrada que la planta, y la salida de la planta es la salida esperada de la red. La figura sólo muestra el caso en donde el modelo es un mapeo de las entradas a las salidas de la planta, pero comúnmente se emplean tipos de modelos más complejos. Por ejemplo, la entrada al modelo puede consistir de varios valores con retardo de las entradas de la planta (y la predicción adaptiva es un caso especial de identificación del sistema), y el modelo puede ser recursivo.

La identificación del sistema es una parte esencial de algunos métodos para control adaptivo. En una aproximación a control adaptivo, se selecciona una clase se selecciona una clase de modelos para propósitos de identificación tal que cada modelo en la clase es un sistema por el cual se puede diseñar un controlador efectivo utilizando probados métodos de diseño conocidos para plantas. Por tanto, se puede expresar una ley (regla) de control en términos de los parámetros estimados que produce el procedimiento de identificación del sistema. Ésta será una ley de control efectivo si la planta puede ser modelada en forma adecuada dentro de la supuesta clase de modelos.

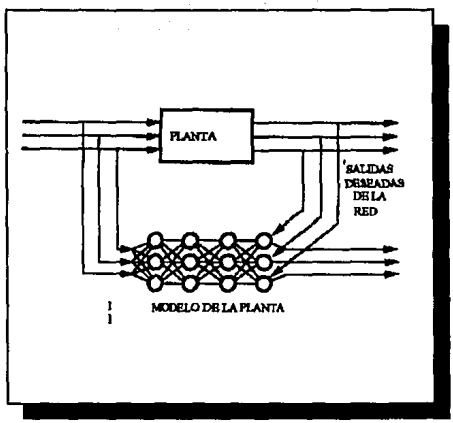


Fig 5.12 Usando una red neural para identificación del sistema.

Utilizar una red neural no lineal en lugar de un procedimiento más convencional de identificación del sistema, no extiende inmediatamente la aplicabilidad del control adaptivo a los sistemas que las redes no lineales son capaces de identificar. Un sistema que pueda ser identificado no implica que pueda ser controlado. Tal vez es necesario contar con técnicas para diseñar un controlador que se apliquen a la clase de sistemas no lineales identificados por las redes. Quizás sea posible desarrollar una teoría de control en esta dirección, pero a causa de que las redes pueden representar esencialmente cualquier sistema, no hay una razón forzosa para esperar que al invocar ideas sinápticas (coneccionistas), probablemente por sí mismas, se emita cualquier aportación a la teoría de control no lineal.

5.2.4.4 IDENTIFICACIÓN DEL SISTEMA INVERSO

La figura 5.13a, muestra como una red neural adaptiva se puede utilizar para identificar el inverso de la planta. A diferencia de la situación que se muestra en la figura 5.12, aquí la entrada a la red es la salida de la planta, y la salida esperada de la red es la entrada de la planta. Si la red puede ser entrenada para que iguale estas salidas esperadas, entonces se implementará un mapeo que es el inverso de la planta. Una vez que se tenga tal

inverso, éste se puede utilizar para propósitos de control, como se muestra en la figura 5.13b. La salida deseada de la planta se proporciona como entrada a la red; la salida resultante de la red será entonces usada como entrada a la planta. Si la red es una planta inversa, entonces esta entrada a la planta produce dicha salida deseada de la planta (aclarando, esta simple descripción pasa por alto muchos detalles de un problema real de control).

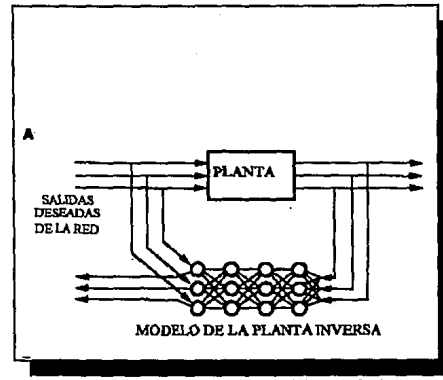


Fig 5.13 (a) Identificación del inverso de la planta usando una red adaptiva.

En la identificación de la inversa, surge un gran problema cuando muchas entradas de la planta producen la misma salida, por ejemplo, cuando la planta inversa no está bien definida. En este caso, la red intentará mapear la misma entrada de la red hacia muchas respuestas esperadas diferentes.

La mayoría de los métodos de estimación de parámetros tienden a promediar sobre varias respuestas esperadas y de este modo produce un mapeo que no necesariamente es inverso. El uso de redes no lineales para la identificación de plantas inversas no lineales tiene una utilidad inmediata para control.

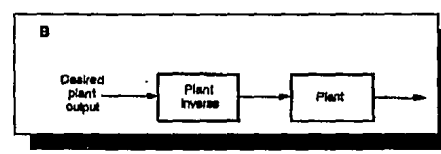


Fig 5.13 (b) El inverso de la planta se utiliza para propósitos de control.

5.2.4.5 Diferenciación del Modelo.

Este método para entrenar un controlador, confía más en la retropropagación que en métodos generales para redes, pero éste es una forma de usar redes neurales apropiadas para control adaptivo. El método se ilustra en la figura 5.14. El algoritmo de retropropagación se usa para identificar la planta como la que se muestra en la figura 5.12 (las señales de entrenamiento para esta etapa de identificación no se muestra en la fig. 5.14), resultando un modelo de planta adelantada con redes neurales multinivel.

La utilidad de un modelo adelantado que tiene esta forma es que se puede calcular eficientemente la derivada de la salida del modelo con respecto a su entrada por medio del proceso de retropropagación. Consecuentemente , los errores propagados entre la salida actual y la deseada de la planta, que regresan a través del modelo adelantado, producen el error en la señal de control, el cual puede ser usado para entrenar otra red, ó a otra clase de sistema con aprendizaje supervisado. En la figura 5.14 este proceso de retropropagación se ilustra con la línea punteada a través de una segunda red neural multinivel que lo usa para aprender la regla de control. Este método tiene ventajas sobre la identificación directa de una planta inversa que no esté bien definida. De seguro, para aplicar esta idea básica sólo se necesita un modelo en una forma tal que pueda ser diferenciado; éste necesita no ser una red multinivel. Sin embargo, utilizar una red neural multinivel que pueda ser entrenada por retropropagación es buena idea, debido a que el mecanismo de retropropagación puede ser usado tanto para ajustar los parámetros de la red como para diferenciar el modelo resultante.

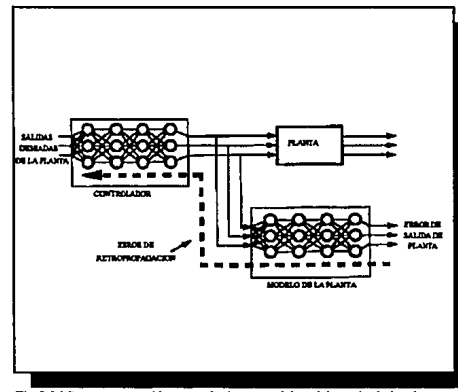


Fig 5.14 Retropropagación a través de un modelo adelantado de la planta para determinar los errores del controlador.

Este método para obtener la información de entrenamiento para un controlador adaptivo ilustra un principio general que puede ser explotado de otras maneras en problemas de control. Este principio selecciona la clase de modelos para identificación del sistema basándose, parcialmente, en la existencia de técnicas especializadas aplicables a los modelos dentro de la clase. Por ejemplo, en aproximaciones convencionales a control adaptivo, una clase de modelos se selecciona tal que las técnicas para la síntesis del controlador puedan ser aplicadas a cualquier sistema representable con la clase (por ejemplo, sistemas lineales invariantes en el tiempo). Cuando un modelo está especificado por el proceso de estimación de parámetros, se puede diseñar un controlador bajo la suposición de que este modelo es una aproximación al modelo de la planta. De la misma forma, al determinar un modelo de planta en la forma de una red neural multinivel con funciones exclusivas diferenciables, permite que se aplique el proceso de retropropagación para estimar cuantas entradas a la planta deben ser alteradas para cambiar la salida de la planta en la forma deseada.

Es interesante resaltar el hecho de que el método que utiliza retropropagación para diferenciar un modelo para propósitos de control está más relacionadamente cerca a los métodos de control óptimo que su aplicación en la estimación de parámetros para identificación del sistema.

5.2.5 APRENDIZAJE REFORZADO

Existen algunas tareas de aprendizaje en control que requieren de métodos que no precisamente se caracterizan como métodos de aprendizaje supervisado. Por ejemplo, si se desea ajustar una regla de control para que se mejore el desempeño de una planta, midiéndolo con cierto parámetro que de alguna forma evalúe el comportamiento completo de la planta esperando maximizar la eficiencia en energía de la planta medida en el tiempo. Los métodos de control óptimo se aplican siempre que los modelos de la planta y los parámetros de desempeño estén lo suficientemente aproximados y que se puedan manipular con facilidad. Sin embargo, en situaciones menos estructuradas, puede ser posible mejorar el desempeño de la planta, con respecto al tiempo, por medio de métodos de aprendizaje en línea que realizan lo que se denomina **Aprendizaje Reforzado.** Puesto que la medida de desempeño para un sistema supervisado se define en términos de un conjunto de valores esperados a través de un criterio de error conocido (por ejemplo, el error cuadrático medio), el aprendizaje reforzado maneja el problema de mejorar el desempeño evaluándolo con cualquier parámetro cuyos valores resultantes se pueden suministrar al sistema de aprendizaje. Consecuentemente, en tareas de este tipo se tienen señales de control deseadas (aquellas que conllevan hacia el desempeño óptimo de la planta, pero que el sistema de aprendizaje no ha determinado cuales son debido a que no se tiene una acción lo suficientemente inteligente para actuar como instructor (maestro) de este tipo. El problema es encontrar estas señales de control óptimo, no sólo para recordarlas (retenerlas como experiencia) y generalizar a partir de las mismas.

Así entonces, el aprendizaje reforzado involucra muchos de los resultados relacionados con la obtención de la información de entrenamiento requerida para el aprendizaje supervisado cuando dicha información no era conseguible directamente (fig. 5.8). Sin embargo, en el caso del aprendizaje reforzado, la situación es más general que la mostrada en la figura 5.8 en la que en lugar de tratar de determinar las salidas esperadas del controlador a partir de las respuestas esperadas de la planta, se trata de determinar las salidas esperadas del controlador, o los cambios deseados en sus salidas, que deberán conducir hacia los incremen-

tos en la medida del desempeño de la planta, una medida no necesariamente en términos de las respuestas esperadas de la planta. Esto se ilustra en la figura 5.15, la cual ilustra un controlador con aprendizaje reforzado interactuando con una planta y recibiendo una "señal de evaluación" generada por un "crítico" (evaluador implementado con un autómata) capaz de evaluar el desempeño de la planta.

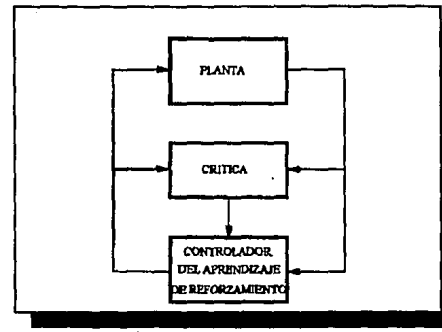


Fig 5.15 Sistema de control con aprendizaje reforzado.

5.2.5.1 EL PAPEL DEL APRENDIZAJE REFORZADO EN CONTROL

El aprendizaje reforzado es una aproximación muy general al aprendizaje que se puede aplicar cuando el conocimiento requerido para aprendizaje supervisado no esta disponible. La generalidad del aprendizaje reforzado se obtiene cuando se compara el costo de la eficiencia con los métodos supervisados especializados, siempre y cuando estos métodos sean aplicables. Consecuentemente, aunque cualquier tarea de control que requiera aprendizaje pueda ser formulada como un problema de aprendizaje reforzado (cuando el objetivo es maximizar la medida de desempeño), será mejor usar métodos más especializados, siempre y cuando sea posible que tomen total ventaja del conocimiento disponible. Por ejemplo, si la medida de desempeño es el error basado en los valores esperados conocidos, entonces el método de aprendizaje reforzado puede resolver el mismo problema que un método supervisado resolvería, pero lo realizará sin hacer uso del hecho de que la medida de desempeño tiene una forma específica; por tanto, éste tomará más tiempo que un método supervisado especializado.

Por tanto, el uso de aprendizaje reforzado es más fácilmente justificable cuando se tiene una carencia real del conocimiento requerido para aplicar métodos de aprendizaje más especializado. Algunos métodos en control adaptivo implementan una ley de control diseñada en base a un modelo, bajo la suposición de que el modelo proporciona una caracterización precisa y completa de la planta que será controlada. El modelo de la planta se utilizará como base para una aproximación convencional de programación dinámica para control óptimo; o quizás se pueda obtener una clase de modelos espacio/tiempo en la forma de una

red neural multinivel, para que las señales de control apropiadas se calculen por "retropropagación a través del tiempo". Aunque estas aproximaciones todavía sean vistas como casos de aprendizaje reforzado, debido a que éstos no inician con un conocimiento de las salidas esperadas momento a momento, tanto para la planta como para el controlador, estos casos no emplean la clase de búsqueda directa en línea en control espacial que por lo general está asociado con aprendizaje reforzado.

Los métodos de aprendizaje reforzado tienen un papel definitivo debido a que los métodos basados en modelos no son necesariamente efectivos con modelos inexactos, y será necesario, de cualquier forma, un comportamiento exploratorio para lograr modelos precisos. Tiene sentido tomar ventaja de esta exploración para modificar directamente reglas de control a través del aprendizaje reforzado de manera que se obtengan mejoras en el desempeño mientras que los parámetros del modelo están siendo ajustados. Esta estrategia es apropiada para ser utilizada especialmente en tareas que involucraban la optimización en tiempo del desempeño de plantas no lineales. Ajustando la regla de control con ayuda directa de la medida de desempeño es posible acortar la cadena de suposiciones de las cuales un método es dependiente. Además, en algunos problemas puede no haber el tiempo suficiente para tomar ventaja de un modelo elaborado, especialmente si se requieren métodos de programación dinámica para generar acciones de control a partir de dicho modelo. Razones como estas, las cuales involucran aspectos de los intercambios convenientes usuales entre la adquisición del conocimiento y su uso para control, sugieren que el ajuste directo de una ley de control por medio de aprendizaje reforzado puede jugar un papel útil en control, y que cuando se combina con métodos críticos adaptivos, será más útil para perfeccionar el desempeño de la planta con respecto a medidas complejas de desempeño.

5.2.6 EXCITACIÓN PERSISTENTE

Es generalmente aceptada la importancia de la clase de entradas que son usadas en el entrenamiento adaptivo y en los sistemas de aprendizaje. El conjunto de entrenamiento tiene que ser representativo de la clase completa de entradas a que el sistema será sujeto. Esto asegurará que el sistema responderá en la forma deseada cuando se le aplique una entrada no incluida en el conjunto de entrenamiento. Este concepto, referido como una **Excitación Persistente**, ha sido extensivamente tratado en la teoría de control adaptivo convencional tanto en el contexto de identificación como de problemas de control.

Sea θ un vector de parámetros ajustables en un sistema y sea θ^* un vector constante tal que cuando $\theta = \theta^*$, el sistema tiene la misma función de transferencia que el modelo. El vector θ^* debe ser único o pertenecer a un conjunto $\mathbf{S}$. En el segundo caso, el sistema dado tiene el mismo comportamiento entrada/salida que el modelo dado para todos los valores constantes de $\theta \in \mathbf{S}$. Si entradas específicas se aplican al sistema y al modelo de referencia, el error de salida tiende a cero asintóticamente con el tiempo aún si $\theta \notin \mathbf{S}$. Sin embargo, si la entrada está persistentemente excitando, $\theta(\mathbf{k}) \longrightarrow \theta^* \in \mathbf{S}$. Si θ^* es único, también implica que los parámetros del sistema desconocidos se pueden identificar con exactitud.

El concepto de excitación persistente es importante para los procedimientos de identificación y control de sistemas no lineales que utilizan redes neurales. Sea U el conjunto compacto al cual pertenece la entrada u. Si se selecciona una entrada específica $u \in U$ (por ejemplo, una simple senoidal), el error de salida rápidamente se aproxima a cero si se utiliza retropropagación para ajustar los pesos de la red neural. Sin embargo, si L y L' representan a los operadores de la planta y del modelo de identificación respectivamente, L' no esta "cerca" a L en cualquier sentido, por ejemplo la norma $\|Lu - L'u\|$ no es pequeña para todo $u \in U$. Esto será evidente si se considera una entrada u no incluida en el conjunto de entrenamiento.

Si la entrada a la planta (y al modelo de identificación) es lo suficientemente general y los pesos de la red neural se ajustan para una período suficientemente largo, el error de salida puede hacerse pequeño para cualquier entrada en U. La entrada general mencionada puede entonces considerarse para estar persistentemente excitando.

Si el operador L' aproxima a L en algún sentido, el modelo puede usarse fuera de línea en el sitio de la planta para propósitos de predicción. Sin embargo, si el modelo exacto a seguir se consigue únicamente al ajustar los parámetros del modelo a cada instante, éste tendrá limitada la habilidad predictiva, pero se puede utilizar eficientemente en línea.

5.2.7 CONTROL ADAPTIVO

El control adaptivo paramétrico se refiera al problema de controlar la salida de un sistema con estructura conocida pero parámetros desconocidos. Para hacer analíticamente tratable a dicho problema, en la teoría de sistemas adaptivos, la planta a ser controlada se considera lineal e invariante en el tiempo con

parámetros desconocidos. Estos parámetros pueden considerarse como los elementos de un vector **p**. Si **p** es conocido, el vector paramétrico θ de un controlador puede ser elegido como θ^* tal que la planta y el controlador fijo en conjunto se comporten como un modelo de referencia descrito por una ecuación en diferencias (ó diferencial) lineal con coeficientes constantes. Si **p** es desconocido, el vector $\theta(t)$ tiene que ser ajustado en línea usando toda la información disponible que le sea concerniente al sistema.

Dos aproximaciones distintas al control adaptivo de una planta no conocida son el control directo y control indirecto. En control directo, los parámetros del controlador se ajustan directamente para reducir un poco de la norma del error de salida. En control indirecto, los parámetros de la planta se estiman como $p'(t)$ en cualquier instante de tiempo y el vector paramétrico $\theta(t)$ del controlador se elige considerando que $p'(t)$ representa el valor verdadero del vector paramétrico de la planta. Cuando la planta se considera lineal e invariante en el tiempo, el control adaptivo, tanto directo como indirecto, resulta un sistema no lineal. Cuando la planta es no lineal y dinámica (por ejemplo, el valor actual de su salida depende de los valores anteriores de la entrada y de la salida respectivamente), una red neural puede usarse como controlador, tal como se ilustra en la figura 5.16. Esta corresponde a control directo.

5.2.7.1 CONTROL DIRECTO

En la teoría de control adaptivo directo convencional, los métodos para ajustar los parámetros de un controlador se basan en el error de salida.

En la actualidad, no están disponibles los métodos para ajustar directamente los parámetros del controlador (la red neural N_c en la fig.5.16) en forma estable en base al error de salida. Esto se debe a la naturaleza no lineal de la planta y del controlador. Tampoco la retropropagación puede ser usada para generar las derivadas parciales deseadas. Por tanto, hasta que los métodos de control directo sean desarrollados, el control

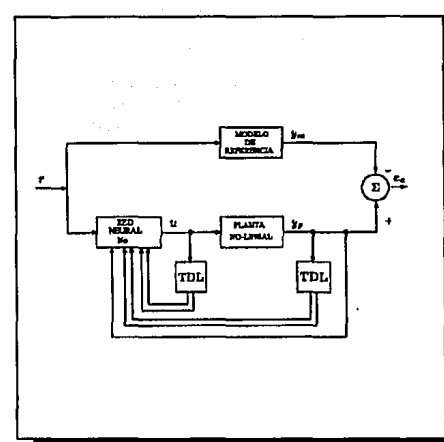


Fig 5.16 Control adaptivo directo usando redes neurales.

adaptivo de sistemas dinámicos no lineales tiene que llevarse a cabo utilizando métodos de control indirecto.

5.2.7.2 CONTROL INDIRECTO

Cuando se utiliza control indirecto para controlar un sistema no lineal, la planta se parametriza usando algún modelo específico, y los parámetros del modelo se actualizan utilizando el error de identificación. Los parámetros del controlador en turno se ajustan por la retropropagación del error (entre las salidas del modelo identificado y del modelo de referencia) a través del modelo identificado. Un diagrama a bloques de tal sistema adaptivo se muestra en la figura 5.17.

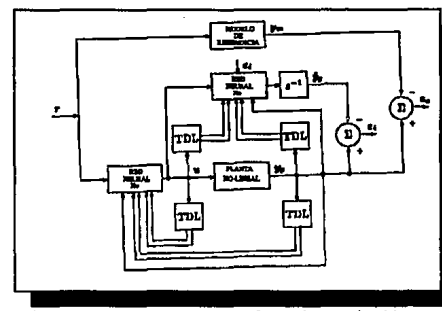


Fig 5.17 Control adaptivo indirecto usando redes neurales.

Tanto la identificación como el control pueden llevarse a cabo, en cualquier momento, durante ó después del procesamiento de datos, en intervalos finitos. Cuando una perturbación externa y/o ruido no están presentes en el sistema, es razonable ajustar sincronizadamente los parámetros del control y de la identificación. Sin embargo, cuando un ruido sensible o una perturbación externa se presenta, la identificación se lleva a cabo a cada instante mientras que la actualización de los parámetros de control se logra sobre una escala más pequeña de tiempo, para asegurar fortaleza (por ejemplo, los parámetros de control se ajustan con menor frecuencia que los parámetros de identificación).

5.2.7.3 INFORMACIÓN PRELIMINAR

En la teoría de control adaptivo convencional, en donde la planta se considera lineal e invariante en el tiempo con parámetros desconocidos, las leyes de estabilidad adaptiva han sido derivadas únicamente con la suposición de que la información preliminar considerable respecto a la función de transferencia de la planta está disponible. En particular, se considera que :

(a) el signo de la ganancia de alta frecuencia,
(b) se conoce el orden de la planta,
(c) se conocen los grados relativos de la función de transferencia,
(d) los ceros de la planta se encuentran en el interior del círculo unitario.

Es razonable asumir que más información preliminar con respecto a las características entrada/salida se tiene que suponer, cuando la planta es no lineal, antes que los resultados teóricos puedan ser derivados para controlarla en una forma estable. por ejemplo, si los ceros de la función de transferencia de la planta se encuentran afuera del círculo unitario, es posible para la entrada de la planta crecer sin límites cuando el error de salida tiende a cero. Las consideraciones (b) y (c) están hechas para asegurar la existencia tanto del controlador deseado como del modelo de referencia.

5.2.7.4 EL MODELO DE REFERENCIA

Se considera que este modelo es lineal debido a que la teoría de sistemas lineales esta bien desarrollada y los métodos de selección de modelos lineales que tienen propiedades deseadas están bien establecidos. En contraste a esto, no están actualmente disponibles los métodos apropiados para determinar modelos generales no lineales con estados transitorio y estable deseados.

Sin importar que el modelo de referencia sea lineal o no lineal, una valiosa consideración práctica es que el mapeo dinámico representado por el sistema, el cual contiene al controlador y a la planta no lineal, se debe aproximar al modelo de referencia cuando $k \longrightarrow \infty$.

5.2.8 REDES NEURALES COMO CONTROLADORES ADAPTIVOS

¿ En dónde deben ser más efectivas las redes neurales en los problemas de control adaptivo ?. Esta es una cuestión fundamental a la que se debe dar respuesta. Desde que el control adaptivo de sistemas lineales esta altamente desarrollado en la actualidad, las aplicaciones principales de redes neurales son limitadas para estar en control de sistemas no lineales.

Cuatro clases distintas de problemas, donde el control no lineal puede ser deseable, se describen a continuación. En cada caso, se supone que la planta se define por una ecuación en diferencias (ó diferencial) no lineal compleja.

1. La primera clase incluye las plantas cuyas ecuaciones que las rigen están completamente especificadas. Por consiguiente, un controlador puede ser diseñado por lo menos en teoría. De cualquier modo, si se determina que un controlador puede resultar demasiado complejo para propósitos prácticos, entonces será preferible utilizar una red neural en una forma adaptiva.

2. En algunas situaciones (por ejemplo, estructuras espaciales flexibles) la planta puede ser estable pero tener una respuesta transitoria pobre para las condiciones iniciales dentro del dominio de interés. Además, un controlador lineal con retroalimentación puede no ser satisfactorio para extensas condiciones iniciales. En este caso, la identificación de la planta usando una red neural por un período grande de tiempo seguido por el uso de un controlador no lineal puede dar como resultado una mejor respuesta completa.

3. En algunos casos (por ejemplo, en procesos químicos) la planta puede operar eficientemente en varios estados de equilibrio. El propósito de un controlador adaptivo, en este caso, es el de estabilizar el sistema alrededor de uno de esos estados de equilibrio dependiendo de la información disponible en cualquier momento.

4. En muchas situaciones, el control puede lograse en base a patrones de salida. El estado espacial en un proceso es dividido en regiones disjuntas las cuales son equivalentes para propósitos de control. Las redes neurales pueden utilizarse para generar la entrada óptima correspondiente a cada región.

5.2.9 REDES NEURALES ADAPTIVAS EN CONTROL DE PROCESOS QUÍMICOS

El control de procesos químicos ofrece un conjunto fructífero de cuadros de prueba y de criterios de desempeño para el desarrollo de nuevos algoritmos de control. Los procesos químicos muchas veces son demasiado no lineales y difíciles de controlar, aún cuando sea fácil hacer sus modelos aproximados.

El control de procesos de plantas químicas es una aplicación atractiva a causa de los beneficios potenciales tanto para la investigación de redes adaptivas como para el control actual de procesos químicos. El control de sistemas químicos tales como los reactores y las columnas de destilación, proporcionan, en

realidad, la mayor aplicación actual de control adaptivo, el cual incluye el conjunto de técnicas contra las cuales los controladores basados en redes adaptivas competirán en los mercados científicos y comerciales. Los estudios de la convergencia, fortaleza y estabilidad que se han hecho para controladores adaptivos convencionales, eventualmente también se harán para controladores basados en redes. Sin embargo, en la búsqueda de sistemas con propiedades probables, el control tradicional ha descuidado algunos métodos de control atractivos pero más complejos. Trabajar con redes adaptivas puede estimular al control adaptivo al sugerir nuevas aproximaciones, algoritmos, y arquitecturas.

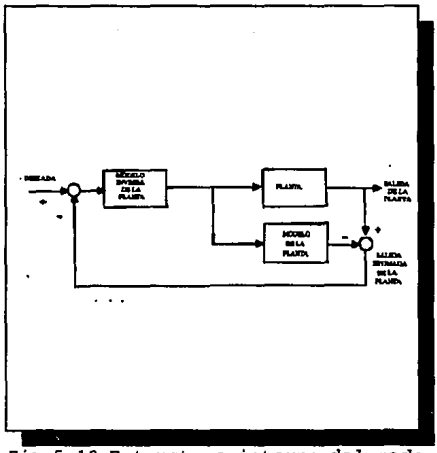


Fig 5.18 Estructura interna del modelo de control.

Una de las utilidades más promisorias de las redes adaptivas es como modelo de la planta o proceso que es controlado por un controlador convencional. El modelo de control interno se utiliza tanto para modelo de la planta como para modelo de la planta inversa. Una red neural adaptiva puede usarse en lugar del modelo y del modelo inverso (Ver figura 5.18).

5.2.9.1 MOTIVACIONES PARA EL CONTROL DE PROCESOS QUÍMICOS

A pesar del trabajo extensivo en controladores autoajustables y en control de modelos de referencia, existen muchos problemas en las industrias de procesamiento químico, en las cuales son inadecuadas sus técnicas actuales. Muchas de las limitaciones de los actuales controladores adaptivos surgen al tratar de controlar sistemas no lineales pobremente modelados. Para la mayoría de estos procesos se dispone de amplia información, obtenida de las anteriores ejecuciones de los procesos, no obstante es difícil formular sus modelos exactos. Aquí es precisamente donde se puede esperar la utilidad de las redes neurales adaptivas.

La gran mayoría de los procesos químicos son extremadamente no lineales. Las no lineridades pueden ser intrínsecas a la física o a la química de un proceso así como en la extracción supercrítica, en cuyo comportamiento de fase compleja conduce a una sensible dependencia de operar en condiciones de control y operación. Las no linearidades también pueden surgir a través de un acoplamiento cerrado de procesos simples. Por ejemplo cuando integración de calor se utiliza para ahorrar energía, el proceso se convierte en más fuertemente acoplado, más multivariado y más difícil de controlar.

Las viejas tecnologías también presentan cambios. La destilación es un proceso altamente no lineal, y uno de los problemas de control más ampliamente estudiados. Los procesos como los de fundición del aluminio pueden ofrecer grandes cambios. Diferentes relaciones de enfriamiento conducen a diferentes mandos de operación que producen aluminio con propiedades diferentes. De la experiencia obtenida de anteriores ejecuciones del proceso se esperará producir un esquema óptimo de control.

Los sistemas de reacción química también presentan problemas de control importantes y ampliamente estudiados. Se han hecho extensivos estudios teóricos y prácticos de los **Reactores Tanque de Movimiento Continuo** (**RTMC**). Aunque tales reactores puedan ser (aproximadamente) descritos pos ecuaciones simples, éstos exhiben un comportamientos complejos tales como múltiples estados de equilibrio y comportamientos caóticos y periódicos. Los bioreactores son un ejemplo de dichos reactores que presentan problemas especiales. Son difíciles de modelar debido a la complejidad de los organismos que viven dentro de ellos, y que puede variar de un conjunto a otro. También son difíciles de controlar a causa de que frecuentemente no se pueden medir en línea las concentraciones de los químicos que están siendo producidos o metabolizados. Los bioreactores también tienen marcados mandos de operación diferentes, dependiendo si los microbios (bacterias o levaduras) están rápidamente creciendo o están fabricando el producto. El control basado en el modelo de dichos reactores ofrece de este modo un problema dual : determinar el modelo real del proceso y determinar las leyes de control efectivo en presencia de modelos inadecuados del proceso y de procesos altamente no lineales.

El problema del bioreactor (ó RTMC) es el más accesible para la experimentación y uso de redes neurales adaptivas. Las fuertes no linearidades dan el sensible comportamiento de los parámetros en los modelos y en la estructura de los modelos. Se pueden observar varios comportamientos diferentes en mandos de operación diferentes, incluyendo la dinámica compleja, sin embargo, se puede capturar suficiente de estos comportamientos en dos ecuaciones para ofrecer un cambio significante de control. Versiones multivariadas del problema surgen cuando se espera utilizar un control de temperatura y relaciones de alimentación

de diferentes nutrientes para optimizar la fabricación de diferentes productos.

5.2.9.2 Comparación entre el Control de Procesos Químicos y El Control de Robots.

Aunque se tienen varias similitudes, el control de procesos y el control de robots difieren de varias maneras importantes. Las plantas químicas son en su mayoría operadas continuamente, y tratan de responder y minimizar a los efectos de las perturbaciones antes que, por ejemplo, seguir las trayectorias especificadas. Es necesario más trabajo para planear métodos más poderosos para modelos de aprendizaje de sistemas dinámicos (ó el aprendizaje para corregir modelos derivados de los principios que describen a las plantas reales) y en el desarrollo de esquemas de aprendizaje reforzado apropiados para aplicaciones en control.

Por otro lado, algunos aspectos del control de procesos químicos son tan simples como en el control de robots. Los problemas de control de procesos químicos no tienen las cuestiones complicadas de interpretación visual tan comunes en la robótica. Si lecturas bastante exactas (aunque algunas veces erróneas) de temperaturas y presiones están virtualmente disponibles continuamente y si se tiene una identificación bastante clara, entonces un problema de control puede ser propuesto. Cuando se tiene un conjunto exacto de ecuaciones que describen la planta, la manera en que se pueden utilizar eficientemente las mediciones de abundancia y frecuencia para mejorar un modelo del proceso, es una cuestión propia de los parámetros ("identificación del sistema" en control). Cuando no se tiene dicho conjunto, son necesarias ecuaciones de forma más general tales como las que ofrecen las redes adaptivas.

Una manera de acercarse al problema de desarrollar problemas de prueba y criterios de desempeño es hacerse la siguiente pregunta : ¿ Qué es lo que ocasiona que un sistema sea difícil de controlar ?. Mucha de la dificultad de controlar cualquier proceso viene de la complejidad de dicho proceso. Esta complejidad se puede describir de diversas formas. Los sistemas altamente no lineales son difíciles de controlar, particularmente cuando tienen dinámicas complejas (tales como las inestabilidades de ciclos límite y caos). Las dificultades frecuentemente pueden presentarse por restricciones, ya sea en los parámetros de control (por ejemplo, existe una razón máxima a la cuál el sistema puede ser calentado) ó en los mandos de operación (por ejemplo, el calentamiento por arriba de cierto punto conduce a una reacción fugitiva). La falta del conocimiento exacto del proceso, de seguro, hace el control más difícil.

Otras dificultades tales como los retrasos de tiempo significantes entre el tiempo que dura una acción de control y el tiempo de respuesta, son más características del control de procesos químicos. Los retardos en la respuesta indican que el sistema no es invertible. Estos también crean un problema temporal de asignación de crédito: no es tan trivial determinar cual resultado debe ser acreditado a una determinada acción de control. Muchos procesos químicos también tienen el problema de asignación de crédito espacial : los sistemas tienen muchos sensores y controladores (son de múltiple entrada-múltiple salida), y no es tan claro como conectar los sensores y controladores o como cambiar la interacción de los múltiples controladores.

El control óptimo de muchas plantas químicas también requieren sistemas que hagan uso de predicciones del comportamiento futuro. Esto puede ocurrir en procesos de variación en tiempo tales como procesos en conjunto (lote), donde se quiere optimizar sobre el tiempo, o puede ocurrir en sistemas continuos bastante simples, donde las no linearidades pueden causar una "respuesta inversa" (un cambio a un parámetro de control que puede inicialmente mover el proceso en dirección opuesta con un efecto a largo plazo).

Por último, los problemas químicos son diferentes de los problemas de control en robótica en que su experimentación debe ser mucha más limitada y conservadora. Aunque pueda ser factible tener un robot que trate, no obstante falle, veinte veces poner una pieza en una caja, no es aceptable tener un reactor que trate, no obstante falle, veinte veces evite fabricar un producto que esté fuera de especificación.

5.2.9.3 CRITERIOS DE DESEMPEÑO EN EL CONTROL DE UN BIOREACTOR.

Los sistemas químicos pueden ser relativamente simples dado que tienen unas cuantas variables, más sin embargo muy difíciles de controlar debido a las fuertes no linearidades, las cuales son difíciles de modelar exactamente. Un ejemplo trascendental es el bioreactor. En su forma más simple, un bioreactor es simplemente un tanque conteniendo agua y células (por ejemplo, bacterias o levaduras) las cuales consumen nutrientes ("substrato") y fabrican productos (deseados y no deseados) y más células. Los bioreactores pueden ser totalmente complejos : las células son mecanismos autoregulables, y pueden ajustar sus relaciones de crecimiento y fabricación de diferentes productos dependiendo radicalmente de la temperatura y de las concentraciones de productos sobrantes (desechos), el alcohol, por ejemplo. Los sistemas con calentamiento o enfriamiento, reactores múltiples, u operación inestable, complican enormemente el análisis. Para

un criterio de desempeño, sin embargo, un sistema relativamente simple es mejor.

La versión más simple de un bioreactor es un reactor tanque de movimiento continuo de flujo (RTMCF), en el cual el crecimiento de las células depende únicamente de los nutrientes con que se alimenta al sistema. El valor esperado a ser controlado es la cantidad de células producidas. Frecuentemente se quiere producir la mayor cantidad posible de células. Un conjunto básico de ecuaciones para dicho bioreactor son :

$$\frac{dC_1}{dt} = - C_1 w + C_1 (1 - C_2) e^{C_2/\gamma}$$

$$\frac{dC_2}{dt} = - C_2 w + C_1 (1 - C_2) e^{C_2/\gamma} \frac{1 + \beta}{1 + \beta - C_2}$$

donde C_1 y C_2 son, respectivamente, la cantidad de células de menor dimensión y la conversión de substratos, con C_2 definido como $(S_F - S)/S_F$, donde S_F es la concentración del substrato (nutriente) en la alimentación del reactor y S es la concentración del substrato en el reactor (Ver Figura 5.19). El parámetro de control, w, es la razón de flujo a través del reactor. La primera ecuación describe que la razón de cambio en la cantidad de células es igual a la cantidad de células transportadas fuera del tanque ($C_1 w$) más el número en que han crecido las células ($C_1 (1 - C_2) e^{C_2/\gamma}$). La razón de crecimiento es proporcional a la cantidad actual de células (C_1), pero depende no linealmente de la concentración de nutrientes (C_2). La segunda ecuación describe que el cambio en la cantidad de nutrientes es igual a la razón en la cual el nutriente es llevado fuera del sistema más la razón en la cual éstos son metabolizados por las células. Las constantes β y γ determinan la relación del crecimiento de la célula y el consumo de nutrientes. El sistema de ecuaciones implica que el crecimiento más rápido ocurre cuando se tienen concentraciones intermedias de substratos y el más lento cuando se tienen concentraciones altas o bajas. Algunos investigadores definen otro parámetro Da, el cual es igual a $1/w$. Este no es un modelo completamente real de algún bioreactor, pero propor-

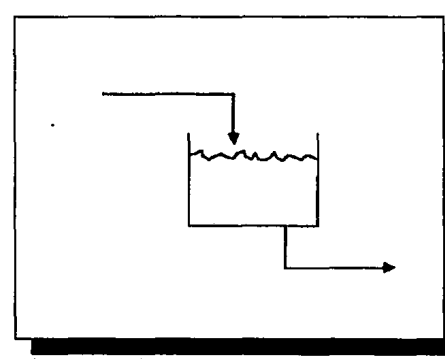


Fig 5.19 Diagrama del bioreactor.

ciona un sistema simple, aunque cambiante, que ha sido estudiado por expertos en control de procesos.

Este sistema es difícil de controlar por diversas razones: las ecuaciones no controladas son altamente no lineales y presentan ciclos límite. El comportamiento óptimo ocurre dentro o cerca de una región inestable. El problema exhibe multiplicidad: dos valores diferentes del parámetro de control (razón de flujo) pueden conducir al mismo punto de inicio (set point) deseado en la cantidad de células producidas.

5.2.9.3.1 CONTROL DEL BIOREACTOR UTILIZANDO UNA RED NEURAL ADAPTIVA.

La figura 5.20 describe una arquitectura de control basado en un modelo de neurocontrolador. Se utilizan dos redes neurales multinivel, una para el modelo de la planta y la segunda para el controlador. Cada red tiene 2 niveles ocultos de cinco neuronas cada uno; con mayor número de neuronas, con seguridad, se dará una mejor exactitud.

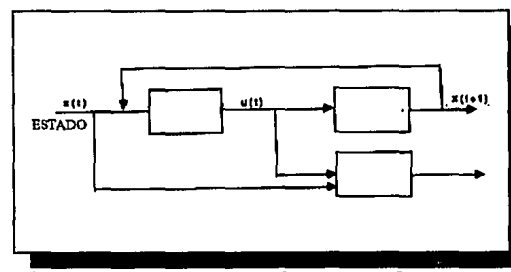


Fig 5.20 Arquitectura del control. Tanto el controlador como el modelo de la planta son redes adaptivas.

Se ha entrenado a la red adaptiva usando como entradas las concentraciones y las acciones de control, todas variando sobre el rengo completo de menor dimensión de cero a uno. Cuando el controlador se ha fijado, entrenado, y probado en los valores esperados de C_1=0.1130 y C_2=0.8902, la red producirá las concentraciones de salida de C_1=0.1224 y C_2=0.8778.

La discrepancia entre los valores esperados y los valores actuales se debe a la inexactitud en el modelo.

También se probo la red introduciéndole ruido aleatorio en la señal de control *w*. Los resultados mostraron que el control es estable y razonablemente exacto. Para conseguir el mejor control en la realidad, los sistemas ruidosos requerirán del uso de redes las cuales tendrán como entradas las salidas de las corridas previas y que intenten pronosticar los valores futuros.

Se ha sugerido como criterio de desempeño (benchmark) que este sistema se pruebe en tres problemas: (1) el control alrededor del punto w=0.75, que es un lazo abierto estable, (2) el control alrededor del punto w=1.25, que es inestable, y (3) el cambio descrito anteriormente iniciando con w=0.769, que corresponde a C_1=0.1236837 e incrementando la concentración esperada C_1 por 0.05, que cruza el límite de estabilidad.

5.2.9.3.2 VERSIONES MAS COMPLEJAS DEL CONTROL DEL BIOREACTOR.

Una serie de problemas de mayor dificultad se pueden construir basándose en un sistema similar al del bioreactor anteriormente descrito. Dicho sistema tiene una entrada y dos salidas. Es fácil crear un problema de dos entradas y dos salidas agregando un segundo flujo de alimentación con un substrato concentrado (S_C)S_C >> S_F y una razón de flujo w_C que debe de usarse para aumentar la concentración del substrato. También puede quererse una tercera alimentación (razón de flujo w_3) sin substrato para disminuir la concentración de substrato. Esto da como resultado el conjunto de ecuaciones:

$$\frac{dC_1}{dt} = - C_1(w + w_C + w_3) + C_1(1 - C_2)\ e^{C_2/\gamma}$$

$$\frac{dC_2}{dt} = - C_2(w+w_C+w_3) + c_C w_C + C_1(1-C_2)\ e^{C_2/\gamma}\ \frac{1 + \beta}{1 + \beta - C_2}$$

con c_C representando la forma de menor dimensión de S_C y w, w_C y w_3 como parámetros de control. Ahora se pueden poner independientes los valores esperados para C_1 y C_2, dando un problema de control multivariable.

Los problemas anteriores son los más simples de una enorme clase de problemas. Versiones más complejas del sistema se tienen frecuentemente y son fáciles de construir; sólo se necesitan considerar los productos fabricados por las células. Estos productos frecuentemente inhiben un mayor crecimiento (adicional), por lo que se incrementa la complejidad. También es deseable con frecuencia maximizar la cantidad de uno o más de los químicos producidos por las células. Otros modelos del comportamiento de la célula incluyen el comportamiento que varía fuer-

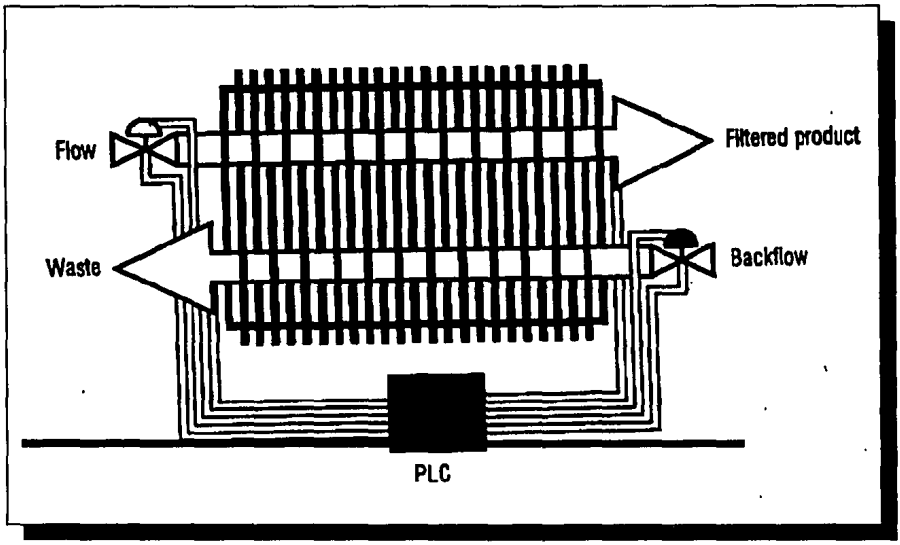


Fig 5.21 El PLC es mejor en el control de temperatura, procesos por pasos secuenciales y en pequeños procesos, como el de filtración que aquí se muestra.

temente con las direcciones de crecimiento. Antes que de moverse de un estado estable a otro en un reactor de flujo continuo, se pueden observar los problemas de una colonia (conjunto de microbios) en que el substrato es (periódicamente) agregado al tanque, pero no se tiene algún flujo de salida. En este caso lo que se quiere es maximizar la producción de grupos de células sobre el curso de una colonia, y el surgimiento de una optimización del problema (reforzamiento retardado).

Aunque no es posible esperar que todo controlador maneje este rango completo de problemas, es importante saber que están disponibles para pruebas aspectos diferentes de aprendizaje. Diferentes cambios de control, tales como la no linearidad, interacción de múltiples variables y la optimización sobre el tiempo, se pueden introducir a diferentes variaciones del problema.

5.2.10 SISTEMAS DE CONTROL DISTRIBUIDO BASADOS EN PLC's

Los administradores de las plantas pueden elevar la productividad con una mínima inversión, al interconectar los PLC's existentes a un sistema de control distribuído (SCD). El incremento de la competencia en el mercado mundial actual está forzando a las corporaciones a exprimir completamente la productividad en todo momento más allá de sus operaciones. Frecuentemente se debe lograr con una mínima inversión. Las fábricas y plantas que ya utilizan controladores programables pueden conectar sus PLC's a un SDC.

5.2.10.1 EL PLC.

Cuando por primera vez se introdujo en operaciones discretas, el PLC proporcionó un incremento dramático en la productividad de las fábricas, el rendimiento de los PLC's en el control de maquinaria fue muy superior del que se logró por relevadores alambrados secuencialmente. Posteriormente, el PLC se trasladó a la administración de pequeños procesos, tal como la filtración (Fig. 5.21) y de reacciones escalonadas secuencialmemte en bloque, gracias a sus inherentes ventajas en control de temperatura, y funciones matemáticas.

Los actuales PLC basados en microprocesadores pueden proporcionar una mayor funcionalidad que sus versiones anteriores, expandiendo enormemente sus aplicaciones potenciales, por ejemplo en un proceso químico el PLC puede ejecutar reacciones químicas secuencialmente. El alcance de este control podría abarcar niveles de temperatura y agitación. Además, el PLC hace uso de la nueva tecnología, tal como la inteligencia artificial (Redes Neurales), ofreciendo así funciones más sofisticadas, tales como procesos estadísticos (funciones de predicción con redes neurales), y control de calidad (implementado con una red neural con aprendizaje supervisado), las cuales se aplican en un amplio rango de aplicaciones a control.

Sin embargo, la mayoría de los PLC's actualmente funcionando en las plantas operan secuencialmente y dependen ampliamente de la lógica de escalón del relevador (diagramas de escalera) para su programación. En cambio, los PLC's más modernos se pueden programar a través de lenguajes gráficos que utilizan elementos (dispositivos) de lógica digital (Secuenciales y Combinatorios).

5.2.10.2 Los Sistemas de Control Distribuído (SCD).

Un SCD realiza múltiples tareas y programas, incluyendo instrucciones de salto y ejecución de diferentes subrutinas. Puede manejar instrucciones complejas y funcionar como un controlador contínuo o por lotes (Fig. 5.22).

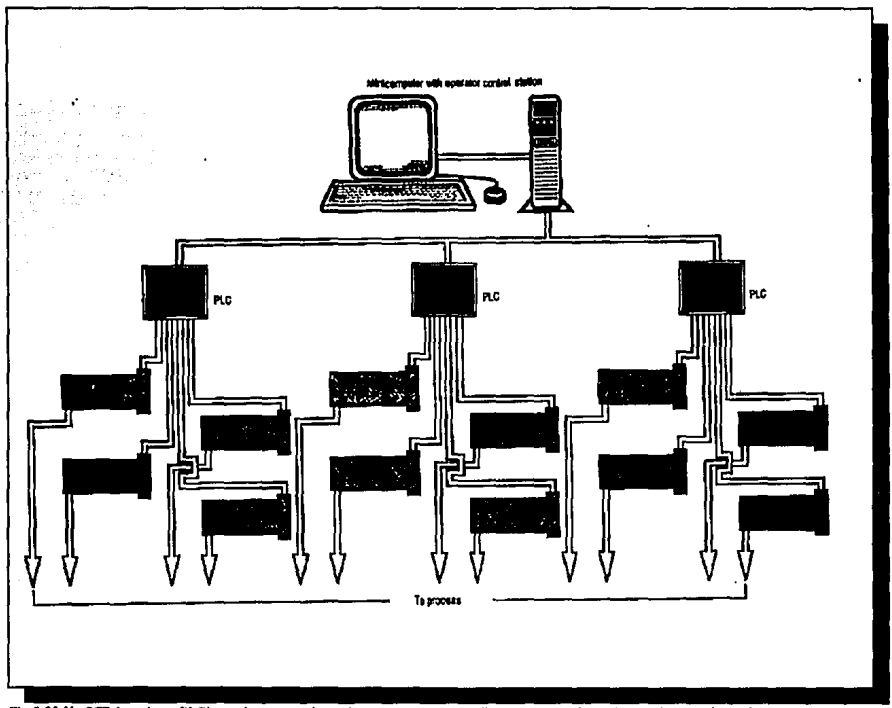


Fig.5.23 Un SCD basado en PLC's puede ser una alternativa no costosa para rediseñar y construir un sistema de control completamente nuevo.

Cuando inicialmente se implantaron, los SCD's eran grandes y difíciles de controlar, debido a que sus sistemas de cómputo eran enormes y difíciles de programar. Los programas se escribían en lenguaje máquina altamente reservado, y las E/S de muchos SCD's no funcionaban en las fábricas sin un medio ambiente adecuado (aire acondicionado y control de polvo). Además, tendían a ser muy caros a comparación de otros tipos de control existentes.

Desde entonces, los SCD's, al igual que sus sistemas de cómputo, han sido perfeccionados con el fin de facilitar su uso y bajar su costo. Una de sus características actuales es el uso de lenguajes de alto nivel, como el lenguaje C, y lenguajes de control propietarios y especializados. resultado. En consecuencia, los operadores pueden aplicar complicados esquemas de control tan fácil como llenar espacios en blanco.

5.2.10.3 COMBINANDO PLC's y SCD's.

Cada vez más directores de fábricas de procesos de manufactura discreta (como plantas automovilísticas) así como plantas con procesos contínuos y por lote (tal como la industria petrolera), están cambiando a una estratégia híbrida de control, SCD's basados en PLC's. Esto permite tomar ventaja del bajo costo de E/S de los PLC's y de la amplia flexibilidad de los SCD's.

Las estrategias híbridas están alcanzando un especial crecimiento en donde los PLC's ya eran usados extensivamente. permitiendo a la compañía reducir su inversión en equipo mientras se incrementa su productividad. Por ejemplo, la industria petrolera, y otras industrias, están comenzando a usar, de principio a fin, los PLC's existentes como E/S remotas reduciendo el número de cables desde los puntos de E/S hacia el SCD.

Para lograr la eficiencia máxima de una estrategia híbrida de control de un SCD basado en PLC's, los directores tienen que sacarle provecho al potencial de cada componente. Por ejemplo, si continúan usando únicamente la lógica de escalón del relevador para control de procesos, se perderá la funcionalidad en las instrucciones así como la habilidad de llamar a diferentes subrutinas y moverse a esquemas diferentes dentro del medio ambiente del control. Además, no tendrán la habilidad de manejar el flujo de materiales (ó productos) a través de un proceso. Así mismo, los SCD's basados en PLC's deberán usar algún lenguaje de alto nivel así como diagramas de escalera.

5.2.10.4 IMPLEMENTACION DE UN SCD BASADO EN PLC's.

Para demostrar como puede trabajar un SCD basado en PLC´s en una aplicación actual, considerese la implementación de una operación de manufatura quimica. Un proceso quimico está compuesto quimicamente de varias plantas quimicas: una planta puede fabricar estirena, otra benzeno, y una tercera cloro. Estás se

combinan para elaborar otros productos.

En este ejemplo, Los PLC´s manejan cada operación a nivel planta. El proceso completo (desde la selección de materiales hasta la facturación al cliente) es manejado por un SCD. En el desarrollo de sistemas hibridos, siempre se deberá tener en cuenta que:

- Los PLC´s no se comunican automaticamente con los SCD´s.

- La interconexión de los PLC´s sin una apropiada interfaz interna puede incrementar los riesgos al personal o propiciar accidentes.

Sin embargo, poniendo los SCD´s basados en PLC´s a varios niveles, las distintas estrategias se pueden combinar de manera que se integre el trabajo por medio de un cable de interconexión (Fig. 23).

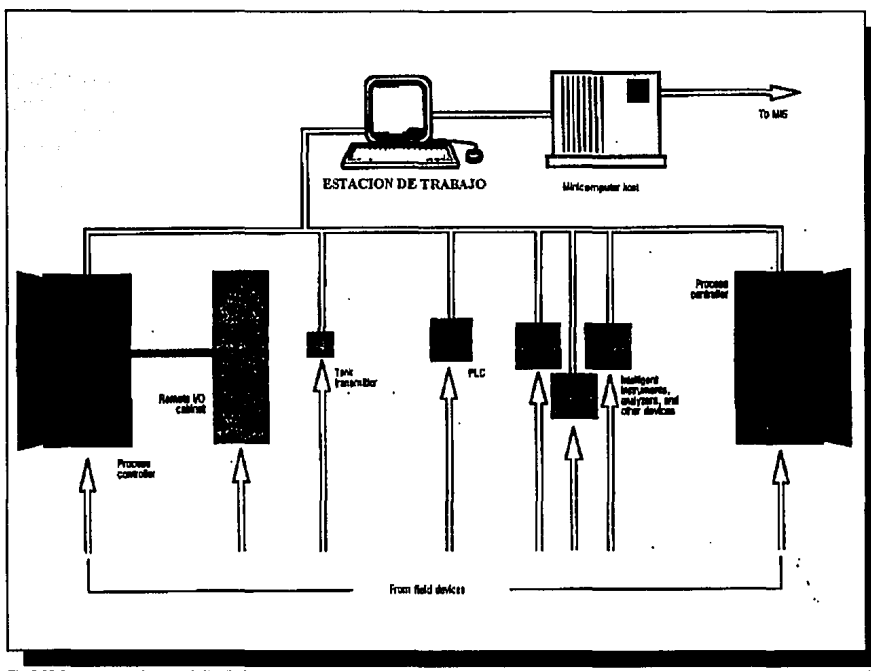


Fig 5.22 Los sistemas de control distribuído han sido buenos en tareas múltiples y procedimientos complejos que se encuentran en procesos industriales contínuos.

Se tienen dos posibles maneras de implementar un SCD basado en PLC´s:

1. El modelo *Arriba/Abajo* (top down) coloca un nivel completo de SCD sobre los PLC´s existentes (los cuales sirven como E/S remoto) para consegir las caracteristicas completas y la funcionalidad de un SCD clasico. Este modelo es factible sin importar que cantidad de puntos E/S estan involucrados en el sistema.

2. El modelo *Aterrizado* (ground up) enlaza los PLC´s a un "microSCD" (tipicamente basado en un microcomputador) para realizar reportes básicos y funciones de control. Este modelo es el mejor logrado para aplicaciones con menos de mil puntos E/S.

Como ya se indicó, un SCD clásico puede proporcionar facilidad de manejo desde la materia prima hasta la facturación al cliente. Para manipular esta tarea, un SCD requiere de cierta información de la fabrica, tales como pesos, porcentajes, temperaturas, biscocidad, es número de acidos en un polimero, características de temperatura y presión. Todo esto esta normalmente más alla del alcance de la mayoría de los PLC´s . En tal situación, un modelo *Arriba/Abajo* tiene sentido.

El modelo *aterrizado* es menos eficiente, pero también menos costoso. Una manera de consegir una fácil administración con este modelo es introducir técnicamente los datos requeridos en una computadora personal en la planta. Esta información podrá entonces ser accesada por un SCD remoto u otro sistema de cómputo. No hay duda de que el gasto inicial que se necesita para conectar los PLC´s a una microcomputadora es significativamente menor que el costo de instalación de un SCD clásico. En procesos con poco menos de mil puntos E/S, pudiera probablemente costar más instalar un SCD que introducir los datos manualmente a un microcomputador. Donde se involucran más de mil puntos E/S, la disminuida eficiencia de la entrada manual de datos puede resultar actualmente en un costo total más alto.

5.2.10.5 EL FUTURO: MAS RAPIDO, MAS ECONOMICO, MAS PEQUEÑO.

En general existe la tendencia a fabricar PLC´s y SCD´s más sofisticados, más fáciles de usar y significamente menos costosos. Esto es debido al dramático progreso en microminiaturización. Por ejemplo, se espera tener pronto, un sistema totalmente funcional que tradicionalmente costaría de USD $250,000 a USD $500,000, cueste de USD $30,000 a USD $50,000. La reducción más grande en costo será debido al hecho de que un sistema a la larga no dependerá necearíamente de una gran computadora en un cuarto de control exclusivo; Este podrá consistir de una red de computadoras personales distribuidas por toda la fabrica.

Gracias a la microminiatirización las E/S pueden colocarse en una caja de 8 x 10 x 12 pulgadas. Aceptados, en lazos criticos. tal densidad de E/S puede tener desventajas. Sin embargo una fácil solución a esto puede ser el de dedicar diferentes tableados a diferentes puntos E/S. Las E/S serán tan baratas que son posibles tableros redundantes. Ciertamente, muchos fabricantes están comenzando a ofrecer este tipo de productos E/S.
La computadora que maneje el sistema probablemente será del mismo tamaño de las E/S densas, y controlará una multitud de procesos.

Una tecnología futurista, la inteligencia artificial, ya está encontrando su camino en el control de sistemas en la forma de redes neurales. Esencialmente, las redes neurales aplican las reglas de control desarrolladas por los expertos para varias situaciones. Un sistema experto se puede implementar con las redes neurales. Una vez que se extienda el uso de dichos sistemas expertos, se espera que entregen un sobresaliente perfeccionamiento en la eficiencia. En aplicaciones de mantenimiento, por ejemplo, un sistema experto que pueda diagnosticar la causa de una falla en el equipo por medio de cuestionamientos a los operadores, ahorrará trabajo en el mantenimiento sin mencionar las horas de búsqueda en los manuales.

En el ejemplo de la figura 5.23, la red neural se aplica como controlador de todo el sistema, utilizando las entradas y salidas de los PLC´s como las entradas y salidas, respectivamente, de la red neural. Un simple cambio se necesita para acoplar la red neural al SCD, esto se muestra en la figura 5.24.

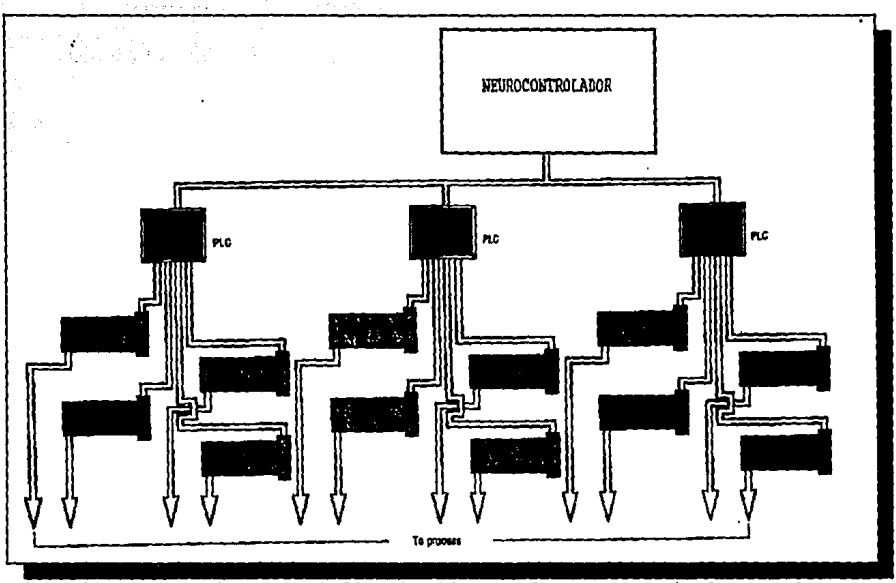


Fig.5.24 Sistema de control distribuído basado en PLC's implementado con un neurocontrolador.

5.2.11
OTRAS APLICACIONES

PREDICCIÓN DE EVENTOS FUTUROS.

Predecir en inteligencia artificial, es predecir eventos futuros basados en información histórica. Primero se escoge un conjunto de factores de entrada que al menos se piensa que serán útiles para predecir un conjunto de salidas. Este conjunto de factores son desarrollados típicamente con métodos estadísticos.

Veamos como se pueden resolver directamente, con menos tiempo y esfuerzo al formular un modelo para determinar los factores de entrada, con una red neural. Utilizando una red neural, podemos mirar la información histórica como un conjunto de patrones asociaciodos. Los patrones de entrada son los valores de todos los factores de entrada elegidos para un período particular. Los patrones de salida son las salidas registradas para un período especifico.

Una vez que la red neural está entrenada con el patrón de asociaciones de factores de entrada y salida para la información-histórica, esta puede "recordar" patrones de salida cuando se presenten determinados patrones de entrada, de este modo, la red neural entrenada puede predecir eventos futuros basados en nuevos conjuntos de factores de entrada. Esta es la primera aplicación que se le encuentra a las redes neuronales, el de obtener una predicción, pero no quiere decir que para esto fue creada.

RECONOCIMIENTO DE IMÁGENES

La segunda área de mayor aplicación de las redes neurales está orientada a el reconocimiento de imágenes. Tratamos a cada pixel como un patrón de una secuencia, ya sea de números binarios (0 ó 1, indicando presencia o ausencia de un pixel) o números reales (para imágenes de escala-de-grises o color). Entonces asociamos cada patrón con una salida, el cual es una descripción de lo que la 2imagen representa. Este podría ser una descripción de texto o puede ser simplemente un carácter, como en una aplicación de reconocimiento óptico de caracteres.

Un ejemplo de una aplicación del reconocimiento de imágenes, es un reconocedor de dígitos escritos a mano. La red neural debe de aprender las asociaciones entre los patrones de entrada (imágenes escritas a mano) y patrones de salida (los valores de los dígitos). Algo que se ha demostrado, es que la retropropagación es muy pobre para realizar esto. Por lo que es muy conveniente utilizar la red con un algoritmo de aprendizaje no supervisado (red con antepropagación).

Para el reconocimiento de imágenes escritas a mano. Cada imagen es leído como un archivo PCX. PCX es un archivo estándar para imágenes gráficas (introducidas por Zsoft en su programa de Paintbrush de PC). Es ciertamente el estándar más utilizado, a pesar de que no es tan amigable como otros formatos como TIFF. los códigos de entrenamiento para redes utilizando imágenes en PCX son extremadamente reusables.

Con el objetivo de obtener más simplicidad, cada imagen de dígito es almacenado en un archivo. Se rastrean las imágenes con un software específico para esto, se recorta el dígito, y se almacena en un archivo PCX. Esto es realizado con un software utilizando "signposts", es decir utiliza una amplitud de nuestra imagen para indicar la composición de nuestro dígito. Por ejemplo, se puede procesar una forma de impuesto donde los números están situados dentro de unas cajas cuadradas. Si se

busca la caja cuadrada con una mayor ampliación de nuestra imagen, podemos tratar los pixels dentro de la caja cuadrada al mismo tiempo que nuestro PCX archiva imágenes.

Para el reconocimiento de un dígito, procesamos el archivo PCX para obtener una ampliación de nuestra imagen al 16x16. Nos da un vector 256 neuronas de longitud. Pero la imagen en el archivo PCX es obviamente mayor que 16 X 16. En realidad, puede ser tan grande como 640 X 480 para imágenes en vga, o hasta 1,024 X 768 para imágenes Super VGA. Hasta para una imagen tan pequeños como 320 X 200, esto es aún un vector de 64,000 neuronas como largo para nuestro patrón de entrada.

Para obtener unos 16 X 16 pixeles de imagen, simplemente se amplían las imágenes en unos 64 X 64 pixeles. Unos 640 X 480 pixeles de imagen resultaría en cada componente de 40 x 30 pixeles. Para codificar cada componente en nuestros vectores de punto flotante, se determina el número de pixeles "activados" en la región y la división de estos por la cantidad de pixeles totales en la región. Así, si 400 pixels estuvieron en la región antes mencionada, el valor codificado tendría 0.333. Un alternativa para representar la imagen es utilizar vectores.

PROCESAMIENTO DE TEXTO

Se han realizado muchas exitosas aplicaciones de redes neurales con respecto al procesamiento de texto, incluyendo redes neuronales para recuperación de la información probabilística [IJCNN 90]. Un ejemplo específico de una aplicación recuperación es un simple corrector de ortografía. Esta no es una aplicación única de redes neurales. Jagota y Colgado de SUNY-BUFALO presentó un "léxico" (el cual en realidad fue justamente un corrector de ortografía) basado en la red de Hopfield [IJCNN 90].

La red de Hopfield no es muy diferente al modelo BAMs. Son redes aditivas discretas con una capa única de pesos (en contraste para la red de Hopfield analógica). También es probable que un enfoque neto de Hopfield para un corrector de ortografía sea casi tan rápida como un BAM. También, muchas redes neuronales basados en el reconocimiento de sistemas están siendo construidas, como parte de estos sistemas, un corrector de fonemas.

OPTIMIZACIÓN

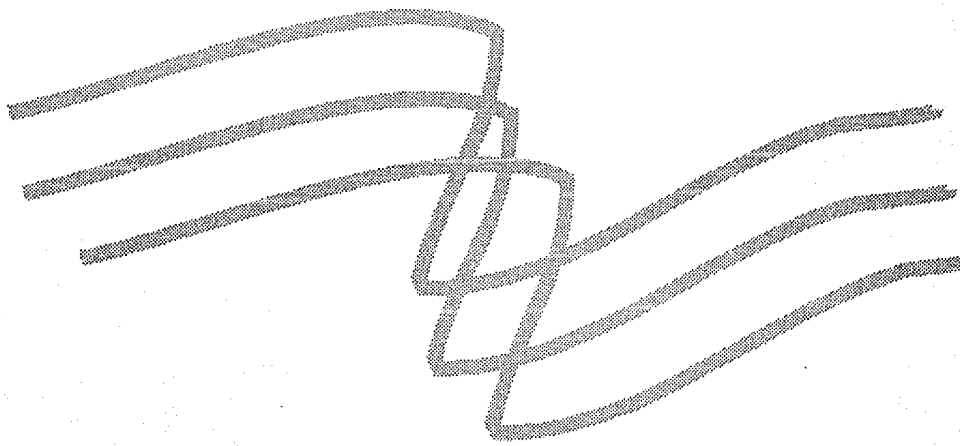
Las redes neurales se han formulado para resolver problemas difíciles. Cuando nos referimos a la optimizacion, nos referimos al proceso de encontrar la "mejor" solución entre muchas otras posibles alternativas. Generalmente, un problema que no tiene optimizacion se puede enumerar prácticamente todas las alternativas y aplicar una fórmula de evaluación para cada uno. Los problemas de optimización requieren típicamente algún forma de algoritmo para llegar a la mejor alternativa.

En la rama de las matemáticas y la ciencia, la más relacionada con problemas de optimizacion es la investigación operativa (OR). Los problemas OR son formulados típicamente como una fórmula de mínimos y máximos de un conjunto de restricciones.

El problema de minimizar algo frecuentemente se refiere en cuanto "cuesta la minimización". Las restricciones se representan como un conjunto de límites establecido de desigualdades en las variables (o un subconjunto de las variables) en el costo de la fórmula. Si estos límites no estuvieran allí, fuera una cuestión muy simple el costo de minimizar la función.

Los problemas con un numero limitado de restricciones y variables se pueden resolver utilizando un algoritmo matemático. Un ejemplo popular de tal algoritmo es el método símplex, que define los límites como un conjunto de n-espacios vectoriales (donde sea n es la cantidad de variables). Estos vectores definen una superficie de n-espacio contenidos en el conjunto de la solución. Si el valor de cada variable en la solución propuesta es dentro de los límites, entonces es una solución posible. Si cualquier dimensión o variable esta afuera de los límites, entonces es una solución no factible. Si el conjunto de vectores implicados por las restricciones sugieren un región encerradas en n-espacio, entonces el problema probablemente es "ilimitado".

APENDICE A



```
/*-----------------------------------------------------------------
*       file:       bprecog.c
*       desc:       recognizing program
*       by:         patrick ko
*       date:       20 aug 1991
*       revi:       v1.32u 26 apr 1992
*-----------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#ifdef __TURBOC__
#include <mem.h>
#include <alloc.h>
#endif
#include "nntype.h"
#include "nncreat.h"
#include "nntrain.h"
#include "nnerror.h"
#include "cparser.h"
#include "bprecogv.h"
#define MAXHIDDEN 128
static INTEGER      hiddencnt = 0;
static INTEGER      hidden[MAXHIDDEN];
static INTEGER      output;
static INTEGER      input;
static INTEGER      totalhidden;
static INTEGER      totalpatt = 0;
static VECTOR       **inputvect;
static VECTOR       **targtvect;
static char tfilename[128];
/*      default dump file name      */
static char dfilename[128] = "bptrain.dmp";
static char ofilename[128] = "bprecog.out";
int         usage( )
{
        printf( "%s %s - by %s\n", PROGNAME, VERSION, AUTHOR );
        printf( "(C)Copyright 1992 All Rights Reserved. %s\n\n", DATE );
        printf( "Description: backprop neural net recognition\n");
        printf( "Usage:\n%s @file -i=# -o=# -hh=# {-h=#} -samp=# -frecog=<fn> [-fdump=<fn>] -fout=<fn>\n\n", PROGNAME );
        printf( "Examples:\n" );
        printf( "Recognize 2 patterns in myinput.rgn with the NN created in bptrain example 1\n" );
        printf( "and generate a result file result.out:\n" );
        printf( "%s -i=2 -o=1 -hh=2 -h=3 -h=4 -samp=2 -frecog=myinput.rgn -fout=result.out\n", PROGNAME );
        printf( "\n" );
        printf( "Where\n\n" );
        printf( "-i=        dimension of input layer\n" );
        printf( "-o=        dimension of output layer\n" );
        printf( "-hh=       number of hidden layers\n" );
        printf( "-h=        each hidden layer dimension (may be multiple)\n" );
        printf( "-samp=     number of train input patterns in train file\n" );
        printf( "-frecog=   name of recog file containing inputs\n" );
        printf( "-fdump=    name of neural net file dumped by bptrain\n" );
        printf( "-fout=     name of recognition result file\n" );
        exit (1);
}
int parse( )
{       int         cmd;
        char        rest[128];
        int         resti;
        while ((cmd = cmdget( rest ))!= -1)
                {
                resti = atoi(rest);
```

```
                switch (cmd)
                        {
                        case CMD_DIMINPUT:
                                input = resti; break;
                        case CMD_DIMOUTPUT:
                                output = resti; break;
                        case CMD_DIMHIDDENY:
                                if (input <= 0 || output <= 0)
                                        {
                                        error( NNIOLAYER );
                                        }
                                if (resti > MAXHIDDEN)
                                        {
                                        error( NN2MANYLAYER );
                                        }
                                totalhidden = resti; break;
                        case CMD_DIMHIDDEN:
                                if (hiddencnt >= totalhidden)
                                        {
                                        /*
                                        * hidden layers more than specified
                                        */
                                        break;
                                        }
                                hidden[hiddencnt++] = resti;
                                break;
                        case CMD_RECOGFILE:
                                strcpy( tfilename, rest );
                                break;
                        case CMD_TOTALPATT:
                                totalpatt = resti;
                                break;
                        case CMD_DUMPFILE:
                                strcpy( dfilename, rest );
                                break;
                        case CMD_OUTFILE:
                                strcpy( ofilename, rest );
                                break;
                        case CMD_COMMENT:
                                break;
                        case CMD_NULL:
                                printf( "unknown command [%s]\n", rest );
                                exit (2);
                                break;
                        }
                }
                if (hiddencnt < totalhidden)
                        {
                        error( NN2MANYHIDDEN );
                        }
}
int getrecogvect( recogfile )
char    *recogfile;
{       int     i, j, cnt;
        VECTOR  *tmp;
        FILE    *ft;
        ft = fopen( recogfile, "r" );
        if (ft == NULL)
                {
                error( NNRFRERR );
                }
        inputvect = malloc( sizeof(VECTOR *) * totalpatt );
        for (i=0; i<totalpatt; i++)
                {
                /*      allocate input patterns                 */
                tmp = v_creat( input );
```

```
                for (j=0; j<input; j++)
                        {
                        cnt = fscanf( ft, "%lf", &tmp->vect[j] );
                        if (cnt < 1)
                                {
                                error( NNTFIERR );
                                }
                        }
                *(inputvect + i) = tmp;
                }
        fclose( ft );
}
int main( argc, argv )
int     argc;
char    **argv;
{
        INTEGER i;
        NET     *nn;
        FILE    *fdump, *fout;
        if (argc < 2)
                {
                usage();
                }
        else
                {
                cmdinit( argc, argv );
                parse();
                }
        /*      create a neural net     */
        nn = nn_creat( totalhidden + 1, input, output, hidden );
        printf( "opening dump file [%s] ...\n", dfilename );
        fdump = fopen( dfilename, "r" );
        nn_load( fdump, nn );
        fclose( fdump );
        printf( "start recognizing...\n" );
        getrecogvect( tfilename );
        fout = fopen( ofilename, "w" );
        if (fout == NULL)
                {
                error (NNOUTNOTOPEN);
                }
        for (i=0; i<totalpatt; i++)
                {
                nnbp_forward( nn, *(inputvect + i) );
                nn_dumpout( fout, nn );
                }
        fclose( fout );
}
```

```
/*---------------------------------------------------------------------
*       file:       bptrain.c
*       desc:       back propagation Multi Layer Perceptron (MLP) training
*       by:         patrick ko
*       date:       02 aug 1991
*       revi:       v1.32u 26 apr 1992
*---------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#ifdef __TURBOC__
#include <mem.h>
#include <alloc.h>
#endif
#include "nntype.h"
#include "nncreat.h"
#include "nntrain.h"
#include "nnerror.h"
#include "cparser.h"
#include "bptrainv.h"
#include "timer.h"
#define MAXHIDDEN 128
static INTEGER      hiddencnt = 0;
static INTEGER      hidden[MAXHIDDEN];
static INTEGER      output;
static INTEGER      input;
static INTEGER      totalhidden;
static INTEGER      totalpatt = 0;
static REAL         trainerr = ERROR_DEFAULT;
static INTEGER      report = 0;
static INTEGER      timer = 0;
static long int     tdump = 0;
static VECTOR       **inputvect;
static VECTOR       **targtvect;
extern REAL         TOLER;
static char  tname[128];
/*          dump file name with default    */
static char  dname[128] = "bptrain.dmp";
static char  dinname[128] = "";
int          usage( )
{
        printf( "%s %s - by %s\n", PROGNAME, VERSION, AUTHOR );
        printf( "Copyright (c) 1992 All Rights Reserved. %s\n\n", DATE );
        printf( "Description: backprop neural net training with adaptive coefficients\n");
        printf( "Usage: %s @file -i=# -o=# -hh=# {-h=#} -samp=# -ftrain=<fn>\n", PROGNAME);
        printf( "[-fdump=<fn>] [-fdumpin=<fn>] -r=# [-t] [-tdump=#] [-w+=# -w-=#]\n" );
        printf( "[-err=] [-torerr=] [// ...]\n");
        printf( "Example: " );
        printf( "create and train a 2x4x3x1 dimension NN with 10 samples\n");
        printf( "%s -i=2 -o=1 -hh=2 -h=4 -h=3 -err=0.01 ", PROGNAME );
        printf( "-ftrain=input.trn -samp=10\n" );
        printf( "Where:\n" );
        printf( "-i=,-o=     dimension of input/output layer\n" );
        printf( "-hh=        number of hidden layers\n" );
        printf( "-h=         each hidden layer dimension (may be multiple)\n" );
        printf( "-ftrain=    name of train file containing inputs and targets\n" );
        printf( "-fdump=     name of output weights dump file\n" );
        printf( "-fdumpin=   name of input weights dump file (if any)\n");
        printf( "-samp=      number of train input patterns in train file\n" );
```

```
        printf( "-r=        report training status interval\n" );
        printf( "-t         time the training (good for non-Unix)\n" );
        printf( "-tdump=    time for periodic dump (specify seconds)\n");
        printf( "-w+=       initial random weight upper bound\n" );
        printf( "-w-=       initial random weight lower bound\n" );
        printf( "-err=      mean square per unit train error ");
        printf( "(def=%f)\n", ERROR_DEFAULT );
        printf( "-torerr=   tolerance error (def=%f)\n", TOLER_DEFAULT);
        exit (0);
}
int     parse( )
{
        int         cmd;
        char        rest[128];
        int         resti;
        long        restl;
        while ((cmd = cmdget( rest ))!= -1)
                {
                resti = atoi(rest);
                restl = atol(rest);
                switch (cmd)
                        {
                        case CMD_DIMINPUT:
                                input = resti; break;
                        case CMD_DIMOUTPUT:
                                output = resti; break;
                        case CMD_DIMHIDDENY:
                                if (input <= 0 || output <= 0)
                                        {
                                        error( NNIOLAYER );
                                        }
                                if (resti > MAXHIDDEN)
                                        {
                                        error( NN2MANYLAYER );
                                        }
                                totalhidden = resti; break;
                        case CMD_DIMHIDDEN:
                                if (hiddencnt >= totalhidden)
                                        {
                                        /* hidden layers more than specified */
                                        break;
                                        }
                                hidden[hiddencnt++] = resti;
                                break;
                        case CMD_TRAINFILE:
                                strcpy( tname, rest );
                                break;
                        case CMD_TOTALPATT:
                                totalpatt = resti;
                                break;
                        case CMD_DUMPFILE:
                                strcpy( dname, rest );
                                break;
                        case CMD_DUMPIN:
                                strcpy( dinname, rest );
                                break;
                        case CMD_TRAINERR:
                                trainerr = atof( rest );
                                break;
                        case CMD_REPORT:
                                report = resti;
                                break;
                        case CMD_TIMER:
                                timer = 1;
                                break;
                        case CMD_TDUMP:
```

```
                                        tdump = rest1;
                                        break;
                                case CMD_WPOS:
                                        UB = atof(rest);
                                        break;
                                case CMD_WNEG:
                                        LB = atof(rest);
                                        break;
                                case CMD_TOLER:
                                        TOLER = atof(rest);
                                        break;
                                case CMD_COMMENT:
                                        break;
                                case CMD_NULL:
                                        printf( "%s: unknown command [%s]\n", PROGNAME, rest );
                                        exit (2);
                                        break;
                                }
                        }
                        if (hiddencnt < totalhidden)
                                {
                                error( NN2MANYHIDDEN );
                                }
}
int     gettrainvect( tname )
char    *tname;
{
        int     i, j, cnt;
        VECTOR  *tmp;
        FILE    *ft;
        ft = fopen( tname, "r" );
        if (ft == NULL)
                {
                error( NNTFRERR );
                }
        inputvect = malloc( sizeof(VECTOR *) * totalpatt );
        targtvect = malloc( sizeof(VECTOR *) * totalpatt );
        if (totalpatt <= 0)
                {
                error( NN2FEWPATT );
                }
        for (i=0; i<totalpatt; i++)
                {
                /*      allocate input patterns                 */
                tmp = v_creat( input );
                for (j=0; j<input; j++)
                        {
                        cnt = fscanf( ft, "%lf", &tmp->vect[j] );
                        if (cnt < 1)
                                {
                                error( NNTFIERR );
                                }
                        }
                *(inputvect + i) = tmp;

                tmp = v_creat( output );
                for (j=0; j<output; j++)
                        {
                        cnt = fscanf( ft, "%lf", &tmp->vect[j] );
                        if (cnt < 1)
                                {
                                error( NNTFIERR );
                                }
                        }
                *(targtvect + i) = tmp;
                }
```

```
        fclose( ft );
}
int     main( argc, argv )
int     argc;
char    ** argv;
{
        NET     *nn;
        FILE    *fdump;
        if (argc < 2)
                {
                usage();
                }
        else
                {
                cmdinit( argc, argv );
                parse();
                }
        /* create a neural net */
        nn = nn_creat( totalhidden + 1, input, output, hidden );
        gettrainvect( tname );
        /* read last dump, if any */
        if (*dinname != NULL)
                {
                printf( "%s: opening dump file [%s] ...\n", PROGNAME, dinname);
                if ((fdump = fopen( dinname, "r" )) != NULL)
                        {
                        nn_load( fdump, nn );
                        fclose( fdump );
                        }
                }
        printf( "%s: start\n", PROGNAME );
        if (timer)
                timer_restart();
        /*   the default training error, ..., etc can be incorporated into
        the interface - if you like.            */
        nnbp_train( nn, inputvect, targtvect, totalpatt, trainerr, ETA_DEFAULT, ALPHA_DEFAULT, report, tdump, dname );
        if (timer)
                printf("%s: time elapsed = %ld secs\n", PROGNAME, timer_stop());

        printf( "%s: dump neural net to [%s]\n", PROGNAME, dname );
        fdump = fopen( dname, "w" );
        nn_dump( fdump, nn );
        fclose(fdump);
}
```

```
/*-----------------------------------------------------------------------
*       file:       nncreat.c
*       desc:       create a fully connected neural net
*       by:         patrick ko
*       date:       v1.1u - 02 aug 91
*       revi:       v1.2b - 15 jan 92, adaptive coefficients (beta)
*                   v1.3u - 17 jan 92, revised data structures
*-----------------------------------------------------------------------*/
```

```
#include    <stdio.h>
#ifdef          __TURBOC__
#include    <alloc.h>
#endif
```

```
#include    <stdarg.h>
#include    "nntype.h"
#include    "nnerror.h"
#include    "nncreat.h"
#include    "random.h"
REAL    UB=1.5;
REAL    LB=0.5;
/*=============================================================================
*       funct:      nn_creat
*       dscpt:      creat an nn object
*       given:      totallayer = total number of hidden and output layers
*                   diminput = dimension of inputlayer
*                   dimother = dimension of other layers
*       retrn:      allocated NET
*============================================================================*/
NET * nn_creat( totallayer, diminput, dimoutput, dimother )
INTEGER             totallayer;
INTEGER             diminput;
INTEGER             dimoutput;
INTEGER             * dimother;
{
        INTEGER i;
        INTEGER dimwt;
        NET     *nntmp;
        INTEGER dimlayer;
        /* malloc the NET struct */
        if ((nntmp= (NET *)malloc(sizeof(NET))) == NULL)
                {
                error( NNMALLOC );
                }
        else
                {
                /* malloc the LAYER *'s          */
                DimNet(nntmp) = totallayer;
                if ((nntmp->layer = (LAYER **)malloc(sizeof(LAYER *) * DimNet(nntmp))) == NULL)
                        {
                        error( NNMALLOC );
                        }
                else
                        {
                        /* dimension of first layer's wgt vector
                        * is equal to dimension of input vector                 */
                        dimwt = diminput;
                        for (i=0; i<DimNet(nntmp); i++)
                                {
                                if (i == DimNet(nntmp)-1)
                                        {
                                        dimlayer = dimoutput;
                                        }
                                else
                                        {
                                        dimlayer = *(dimother + i);
                                        }
                                Layer(nntmp,i) = l_creat( dimlayer, dimwt );
                                /*  dimension of this layer's wgt vector is equal
                                * to dimension of previous layer's out vector    */
                                dimwt = dimlayer;
                                }
                        return (nntmp);
                        }
                }
}
```

```
/*=========================================================================
*       funct:      v_creat
*       dscpt:      create an allocated VECTOR object
*       given:      dim = dimension of the vector
*       retrn:      allocated VECTOR
*=========================================================================*/
VECTOR * v_creat( dim )
INTEGER  dim;
{
        VECTOR  *vtmp;
        INTEGER  i;
        if ((vtmp = (VECTOR *)malloc(sizeof(VECTOR))) == NULL)
                {
                error( NNMALLOC );
                }
        else
                {
                DimVect(vtmp) = dim;
                if ((Vect(vtmp) = (REAL *)malloc(sizeof(REAL) * dim)) == NULL)
                        {
                        error( NNMALLOC );
                        }
                else
                        {
                        v_fill(vtmp, 0.0);
                        return (vtmp);
                        }
                }
}
/*=========================================================================
*       funct:      v_fill
*       dscpt:      fill all dim of a vector with a value
*       given:      v = vector, m = value
*       retrn:      v
*=========================================================================*/
VECTOR * v_fill( v, m )
VECTOR  *v;
REAL     m;
{
        int      i;
        for (i=0; i<DimVect(v); i++)
                {
                Vi(v,i) = m;
                }
        return (v);
}
/*=========================================================================
*       funct:      v_rand
*       dscpt:      fill a vector with random value (default 0.5 - 1.5)
*       given:      v = vector
*       retrn:      v
*=========================================================================*/
VECTOR *v_rand( v )
VECTOR  *v;
{
        INTEGER  i;
        for (i=0; i<DimVect(v); i++)
                {
                Vi(v,i) = rnd() * (UB-LB) + LB;
                }
        return (v);
}
```

```
/*=========================================================================
*       funct:      u_creat
*       dscpt:      create an allocated UNIT
*       given:      dimwgtvect = dimension of the unit's weight vector
*       retrn:      allocated UNIT
*=========================================================================*/
UNIT * u_creat( dimwgtvect )
INTEGER  dimwgtvect;
{
        UNIT    *utmp;
        if ((utmp=(UNIT *)malloc(sizeof(UNIT))) == NULL)
                {
                error(NNMALLOC);
                }
        else
                {
                vWeight(utmp) = v_creat( dimwgtvect );
                v_rand( vWeight(utmp) );
                vdWeight1(utmp) = v_creat( dimwgtvect );
                vdWeight2(utmp) = v_creat( dimwgtvect );
                vDO(utmp) = v_creat( dimwgtvect );
                Out(utmp) = 0.0;
                Net(utmp) = 0.0;
                Dlt(utmp) = 0.0;
                nDlt(utmp)= 0.0;
                Bias(utmp) = rnd0 / 5000.0 ;
                dBias1(utmp) = 0.0;
                dBias2(utmp) = 0.0;
                /* allocation successful */
                return (utmp);
                }
}
/*=========================================================================
*       funct:      l_creat
*       dscpt:      create an allocated LAYER object
*       given:      dimlayer = number of units in this layer
*                   dimwgtvect = dimension of the weight vector of each unit
*       retrn:      allocated LAYER
*=========================================================================*/
LAYER * l_creat( dimlayer, dimwgtvect )
INTEGER  dimlayer;
INTEGER  dimwgtvect;
{
        LAYER   *ltmp;
        INTEGER  i;
        if ((ltmp= (LAYER *)malloc(sizeof(LAYER))) == NULL)
                {
                error(NNMALLOC);
                }
        else
                {
                DimLayer(ltmp) = dimlayer;
                if ((ltmp-> unit= (UNIT **)malloc(sizeof(UNIT *) * dimlayer)) == NULL)
                        {
                        error(NNMALLOC);
                        }
                else
                        {
                        for (i=0; i<dimlayer; i++)
                                {
                                Unit(ltmp,i) = u_creat( dimwgtvect );
                                }
                        /* allocation successful */
                        return (ltmp);
                        }
                }}
```

```
/*-----------------------------------------------------------------------
*       file:       nndump.c
*       desc:       dump structures in nntype.h
*       by:         patrick ko
*       date:       13 aug 1991
*       revi:       v1.2b - 15 jan 1992, coefficient adaptation
*                   v1.3u - 18 jan 1992, revised data structures
*-----------------------------------------------------------------------*/

#include <stdio.h>
#include "nntype.h"
void        v_dump( fp, vp )
FILE        *fp;
VECTOR      *vp;
{
            INTEGER  i;
            for (i=0; i<DimVect(vp); i++)
                        {
                        fprintf( fp, "%f ", vp->vect[i] );
                        }
            fprintf( fp, "\n" );
}
void        v_load( fp, vp )
FILE        *fp;
VECTOR      *vp;
{
            INTEGER  i;
            for (i=0; i<DimVect(vp); i++)
                        {
                        fscanf( fp, "%lf ", &vp->vect[i] );
                        }
}
void        u_dumpweight( fp, unit )
FILE        *fp;
UNIT        *unit;
{
            v_dump( fp, vWeight(unit) );
            fprintf(fp, "%f\n", Bias(unit));
}
void        u_loadweight( fp, unit )
FILE        *fp;
UNIT        *unit;
{
            v_load( fp, vWeight(unit) );
            fscanf(fp, "%lf\n", &Bias(unit));
}
void        l_dump( fp, ly )
FILE        *fp;
LAYER       *ly;
{
            INTEGER  i;
            for (i=0; i<DimLayer(ly); i++)
                        {
                        u_dumpweight( fp, Unit(ly,i) );
                        }
}
void        l_load( fp, ly )
FILE        *fp;
LAYER       *ly;
{
            INTEGER  i;
```

```
        for (i=0; i<DimLayer(ly); i++)
                {
                u_loadweight( fp, Unit(ly,i) );
                }
}
void    nn_dump( fp, nn )
FILE    *fp;
NET     *nn;
{
        INTEGER  i;
        for (i=0; i<DimNet(nn); i++)
                {
                l_dump( fp, Layer(nn,i) );
                }
}
void    nn_load( fp, nn )
FILE    *fp;
NET     *nn;
{
        INTEGER  i;
        for (i=0; i<DimNet(nn); i++)
                {
                l_load( fp, Layer(nn,i) );
                }
}
void    nn_dumpout( fp, nn )
FILE    *fp;
NET     *nn;
{
        INTEGER  i, j;
        LAYER    *I;
        UNIT     *J;
        I = Layer(nn,DimNet(nn)-1);
        for (j=0; j<DimLayer(I); j++)
                {
                J = Unit(I,j);
                fprintf( fp, "%f ", Out(J) );
                }
        fprintf( fp, "\n" );
}
```

```
/*-----------------------------------------------------------------
*       file:       nnerror.c
*       desc:       define all errors
*       by:         patrick ko
*       date:       2 aug 91
*-----------------------------------------------------------------*/
```

```
#include "nnerror.h"
struct error {
        int         error;
        char        *errmsg;
        };
static struct error     errtbl[] =
{
        { NNMALLOC,     "malloc error" },
        { NNTFRERR,     "train file reading error" },
        { NNRFRERR,     "recognition file reading error" },
```

```
        { NNTFIERR,         "train file input error" },
        { NNIOLAYER,        "input/output layer must be specified first" },
        { NN2MANYLAYER,     "hidden layer more than specified" },
        { NN2FEWPATT,       "no training pattern" },
        { NN2MANYHIDDEN,    "too many hidden layers specfied" },
        { NNOUTNOTOPEN,     "output file cannot be opened" }
};
int error( errno )
int     errno;
{
        printf( "nnerror %d: %s\n", errno, errtbl[errno].errmsg );
        exit (errno);
}
```

```
/*-----------------------------------------------------------------
*       file:       nntrain.c
*       desc:       train a fully connected neural net by backpropagation
*       by:         patrick ko
*       date:       2 aug 91
*       revi:       v1.2b - 15 jan 92, adaptive coefficients
*                   v1.3u - 18 jan 92, revised data structures
*                   v1.31u - 20 jan 92, periodic dump, weights retrieval
*-----------------------------------------------------------------*/
```

```
#include <stdio.h>
#include <math.h>
#include <values.h>
#include <time.h>
#include "nntype.h"
#include "nnmath.h"
#include "nntrain.h"
#include "nndump.h"
#define     global
#define     LAMBDA0         0.1
static REAL         ETA = ETA_DEFAULT;
static REAL         MOMENTUM = ALPHA_DEFAULT;
static REAL         LAMBDA = LAMBDA_DEFAULT;
static INTEGER      REPINTERVAL = 0;
global REAL         TOLER = TOLER_DEFAULT;
/*================================================================================
*       funct:      nnbp_train
*       dscpt:      train a neural net using backpropagation
*       given:      net = the neural net
*                   inpvect = 1 input vector ( 1 train pattern )
*                   tarvect = 1 target vector ( 1 target pattern )
*                   np = number of patterns
*                   err = error
*                   eta, momentum
*                   report = dump info interval (no.of train cycles), 0=not dump
*                   tdump = no of seconds for periodic dump, 0=not dump
*                   dfilename = period dump file name
*       retm:       measure of error
*================================================================================*/
REAL nnbp_train( net, inpvect, tarvect, np, err, eta, momentum, report, tdump, dfilename )
NET     *net;
VECTOR  **inpvect;
```

```
VECTOR  **tarvect;
REAL    err, eta, momentum;
INTEGER np, report;
long int tdump;
char    *dfilename;
{
        REAL    Error;
        INTEGER cnt;
        time_t  lasttime, thistime;
        FILE    *fdump;
        cnt = 0;
        REPINTERVAL = report;
        ETA = eta;
        MOMENTUM = momentum;
        Error = MAXFLOAT;
        if (tdump)
                time(&lasttime);
        while (Error > err)
                {
                cnt++;
                Error = nnbp_train1( net, inpvect, tarvect, np, Error );
                if (report)
                        nnbp_report(cnt, Error);
                if (tdump)
                        {
                        if (((thistime = time(&thistime)) - lasttime) >= tdump)
                                {
                                fdump = fopen( dfilename, "w" );
                                nn_dump( fdump, net );
                                fclose(fdump);
                                lasttime = thistime;
                                }
                        }
                }
        return (Error);
}
/*=============================================================================
*       funct:  nnbp_report
*       dscpt:  print report lines to terminal
*       given:  cnt = number of train cycles
*               error = overall energy
*       retrn:  nothing
*============================================================================*/
void nnbp_report( cnt, error )
INTEGER cnt;
REAL    error;
{
        if (!(cnt%REPINTERVAL))
                {
                printf("nntrain: cycle %d, mean square error per unit = %f\n", cnt, error );
                }
}
/*=============================================================================
*       funct:  nnbp_train1
*       dscpt:  train a neural net 1 cycle using backpropagation
*       given:  net = the neural net
*               inpvect = 1 set of input vectors
*               tarvect = 1 set of target vectors
*               np = number of patterns
*               LastError = energy at last cycle
*       retrn:  measure of error after this train cycle
*============================================================================*/
REAL nnbp_train1( net, inpvect, tarvect, np, LastError )
NET     *net;
VECTOR  **inpvect;
VECTOR  **tarvect;
```

```
INTEGER np;
REAL    LastError;
{
        REAL    Error;
        INTEGER i;
        INTEGER fire=0;
        Error = 0.0;
        nnbp_init( net );
        for (i=0; i<np; i++)
                {
                nnbp_forward( net, *(inpvect + i));
                Error += nnbp_backward( net, *(inpvect + i), *(tarvect + i));
                }
        Error = Error / np / DimNetOut(net);
        /*  coefficients adaptation and dWeight calculations            */
        if (Error <= LastError + TOLER )
                {
                /* weights will be updated, go ahead */
                fire = 1;
                nnbp_coeffadapt( net );
                nnbp_dweightcalc( net, np, fire );
                return (Error);
                }
        else
                {
                /* weights will not be updated, backtrack */
                fire = 0;
                ETA *= BACKTRACK_STEP;                  /* half the ETA */
                ETA = ground(ETA,ETA_FLOOR);
                MOMENTUM = ETA * LAMBDA;
                nnbp_dweightcalc( net, np, fire );
                return (LastError);
                }
}
/*=============================================================================
*       funct:      nnbp_forward (pass)
*       dscpt:      forward pass calculation
*       given:      net = the neural net
*                   inpvect = 1 input vector ( 1 train pattern )
*       retrn:      nothing
*       cmmnt:      net's output Out(J) calculated at every unit
*============================================================================*/
void nnbp_forward( net, inpvect )
NET     *net;
VECTOR  *inpvect;
{
        LAYER   *I, *input;
        UNIT    *J;
        INTEGER i, j, k;
        REAL    sum, out;
        /* phase 1 - forward compute output value Out's */
        input = NULL;
        /* For each layer I in the network */
        for (i=0; i<DimNet(net); i++)
                {
                I = Layer(net,i);
                /* For each unit J in the layer */
                for (j=0; j<DimLayer(I); j++)
                        {
                        J = Unit(I,j);
                        Net(J) = Bias(J) + dBias1(J); /* add bias */
                        for (k=0; k<DimvWeight(J); k++)
                                {
                                if (i==0)
                                        out = Vi(inpvect,k);
                                else
```

```
                                        out = Out(Unit(input,k));
                                    Net(J) += (Weight(J,k) + dWeight1(J,k)) * out;
                                    }
                                Out(J) = sigmoid(Net(J));
                                }
                        input = I;
                        }
}
void nnbp_init( net )
NET     *net;
{
        LAYER   *I;
        UNIT    *J;
        INTEGER i, j, k;
        i = DimNet(net);
        while (i--)
                {
                I = Layer(net,i);
                for (j=0; j<DimLayer(I); j++)
                        {
                        J = Unit(I,j);
                        nDlt(J) = 0.0;
                        for (k=0; k<DimvWeight(J); k++)
                                {
                                DO(J,k) = 0.0;
                                }
                        }
                }
}
/*=============================================================================
*       funct:  nnbp_backward
*       dscpt:  backward pass calculation
*       given:  net = the neural net
*               inpvect = I input vector ( I train pattern )
*               tarvect = I target vector
*       retrn:  Ep * 2
*       cmmnt:  net's weight and bias adjusted at every layer
*============================================================================*/
REAL nnbp_backward( net, inpvect, tarvect )
NET     *net;
VECTOR  *inpvect;
VECTOR  *tarvect;
{
        LAYER   *I, *F, *B;
        UNIT    *J, *JF;
        INTEGER i, j, k;
        REAL    sum, out;
        REAL    Ep, diff;
        Ep = 0.0;
        /*      phase 2 - target comparison and back propagation        */
        i = DimNet(net) - 1;
        F = I = Layer(net,i);
        B = Layer(net,i - 1);
        /*
        *       Delta rule 1 - OUTPUT LAYER
        *       dpj = (tpj - opj) * f'(netpj)
        */
        for (j=0; j<DimLayer(I); j++)
                {
                J = Unit(I,j);
                diff = V(tarvect,j) - Out(J);
                Dlt(J) = diff * Out(J) * (1.0 - Out(J));
                nDlt(J) += Dlt(J);      /* accumulate Dpj's */
                for (k=0; k<DimvWeight(J); k++)
                        {
                        if (i==0)
```

```
                                        out = Vi(inpvect,k);
                                else
                                        out = Out(Unit(B,k));
                                DO(J,k) += Dlt(J) * out;
                                }
                        Ep += diff * diff;
                        }
                --i;
                while (i >= 0)
                        {
                        I = Layer(net,i);                       /* current layer */
                        B = Layer(net,i - 1);
                        /*
                        *       delta rule 2 - HIDDEN LAYER:
                        *       dpj = f'(netpj) * SUMMATEk( Dpk * Wkj )
                        */
                        for (j=0; j<DimLayer(I); j++)
                                {
                                J = Unit(I,j);
                                sum = 0.0;
                                for (k=0; k<DimLayer(F); k++)
                                        {
                                        JF = Unit(F,k);
                                        sum += Dlt(JF) * (Weight(JF,j)+dWeight1(JF,j));
                                        }
                                Dlt(J) = Out(J) * (1.0 - Out(J)) * sum;
                                nDlt(J) += Dlt(J);
                                for (k=0; k<DimvWeight(J); k++)
                                        {
                                        if (i==0)
                                                out = Vi(inpvect,k);
                                        else
                                                out = Out(Unit(B,k));
                                        DO(J,k) += Dlt(J) * out;
                                        }
                                }
                        F = I;
                        i--;
                        }
                return (Ep);
}
void nnbp_coeffadapt( net )
NET     *net;
{
        LAYER   *I, *B;
        UNIT    *J;
        INTEGER n, i, j, k;
        REAL    EW, ME, MW, costh;
        EW = ME = MW = 0.0;
        i = DimNet(net);
        while (i--)
                {
                I = Layer(net,i);
                for (j=0; j<DimLayer(I); j++)
                        {
                        J = Unit(I,j);
                        for (k=0; k<DimvWeight(J); k++)
                                {
                                ME += DO(J,k) * DO(J,k);
                                MW += dWeight1(J,k) * dWeight1(J,k);
                                EW += DO(J,k) * dWeight1(J,k);
                                }
                        }
                }
        ME = sqrt(ME);                  /* modulus of cost funct vector B */
        MW = sqrt(MW);                  /* modulus of delta weight vector dWn-1*/
```

```
        ME = ground(ME,ME_FLOOR);               /* constraints */
        MW = ground(MW,MW_FLOOR);                       /* constraints */
        costh = EW / (ME * MW);
        /* coefficients adaptation !!! */
        ETA = ETA * (1.0 + 0.5 * costh);
        ETA = ground(ETA,ETA_FLOOR);
        LAMBDA = LAMBDA0 * ME / MW;
        MOMENTUM = ETA * LAMBDA;
}
void nnbp_dweightcalc( net, np, fire )
NET     *net;
INTEGER np;
INTEGER fire;
{
        LAYER   *I;
        UNIT    *J;
        INTEGER n, i, j, k;
        i = DimNet(net);
        /* calculate dWeights for every unit */
        while (i--)
                {
                I = Layer(net,i);
                for (j=0; j<DimLayer(I); j++)
                        {
                        J = Unit(I,j);
                        nDlt(J) /= np;
                        for (k=0; k<DimvWeight(J); k++)
                                {
                                DO(J,k) /= np;
                                if (fire)
                                        {
                                        /* commit weight change */
                                        Weight(J,k) += dWeight1(J,k);
                                        /* dW n-2 = dW n-1 */
                                        dWeight2(J,k) = dWeight1(J,k);
                                        }
                                dWeight1(J,k) = ETA * DO(J,k) + MOMENTUM * dWeight2(J,k);
                                }
                        if (fire)
                                {
                                Bias(J) += dBias1(J);
                                dBias2(J) = dBias1(J);
                                }
                        dBias1(J) = ETA * nDlt(J) + MOMENTUM * dBias2(J);
                        }
                }
}
```

```
/*=============================================================================
*       BPRECOGU.H
*============================================================================*/
```

```
#define     VERSION             "1.32u"
#define     PROGNAME            "bprecog"
#define     AUTHOR              "Patrick KO Shu Pui"
#define     DATE                "26th April, 1992."
```

```
/*=============================================================================
*       BPTRAINV.H
*============================================================================*/
```

```
#define     VERSION             "1.32u"
#define     PROGNAME            "bptrain"
#define     AUTHOR              "Patrick KO Shu Pui"
#define     DATE                "26th April, 1992."
```

```
/*-----------------------------------------------------------------------
**      file:       nncreat.h
**      desc:       nncreate.c header file
**      by:                 patrick ko
**      date:       2 aug 91
**-----------------------------------------------------------------------*/
```

```
#include    "nntype.h"
#ifdef __TURBOC__
NET *               nn_creat    (INTEGER, INTEGER, INTEGER, INTEGER *);
VECTOR *            v_creat             (INTEGER);
UNIT *              u_creat             (INTEGER);
LAYER *             l_creat             (INTEGER, INTEGER);
VECTOR *            v_rand              (VECTOR *);
VECTOR *            v_fill      (/*VECTOR *, REAL*/);
#else
NET *               nn_creat    0;
VECTOR *            v_creat             0;
UNIT *              u_creat             0;
LAYER *             l_creat             0;
VECTOR *            v_rand              0;
VECTOR *            v_fill              0;
#endif
extern      REAL    UB;
extern      REAL    LB;
```

```
/*-----------------------------------------------------------------------
*       file:               nndump.h
*       desc:               nndump.c header
*       by:                         patrick ko
*       date:               13 aug 1991
*-----------------------------------------------------------------------*/
```

```
#ifdef      __TURBOC__
```

```
void        v_dump              (FILE *, VECTOR *);
void        v_load              (FILE *, VECTOR *);
void        u_dumpweight        (FILE *, UNIT *);
void        u_loadweight        (FILE *, UNIT *);
void        l_dump              (FILE *, LAYER *);
void        l_load              (FILE *, LAYER *);
void        nn_dump             (FILE *, NET *);
void        nn_load             (FILE *, NET *);
#else
void        v_dump              ();
void        v_load              ();
void        u_dumpweight        ();
void        u_loadweight        ();
void        l_dump              ();
void        l_load              ();
void        nn_dump             ();
void        nn_load             ();
#endif
```

```
/*-------------------------------------------------------------------
*           file:       nnerror.h
*           desc:       define all errors
*           by:         patrick ko
*           date:       2 aug 91
*-------------------------------------------------------------------*/
```

```
#define     NNMALLOC            0
#define NNTFRERR    1
#define NNTFIERR    2
#define NNIOLAYER   3
#define NN2MANYLAYER        4
#define NN2FEWPATT          5
#define     NN2MANYHIDDEN   6
#define     NNOUTNOTOPEN    7
#define     NNRFRERR            8
/*      prototype   */
#ifdef      __TURBOC__
int         error       (int);
void        verbose     (char *, char *);
#else
int         error       ();
void        verbose     ();
#endif
```

```
/*-------------------------------------------------------------------
*           file:               nnmath.h
*           desc:               nnmath.c header
*           by:                         patrick ko
*           date:               2 aug 1991
*-------------------------------------------------------------------*/
```

```
#define     sigmoid( netpj )        (1.0 / (1.0 + exp(-1.0 * (netpj))))
```

```
/*-----------------------------------------------------------------------
*       file:       nntrain.h
*       desc:       nntrain.c header file
*       by:                 patrick ko
*       date:       2 aug 91
*-----------------------------------------------------------------------*/
```

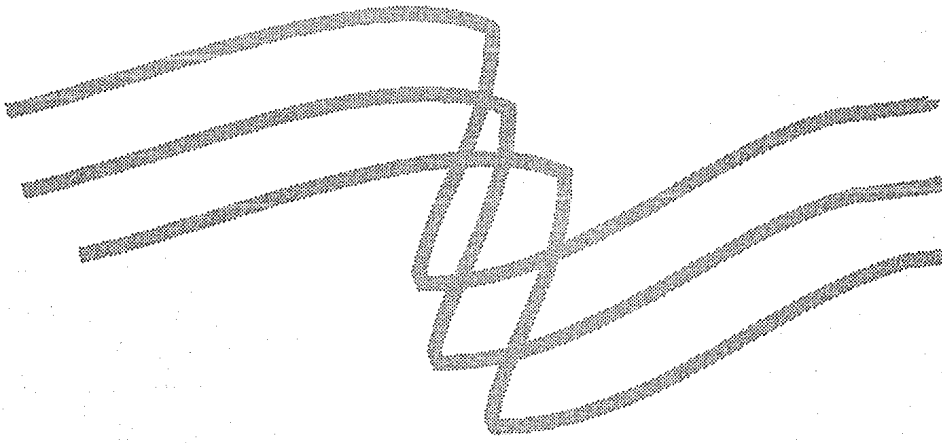
```
#include    "nntype.h"
#ifdef      __TURBOC__
REAL        nnbp_train  (/*NET *,VECTOR **,VECTOR **,INTEGER,REAL,REAL,REAL*,long int,char */);
REAL        nnbp_train1             (/*NET *,VECTOR **,VECTOR **,INTEGER,REAL*/);
void        nnbp_init   (NET *);
void        nnbp_forward            (NET *, VECTOR *);
REAL        nnbp_backward           (NET *, VECTOR *, VECTOR *);
void        nnbp_report             (/*INTEGER, REAL*/);
void        nnbp_coeffadapt         (NET *);
void        nnbp_dweightcalc(NET *, INTEGER, INTEGER);
#else
REAL        nnbp_train  ( );
REAL        nnbp_train1             ( );
void        nnbp_init   ( );
void        nnbp_forward            ( );
REAL        nnbp_backward           ( );
void        nnbp_report             ( );
void        nnbp_coeffadapt ( );
void        nnbp_dweightcalc( );
#endif
```

```
/*-----------------------------------------------------------------------
*       file:       nntype.h
*       desc:       define all types for the neural nets
*       by:         patrick ko
*       date:       2 aug 91
*       revi:       v1.2b - 15 jan 92, adaptive coefficients (beta)
*                   v1.2u - 17 jan 92, revised data structures
*                   v1.31u -20 jan 92, periodic dump
*-----------------------------------------------------------------------*/
```

```
/*      trap multiple nntype include    */
#ifndef     NNTYPE
#define     NNTYPE
typedef     int                     INTEGER;
typedef     int                     FLAG;
typedef     double                  REAL;
typedef struct          {
            INTEGER dim;                    /* vector dimension */
            REAL    * vec;                  /* vector array */
            }       VECTOR;
typedef struct          {
            REAL    out;                    /* output of unit */
            REAL    net;                    /* net product */
```

```
        REAL    dlt;                    /* for delta dpj*/
        REAL    ndlt;                   /* for dpj accumulation (v1.2) */
        VECTOR  *wgtvect;       /* weight vector */
        VECTOR  *dwgtvect1;             /* dweight vector dW at n-1 */
        VECTOR  *dwgtvect2;             /* dweight vector dW at n-2 */
        VECTOR  *dxo;
        REAL    bias;                   /* for bias */
        REAL    dbias1;                 /* for bias at n-1 */
        REAL    dbias2;                 /* for bias at n-2 */
        }       UNIT;
typedef struct  {
        INTEGER dim;                    /* number of units */
        UNIT    **unit;                 /* array of units */
        }       LAYER;
typedef struct {
        INTEGER dim;                    /* number of layers */
        LAYER   **layer;        /* array of layers */
        }       NET;
#define     DimVect(vector)                 ((vector)->dim)
#define     Vect(vector)                    ((vector)->vect)
#define     Vi(vector,i)                    ((vector)->vect[i])
#define     Out(unit)               ((unit)->out)
#define     Net(unit)               ((unit)->net)
#define     Dlt(unit)               ((unit)->dlt)
#define     nDlt(unit)              ((unit)->ndlt)
#define     Weight(unit,i)                  Vi((unit)->wgtvect,i)
#define     dWeight1(unit,i)        Vi((unit)->dwgtvect1,i)
#define     dWeight2(unit,i)        Vi((unit)->dwgtvect2,i)
#define     vWeight(unit)                   ((unit)->wgtvect)
#define     vdWeight1(unit)                 ((unit)->dwgtvect1)
#define     vdWeight2(unit)                 ((unit)->dwgtvect2)
#define     DO(unit,i)              Vi((unit)->dxo,i)
#define     vDO(unit)               ((unit)->dxo)
#define     Bias(unit)              ((unit)->bias)
#define     dBias1(unit)                    ((unit)->dbias1)
#define     dBias2(unit)                    ((unit)->dbias2)
#define     DimvWeight(unit)        DimVect(vWeight(unit))
#define     DimLayer(layer)                 ((layer)->dim)
#define     Unit(layer,i)                   ((layer)->unit[i])
#define     DimNet(nn)                      ((nn)->dim)
#define     Layer(nn,i)             ((nn)->layer[i])
#define DimNetOut(nn)                       (DimLayer(Layer(nn,DimNet(nn)-1)))
#define ground(x,n)             ((x)<(n)?(n):(x))
#define     ETA_DEFAULT                     0.50
#define     ALPHA_DEFAULT                   0.90
#define LAMBDA_DEFAULT                      0.50
#define     ERROR_DEFAULT                   0.01
#define     TOLER_DEFAULT                   0.001
#define     BACKTRACK_STEP                  0.50
#define ME_FLOOR                0.0001
#define     MW_FLOOR                        0.0001
#define     ETA_FLOOR                       0.0001
#endif
```

APENDICE B



```
//=============================================================
//
//                    CLICK.CPP
//
//=============================================================

#define BGI     1
#define BORLAND  1
const int zFAIL=0; const int zEMPTY=0;
const int zYES=1; const int zNO=0;
const int zVGA_12H_1; const int zEGA_10H=2; const int zEGA_EH=3;
const int zMGCA_11H=4; const int zCGA_6H=5; const int zHERC=6;
#define Presione_una_tecla While(KeyCode==0) zKeyboard(); KeyCode=0;
#if defined (BORLAND)
#include <time.h>
#include <string.h>
#include <bios.h>
#include <process.h>
#include <iostream.h>
#include <dos.h>
#elif defined (ZORTECH)
#include <time.h>
#include <string.h>
#include <bios.h>
#include <stdlib.h>
#include <stream.hpp>
#include <dos.h>
#elif defined (MICROSOFT)
#include <time.h>
#include <string.h>
#include <bios.h>
#include <process.h>
#include <iostream.h>
#include <conio.h>
#endif
#include <LIB2D.hpp>
#include <MOUSE.HPP>
static void zStartup(void);                    //inicializa el modo grafico
static void zArguments(int, char far* far*);     //checa los argumentos
static void zKeyboard(void);                    //checa el keystroke
static void zQuit_Pgm(void);                    //termina el programa
static void zPurge(void);                      //vacia el almacenamiento del teclado
static void zStubRoutine(void);
static void zMakeSound(int, double);             //hace un especial sonido
static clock_t zDelay(clock_t, double);          //pausa independiente del cpu
static char *StartUpArg[6]={
     "/M12", "/M10", "/M0E", "/M11", "/M06", "MHC" };
static int CommandLineArg=zNO;                   //indica si el argumento existe
static int CommandLineCompare=0                  //indica si el argumento es legal
static int C0=0,C1=1,C2=2,C3=3,C4=4,C5=5,C6=6,C7=7,C8=8,C9=9,C10=10,C11=11,C12=12,C13=13,C14=14,C15=15;// indice del
codigo de paleta
static int Mode=0;                              //Cual modo grafico es usado
static int Result=0;                           //Captura el resultado de la rutina Graphics
static int CharWidth=0,CharHeigth=0;
static char KeyCode=0;
static char KeyNum=0;
static char SolidFill[]= {255,255,255,255,255,255,255,255};     //100% lleno
static int TextClr=7;
static char Copyrigth[] = "Copyrigh 1992 Quintal and Sotelo. All rights reserved.";
static char Title[]= "USANDO C++ PARA MANEJAR EL MOUSE EN UN PROGRAMA GRAFICO";
static char PressAnyKey[]= "Presione una tecla para continuar";
static char StubMessage[]= "La rutina principal es llamada";
static int X_Res=0, Y_Res=0;                    //la ressolución de la pantalla
PhysicalDisplay Display;                        //Crea un objeto fisicamente el display
Viewport Viewport1(&Display);                   //crea el objeto viewport
PointingDevice Mouse;                           //Crea un objeto pointing-device
static mdata *MPtr;                            //Inicializa el Ptr a la rutina del mouse
```

```
Main(int argc, char *argv[])
{
    int NumArgs; char far* far* Arg;
NumArgs= argc;                          //Graba un número de argumentos
Arg=&argv[0];                           //Graba la dirección de una array de argumentos
zArguments(NumArgs, Arg);                 //Checa la linea de comandos de los argumentos
zStartup(0);                            //Establece el modo de grafico
Display.Init2D(Mode,0,0,X_Res-1,Y_Res-1);       //Conjunto de estados graficos
Result=Display.InitUndo();                //Crea una página oculta
If (Result==zFAIL) zQUIT_Pgm();             //si la página oculta falla

Display.BlankPage();                      //Limpiar la pantalla
Display.SetHue(C7);                       //Conjunto de colores
Display.SetFill(SolidFill,C7);            //Conjunto de stilos
Viewport1.puttext(3,2,TextClr,Title);       //titulo
Viewport1.puttext(4,2,TextClr,Copyrigth);       //Copyrigth
Viewport1.DrawBorder(0,0,X_Res-1,Y_Res-1);      //Dibuja el borde
Viewport1.PutText(6,2,PromptClr,PressAnyKey);     //Muestra el Prompt
Presione_una_tecla                        //Esta macro es definida anteriormente
Result=Mode.Detect(Mode);                   //Inicializa el mouse
if (Result==zFAIL) zQuit_Pgm();
Mptr=Mouse.Data();                        //Graba un puntero a la entrada del mouse
Mouse.Show();                             //Muestra el cursor del mouse
Display.SetHue(C0);Display.SetLine(0xffff);
Display.SetFill(SolidFill,C0); Viewport1.ClearTextLine();
Viewport1.PutText(1,2,C15, "Controles del Mouse:  Izquierdo=Dibuja   derecho=salida");
Display.SetHue(C12);
Viewport1.SetPosition(319,99);

MOUSELOOP;
   Mouse.Info();
   if (MPtr->MouseButton==1)
     {
     Mouse.Hide();
     Viewport1.DrawLine( MPtr->MouseX, MPtr->MouseY);
     Mouse.Show();
     }
   if (MPtr->MouseButton==2) goto MOUSE_DONE;     //Si el boton derecho
```

```
                    //=================================
                    //  Graphic Control Test
                    //
                    //          GCONTROL.CPP
                    //=================================
```

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <graphics.h>
#include "mouse.l"                  // objeto mouse
#include "gcontrol.l"               //objetos boton y scrollbar

char* const ColorStr[] = { "LtBlue", "LtGreen", "LtRed", "LtMagenta", "Yellow", "White" };
int  const RMin = 0, RMax = 6;

main()
{
  Mstatus     Position;
  int         Exit = FALSE, MoveButton, i, j,
                    GDriver = DETECT, GMode, GError;
  viewporttype VRef;
  ScrollBar   HScroll, VScroll;
  VueMeter    CatsEye;
  RadioButton RButton[7];
  Button      ExitButton;
```

```
initgraph( &GDriver, &GMode, "C:\\LENGUAJE\\TC\\BGI");
GError = graphresult();
if( GError )
  {
   printf( "Error Grafico : %s\n",
                grapherrormsg( GError ) );
   printf( "Programa abortado...\n" );
   exit(1);
  }
if( !gmouse.Mreset() ) exit(1);
cleardevice();
for( i=RMin; i<=RMax; i++ )
  RButton[i].Create( 50, 360-(45*i), i+9, 10,
                     ColorStr[i] );
CatsEye.Create( 50, getmaxy()-50, LIGHTRED, 20,
                "VueMeter" );
setviewport( 100, 0, getmaxx(), getmaxy(), TRUE);
getviewsettings( &VRef );
HScroll.Create( 0, getmaxy(), VRef.right-VRef.left-40,
                GREEN, LIGHTGREEN, HORIZ_DIR );
VScroll.Create( getmaxx()-VRef.left-20, 0, getmaxy(),
                GREEN, LIGHTGREEN, VERT_DIR );
ExitButton.SetButtonType( ROUNDED );
ExitButton.Create( HScroll.GetPosition()-20,
                   VScroll.GetPosition(),
                   80, 20, LIGHTRED, "Salida" );
RButton[3].SetState( TRUE );
do
 {
  Position = gmouse.Mpressed( ButtonL );
  if( Position.button_count )
      {
       Exit = ExitButton.ButtonHit();
       if( !Exit )
         do
           {
            MoveButton = FALSE;
            switch( HScroll.ScrollHit() )
             {
              case LEFT :
              case HBAR :
              case RIGHT: MoveButton = TRUE;
             }
            switch( VScroll.ScrollHit() )
             {
              case UP :
              case VBAR :
              case DOWN : MoveButton = TRUE;
             }
            if( MoveButton )
             {
              ExitButton.Move( HScroll.GetPosition()-20,
                               VScroll.GetPosition() );
              CatsEye.Select(
                        (double) HScroll.GetPercent()/100*180 +
                        (double) VScroll.GetPercent()/100*180 );
             }
            else for( i=RMin; i<=RMax; i++ )
                   if( RButton[i].ButtonHit() )
                     {
                      for( j=RMin; j<=RMax; j++ )
                        RButton[j].SetState( FALSE );
                      RButton[i].SetState( TRUE );
                      ExitButton.SetColor(
                        RButton[i].GetColor() );
                      CatsEye.SetColor(
                        RButton[i].GetColor() );
                     }
            Position = gmouse.Mreleased( ButtonL );
           }
         while( !Position.button_count );
      }
}
```

```
  while( !Exit );
  closegraph();          // restore text mode and...
  tmouse.Mreset();       // reset mouse for text operation

}
```

```
//=============================================================//
//          Archivo de Controles Gráficos de objeto
//                        GCONTROL.I //
//=============================================================//
```

```
#include <math.h>
#include <graphics.h>
#include "mouse.i"
#include <string.h>
typedef enum { ROUNDED, SQUARE, THREE_D } ButtonType;
typedef enum { NO_HIT, RIGHT, UP, HBAR, VBAR, LEFT, DOWN } HitType;
typedef int Outline[10];

class Point
 {
  protected :
        int x, y, Color;
        viewporttype VRef;
  public :
        Point();
        void Move( int PtX, int PtY );
        virtual void Draw();
        void Create( int PtX, int PtY, int C );
        void RestoreViewport();
        void SetColor( int C );
        virtual void SetLoc( int PtX, int PtY );
        virtual void Erase();
        int GetColor();
        int GetX();
        int GetY();
 };
class Button : public Point
  {
   protected :
        int State, Rotate, FontSize, TypeFace, SizeX, SizeY;
        ButtonType ThisButton;
        char BtnTxt[40];
   public :
        Button();
        ~Button();
        virtual void Draw();
        void Create( int PtX, int PtY, int Width, int Height, int C,
char* Text);
        virtual void Erase();
        void Invert();
        virtual void Move( int PtX, int PtY );
        void SetColor( int C );
        void SetState( int BState );
        void SetLabel( char* Text );
        void SetButtonType( ButtonType WhatType );
        void SetTypeSize( int TxtSize );
        void SetTypeFace( int TxtFont );
        // ******************  Implementación de SetAll
        void SetAll(ButtonType WhatType, int PtX, int PtY, int Width,
int Height, int C, char* Text);
        int  GetWidth();
        int  GetHeight();
        int  GetState();
        int  GetTextSize();
        int  ButtonHit();
        ButtonType  GetType();
  };
```

```
};

class RadioButton : public Button
  {
    protected:
        int Outline, Radius;
    public:
        RadioButton();
        RadioButton( int PtX, int PtY, int C, int R, char* Text );
        ~RadioButton();
        void Create( int PtX, int PtY, int C, int R, char* Text );
        void Draw();
        void Erase();
        void SetRadius( int R );
        int  GetRadius();
        int  ButtonHit();
  };
class VueMeter : public RadioButton
  {
    int Closure;
    public :
        VueMeter();
        VueMeter( int PtX, int PtY, int C, int R, char* Text );
        ~VueMeter();
        void Create( int PtX, int PtY, int C, int R, char* Text );
        void Draw();
        void Erase();
        void Select( int Degree );
        void Close( int Degree );
        void Open( int Degree );
  };
class ScrollBar : public Button
  {
    private :
        int  LineColor, SPos, Step, ScrollMove;
    public :
        ScrollBar();                    // método constructor
        ScrollBar( int PtX, int PtY, int Size, int C1, int C2, int
Orientation);
        ~ScrollBar();                   // método destructor
        void Create( int PtX, int PtY, int Size, int C1, int C2, int
Orientation);
        virtual void SetLoc( int PtX, int PtY );
        HitType  ScrollHit();
        int  GetPosition();
        int  GetDirection();
        int  GetPercent();
    private :
        virtual void Draw();
        virtual void Erase();
        void SetOutline();
        void SetArrows();
        void SetThumbPad();
        void EraseThumbPad();
  };
////////////////////////////////////////////////////////////////////
//
//      Implementación para el objeto tipo Punto
//
////////////////////////////////////////////////////////////////////

Point::Point()
{
  getviewsettings( &VRef );
}

void Point::SetLoc( int PtX, int PtY )
{
  x = PtX + VRef.left;
  y = PtY + VRef.top;
}

void Point::Draw()
```

```
}
void Point::RestoreViewport()
{
  setviewport( VRef.left,  VRef.top, VRef.right, VRef.bottom, VRef.clip );
}
void Point::Create( int PtX, int PtY, int C )
{
  SetLoc( PtX, PtY );
  Color = C;
  Draw();
}
void Point::Erase()
{
 int Temp;
 Temp = Color;
 Color = getbkcolor();
 Draw();
 Color = Temp;
}
void Point::Move( int PtX, int PtY )
{
 Erase();
 SetLoc( PtX, PtY );
 Draw();
}
void Point::SetColor( int C )
{
  Color = C;
  Draw();
}
int  Point::GetColor()  { return( Color ); }
int  Point::GetX()  { return( x + VRef.left ); }
int  Point::GetY()  { return( y + VRef.top );  }

////////////////////////////////////////////////////////////
//
//        Implementación para el objeto tipo boton
//
////////////////////////////////////////////////////////////

Button::Button()
{
  Rotate = FALSE;
  SetTypeSize( 2 );
  SetTypeFace( TRIPLEX_FONT );
}
Button::~Button()
{
  Erase();
}
void Button::Draw()
{
  int    i, radius = 6, offset = 3, AlignX, AlignY;
  Outline RectArr;
  gmouse.Mshow( FALSE );
  setviewport( x, y, x+SizeX, y+SizeY, TRUE);
  setcolor( Color );
  switch( ThisButton )
   {
     case SQUARE :
          { rectangle( 0,  0,  SizeX, SizeY );
          } break;
     case THREE_D :
          { rectangle( 0, 0, SizeX, SizeY );
            RectArr[0] = RectArr[2] = RectArr[8] = RectArr[1] =
RectArr[7] = RectArr[9] = 1;
            RectArr[4] = RectArr[6] = SizeX-1;
            RectArr[3] = RectArr[5] = SizeY-1;
            setfillstyle( CLOSE_DOT_FILL, Color);
            setlinestyle( USERBIT_LINE, 0, NORM_WIDTH);
            fillpoly( 5, RectArr );
            setlinestyle( SOLID_LINE, 0, NORM_WIDTH);
            rectangle( 2*radius, 2*radius, SizeX-2*radius,
```

```
            setlinestyle( SOLID_LINE, 0, NORM_WIDTH);
            rectangle( 2*radius, 2*radius, SizeX-2*radius,
SizeY-2*radius );
            line( 0, 0, 2*radius, 2*radius );
            line( 0, SizeY, 2*radius, SizeY-2*radius);
            line( SizeX, 0, SizeX-2*radius, 2*radius);
            line( SizeX, SizeY, SizeX-2*radius, SizeY-2*radius );
          } break;
    case ROUNDED :
          {                        // dibujo de esquinas
          arc( SizeX-radius, radius, 0, 90, radius);
          arc( radius, radius, 90, 180, radius );
          arc( radius, SizeY-radius, 180, 270, radius);
          arc( SizeX-radius, SizeY-radius, 270, 360, radius);
                // dibujo de los lados
          line( radius, 0, SizeX-radius, 0 );
          line( radius, SizeY, SizeX-radius, SizeY);
          line( 0, radius, 0, SizeY-radius );
          line( SizeX, radius, SizeX, SizeY-radius);
          }
  }
  switch( ThisButton )                  // rellena el Boton
  {
    case SQUARE  :
    case ROUNDED :
          { RectArr[0] = RectArr[2] = RectArr[8] =
            RectArr[1] = RectArr[7] = RectArr[9] = offset;
            RectArr[4] = RectArr[6] = SizeX-offset;
            RectArr[3] = RectArr[5] = SizeY-offset;
          } break;
    case THREE_D :
          { RectArr[0] = RectArr[2] = RectArr[8] =
            RectArr[1] = RectArr[7] = RectArr[9] = 2*radius+1;
            RectArr[4] = RectArr[6] = SizeX-2*radius-1;
            RectArr[3] = RectArr[5] = SizeY-2*radius-1;
          } break;
  }
  if( State ) setfillstyle( SOLID_FILL, Color );
  else setfillstyle( CLOSE_DOT_FILL, Color);
  setlinestyle( USERBIT_LINE, 0, NORM_WIDTH );
                        // ajuste de fonts y string para fit
  settextstyle( TypeFace, Rotate, FontSize-1/* */ );
  AlignX = (SizeX/2)-3;
  AlignY = (SizeY/2)-3;
  if( State ) setcolor( getbkcolor() );
  outtextxy( AlignX, AlignY, BtnTxt );
  if( State ) setcolor( Color );
  RestoreViewport();
  gmouse.Mshow( TRUE );
}
void Button::Create(int PtX, int PtY, int Width, int Height, int C, char* Text
)
{
  getviewsettings( &VRef );
  setviewport( 0, 0, getmaxx(), getmaxy(), TRUE );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  SetLoc( PtX, PtY );
  if( Width < 20 ) SizeX = 20;          else SizeX = Width;
  if( Height < 20 ) SizeY = 20;         else SizeY = Height;
  if( SizeY > SizeX ) Rotate = TRUE;    else Rotate = FALSE;
  Color = C;
  State  = FALSE;
  strcpy( BtnTxt, Text );
  Draw();
}
void Button::Erase()
{
  gmouse.Mshow( FALSE );
  setviewport( x, y, x+SizeX, y+SizeY, TRUE);
  clearviewport();
  RestoreViewport();
  gmouse.Mshow( TRUE );
}
```

```
  Erase();
  SetLoc( PtX, PtY );
  Draw();
}
void Button::SetLabel( char* Text )
{
  strcpy( BtnTxt, Text );
  Draw();
}
void Button::SetColor( int C )
{
  Color = C;
  Draw();
}
void Button::SetState( int BState )
{
  if( State != BState ) Invert();
}
void Button::SetTypeSize( int TxtSize )
{
  FontSize = TxtSize;
}
void Button::SetTypeFace( int TxtFont )
{
  TypeFace = TxtFont;
}
void Button::SetButtonType( ButtonType WhatType )
{
  ThisButton = WhatType;
}
                    // ********************** implementación de SetAll
void Button::SetAll(ButtonType WhatType, int PtX, int PtY, int Width, int
Height, int C, char* Text)
{
  SetButtonType(WhatType);
  Create(PtX,PtY,Width,Height,C,Text);
  SetColor(C);
  SetLabel(Text);
  Draw();
}
void Button::Invert()
{
  if( State ) State = FALSE;
  else State = TRUE;
  Draw();
}
int Button::GetWidth()      { return( SizeX ); }
int Button::GetHeight()     { return( SizeY ); }
int Button::GetState()      { return( State ); }
int Button::GetTextSize()   { return( FontSize); }
ButtonType Button::GetType() { return( ThisButton); }
int Button::ButtonHit()
 {
   Mstatus  P = gmouse.Mpressed( ButtonL );
   if( ( P.xaxis >= x ) && ( P.xaxis <= x+SizeX) &&
    ( P.yaxis >= y ) && ( P.yaxis <= y+SizeY))
     {
    Invert();
    return( TRUE );
     }
   return( FALSE );
 }

   ////////////////////////////////////////////////////
   //
   //    implementación para el objecto tipo RadioButton
   //
   ////////////////////////////////////////////////////
RadioButton::RadioButton() { }
RadioButton::RadioButton( int PtX, int PtY, int C, int R, char* Text)
   {
    Create( PtX, PtY, C, R, Text );
   }
```

```
    {
     Create( PtX, PtY, C, R, Text );
    }

RadioButton::~RadioButton() {  Erase();  }
void RadioButton::Create( int PtX, int PtY, int C, int R, char* Text )
   {
     getviewsettings( &VRef );
     settextjustify( CENTER_TEXT, CENTER_TEXT );
     SetLoc( PtX, PtY );
     Radius = R;
     Outline = C | 0x08;
     Color = C;
     State = FALSE;
     strcpy( BtnTxt, Text );
     Draw();
   }
void RadioButton::Draw()
  {
   int XAsp, YAsp, i, OldColor = getcolor();
   RestoreViewport();
   getaspectratio( &XAsp, &YAsp );
   setcolor( Outline );
   gmouse.Mshow( FALSE );
   if( State ) setfillstyle( SOLID_FILL, Color);
   else        setfillstyle( INTERLEAVE_FILL, Color );
   fillellipse( x, y, Radius, Radius * (double) ( XAsp / YAsp ) );
   settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
   outtextxy( x, y + Radius + 10, BtnTxt );
   gmouse.Mshow( TRUE );
   setcolor( OldColor );
}
void RadioButton::Erase()
  {
   int OldColor = Color;
   Color = getbkcolor();
   Outline = Color;
   Draw();
   Color = OldColor;
   Outline = OldColor | 0x08;
  }
void RadioButton::SetRadius( int R )  {  Radius = R; }
int  RadioButton::GetRadius()   {  return( Radius ); }
int  RadioButton::ButtonHit()
  {
    int  OffX, OffY;
    Mstatus  P = gmouse.Mpressed( ButtonL );
    OffX = abs( P.xaxis - x );
    OffY = abs( P.yaxis - y );
    if( ( OffX < 2 * Radius ) && ( OffY < 2 * Radius ) )
      if( hypot( OffX, OffY ) < Radius )
       {
       Invert();
       return( TRUE );
       }
    return( FALSE );
  }
    //=====================================================
    // Implementacion para el objecto tipo VueMeter
    //=====================================================
VueMeter::VueMeter()  {  }
VueMeter::VueMeter( int PtX, int PtY,
            int C, int R, char* Text )
 {
   Create( PtX, PtY, C, R, Text );
 }
VueMeter::~VueMeter()  {  Erase();  }
void VueMeter::Create( int PtX, int PtY, int C, int R, char* Text )
  {
   getviewsettings( &VRef );
   settextjustify( CENTER_TEXT, CENTER_TEXT);
   SetLoc( PtX, PtY );
   Radius = R;
```

```
}
void VueMeter::Draw()
 {
  int  XAsp, YAsp;
  RestoreViewport();
  getaspectratio( &XAsp, &YAsp );
  if( Closure < 0 )   Closure = 0;
  if( Closure > 360 ) Closure = 360;
  gmouse.Mshow( FALSE );
  setcolor( Color );
  setfillstyle( SOLID_FILL, Color );
  sector( x, y, 0, Closure, Radius,
      Radius * (double) ( XAsp / YAsp ) );
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  outtextxy( x, y + Radius + 10, BtnTxt );
  setcolor( WHITE );
  setfillstyle( CLOSE_DOT_FILL, Color );
  if( Closure < 360 )
    sector( x, y, Closure, 360, Radius, Radius * (double) ( XAsp / YAsp ) );
  gmouse.Mshow( TRUE );
 }
void VueMeter::Erase()
 {
  int  XAsp, YAsp;
  getaspectratio( &XAsp, &YAsp );
  gmouse.Mshow( FALSE );
  setcolor( getbkcolor() );
  setfillstyle( SOLID_FILL, getbkcolor() );
  sector( x, y, 0, 360, Radius, Radius * (double) ( XAsp / YAsp ) );
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  outtextxy( x, y + Radius + 10, BtnTxt );
  setcolor( WHITE );
  gmouse.Mshow( TRUE );
 }
void VueMeter::Select( int Degree )
 {
  Closure = Degree;
  Draw();
 }
void VueMeter::Close( int Degree )
 {
  Closure += Degree;
  Draw();
 }
void VueMeter::Open( int Degree )
 {
  Closure -= Degree;
  Draw();
 }
//=======================================================
// Implementación para el objeto tipo ScrollBar
//=======================================================
ScrollBar::ScrollBar()
 {
  x = y = SizeX = SizeY = Color =
  LineColor = SPos = Step = ScrollMove = 0;
 }
ScrollBar::ScrollBar( int PtX, int PtY, int Size,
            int C1, int C2, int Orientation )
 {
  Create( PtX, PtY, Size, C1, C2, Orientation);
 }
ScrollBar::~ScrollBar()  {  Erase(); }
void ScrollBar::Create( int PtX, int PtY, int Size, int C1, int C2, int
Orientation )
 {
  getviewsettings( &VRef );
  if( Size < 100 ) Size = 100;
  ScrollMove = Orientation;
  SPos = 21;
  Step = Size / 100;
```

```
  if( Size < 100 ) Size = 100;
  ScrollMove = Orientation;
  SPos = 21;
  Step = Size / 100;
  switch( ScrollMove )
   {
    case VERT_DIR :
          {
           SizeX = 20;
           SizeY = Size;
           while( PtX + SizeX > VRef.right ) PtX--;
           break;
          }
    case HORIZ_DIR :
          {
          SizeX = Size;
          SizeY = 20;
          while( PtY + SizeY > VRef.bottom ) PtY--;
          }
   }
  SetLoc( PtX, PtY );
  LineColor = C1;
  Color = C2;
  Draw();
}
void ScrollBar::SetLoc( int PtX, int PtY )
 {
  Point::SetLoc( PtX, PtY );
  while( ( x + SizeX ) > VRef.right )  SizeX--;
  while( ( y + SizeY ) > VRef.bottom ) SizeY--;
 }
void ScrollBar::EraseThumbPad()
 {
  switch( ScrollMove )
    {
   case VERT_DIR :
          setviewport( x+2, y+SPos, x+18, y+SPos+19, TRUE );
          break;
   case HORIZ_DIR :
          setviewport( x+SPos, y+2, x+SPos+19, y+18, TRUE );
    }
  clearviewport();
  RestoreViewport();
 }
void ScrollBar::SetThumbPad()
 {
  Outline RectArr;
  int    OldColor = getcolor();
  setviewport( x, y, x+SizeX, y+SizeY, TRUE );
  setcolor( LineColor );
  setfillstyle( CLOSE_DOT_FILL, Color );
  switch( ScrollMove )
   {
    case VERT_DIR:
    {
    RectArr[0] = RectArr[2] = RectArr[8] = 2;
    RectArr[1] = RectArr[7] = RectArr[9] = SPos;
    RectArr[4] = RectArr[6] = 18;
    RectArr[3] = RectArr[5] = SPos+19;
    break;
    }
    case HORIZ_DIR :
          {
          RectArr[0] = RectArr[2] = RectArr[8] = SPos;
          RectArr[1] = RectArr[7] = RectArr[9] = 2;
          RectArr[4] = RectArr[6] = SPos+19;
          RectArr[3] = RectArr[5] = 19;
           }
   }
  fillpoly( 5, RectArr );
  setcolor( OldColor );
  RestoreViewport();
}
```

```
  setviewport( x, y, x+SizeX, y+SizeY, TRUE);
  RectArr[0] = RectArr[2] = RectArr[8] = 1;
  RectArr[1] = RectArr[7] = RectArr[9] = 1;
  RectArr[4] = RectArr[6] = SizeX-1;
  RectArr[3] = RectArr[5] = SizeY-1;
  setfillstyle( SOLID_FILL, Color );
  setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
  fillpoly( 5, RectArr );
                              // limpia el centro de la barra
  setfillstyle( SOLID_FILL, getbkcolor() );
  switch( ScrollMove )
   {
    case VERT_DIR :
          {
           RectArr[1] = RectArr[7] = RectArr[9] =21;
           RectArr[3] = RectArr[5] = SizeY-21;
           break;
          }
    case HORIZ_DIR :
          {
           RectArr[0] = RectArr[2] = RectArr[8] = 21;
           RectArr[4] = RectArr[6] = SizeX-21;
          }
   }
  fillpoly( 5, RectArr );
 }
void ScrollBar::SetArrows()
 {
  setcolor( getbkcolor() );
  setlinestyle( SOLID_LINE, 0, THICK_WIDTH );
  switch( ScrollMove )
     {
     case VERT_DIR :
          {
           line( 10, 4,  4, 12 );
           line( 10, 4, 16, 12 );
           line( 10, 4, 10, 16 );
           line( 10, SizeY-4,  4, SizeY-12 );
           line( 10, SizeY-4, 16, SizeY-12 );
           line( 10, SizeY-4, 10, SizeY-16 );
           break;
          }
     case HORIZ_DIR :
          {
           line( 4, 10, 12,  4 );
           line( 4, 10, 12, 16 );
           line( 4, 10, 16, 10 );
           line( SizeX-4, 10, SizeX-12,  4 );
           line( SizeX-4, 10, SizeX-12, 16 );
           line( SizeX-4, 10, SizeX-16, 10 );
          }
     }
  setlinestyle( SOLID_LINE, 0, NORM_WIDTH );
 }
void ScrollBar::Draw()
 {
  int  OldColor = getcolor();
  SetOutline();                  // scrollbar outline
  SetArrows();                    // scrollbar arrows
  SetThumbPad();                   // draw thumbpad
  setcolor( OldColor );            // restore orig color
 }
void ScrollBar::Erase()
 {
  gmouse.Mshow( FALSE );
  setviewport( x, y, x+SizeX, y+SizeY, TRUE);
  clearviewport();
  RestoreViewport();
  gmouse.Mshow( TRUE );
 }
HitType ScrollBar::ScrollHit()
 {
  HitType  Result = NO_HIT;
```

```
}
HitType ScrollBar::ScrollHit()
 {
  HitType  Result = NO_HIT;
  int      NPos = 0;

  Mstatus  P = gmouse.Mpressed( ButtonL );
  switch( ScrollMove )
   {
    case VERT_DIR :
          if( ( P.xaxis >= x ) && ( P.xaxis <= x+20) &&
           ( P.yaxis >= y ) )
          if( P.yaxis <= y+20 )  Result=UP;
          else
          if( P.yaxis <= y+SizeY-31 ) Result = VBAR;
          else
          if( P.yaxis <= y+SizeY )   Result = DOWN;
          break;
    case HORIZ_DIR :
          if( ( P.yaxis >= y ) && ( P.yaxis <= y+20) &&
           ( P.xaxis >= x ) )
          if( P.xaxis <= x+20 )      Result = LEFT;
          else
           if( P.xaxis <= x+SizeX-31 ) Result = HBAR;
          else
           if( P.xaxis <= x+SizeX )   Result = RIGHT;
   }
  if( Result == NO_HIT ) return( Result );
   switch( Result )
    {
    case LEFT :
    case UP : NPos = SPos-Step; break;
    case RIGHT :
    case DOWN : NPos = SPos+Step; break;
    case HBAR : NPos = P.xaxis - ( x + 10 ); break;
    case VBAR : NPos = P.yaxis - ( y + 10 );
    }
   if( NPos < 21 ) NPos = 21;
   switch( Result )
    {
    case LEFT :
    case RIGHT : if( NPos > SizeX-41 ) NPos = SizeX-41; break;
    case UP :
    case DOWN : if( NPos > SizeY-41 ) NPos = SizeY-41;
    }
   gmouse.Mshow( FALSE );
   EraseThumbPad();
   SPos = NPos;
   SetThumbPad();
   gmouse.Mshow( TRUE );
   return( Result );
 }
int ScrollBar::GetPosition()   { return( SPos ); }
int ScrollBar::GetDirection()  { return( ScrollMove ); }
int ScrollBar::GetPercent()
 {
  switch( ScrollMove )
    {
     case HORIZ_DIR : return( 100*(double)(SPos-21)/(double)(SizeX-41));
     case VERT_DIR : return( 100*(double)(SPos-21)/(double)(SizeY-41));
    }
 }
          //============fin de los métodos ============//
```

```
#include <alloc.h>
#include <graphics.h>
#include <stdarg.h>
#include <stdio.h>
void erasestr(int xloc, int yloc, char *str)
  {
    struct textsettingstype textinfo;
    int  xdim, ydim;
    void *textimage;
    gettextsettings(&textinfo);
    switch (textinfo.direction)
    {
          case HORIZ_DIR: xdim = textwidth(str);
                          ydim = textheight(str);
                          xloc--; break;
          case VERT_DIR:  ydim = textwidth(str);
                          xdim = textheight(str);
                          yloc++;
    }
    switch (textinfo.horiz)
    {
          case LEFT_TEXT:                  break;
          case CENTER_TEXT: xloc -= xdim / 2;  break;
          case RIGHT_TEXT:  xloc -= xdim;
    }
    switch (textinfo.vert)
    {
          case BOTTOM_TEXT: yloc -= ydim;    break;
          case CENTER_TEXT: yloc -= ydim / 2; break;
          case TOP_TEXT: ;
    }
    while(xloc<0)
          {
            xloc++; xdim--;
          }
    while(yloc<0)
          {
            yloc++; ydim--;
          }
    textimage = malloc(imagesize(xloc, yloc, xdim, ydim));
    getimage(xloc, yloc, xloc+xdim, yloc+ydim, textimage);
    putimage(xloc, yloc, textimage, XOR_PUT);
    free(textimage);
  }

  ////////////////////////////////////////////////////////////////////////
  //
  // gprintf. usado como printf excepto la salida es duda en la pantalla
  // en el modo grafico que especifica la coordenada. Dependiendo en la
  // dirección del texto, las coordenadas xloc o yloc retornan de acuerdo al
  // string.
  //
  ////////////////////////////////////////////////////////////////////////

void gprintf(int *xloc, int *yloc, char *fmt, ...)
  {
    va_list argptr;
    char str[140];
    struct textsettingstype textinfo;
    int pos_adj = *xloc;
    va_start(argptr, fmt);
    vsprintf(str, fmt, argptr);
    gettextsettings(&textinfo);
    outtextxy(pos_adj, *yloc, str);
    switch (textinfo.direction)
          {
            case HORIZ_DIR: *yloc += textheight(str) + 2;

            case VERT_DIR: *xloc += textheight(str) + 2;
          };
          va_end(argptr);
  }
```

```
    int pos_adj = *xloc;
    va_start(argptr, fmt);
    vsprintf(str, fmt, argptr);
    gettextsettings(&textinfo);
    outtextxy(pos_adj, *yloc, str);
    switch (textinfo.direction)
          {
            case HORIZ_DIR: *yloc += textheight(str) + 2;

            case VERT_DIR: *xloc += textheight(str) + 2;
          };
          va_end(argptr);
  }

//////////////////////////////////////////////////////////////////
//
// gprinte:  usado como GPRINTF excepto en el area cuando el texto podría
// escribir en la pantalla primero resetea a el color que antecede.
//
//////////////////////////////////////////////////////////////////

void gprinte(int *xloc, int *yloc, char *fmt, ...)
 {
   va_list argptr;
   char str[140];
   struct textsettingstype textinfo;
   int pos_adj = *xloc;
   va_start(argptr, fmt);
   vsprintf(str, fmt, argptr);
   gettextsettings(&textinfo);
   erasestr(*xloc, *yloc, str);
   outtextxy(pos_adj, *yloc, str);
   switch (textinfo.direction)
    {
          case HORIZ_DIR: *yloc += textheight(str) + 2;
          case VERT_DIR: *xloc += textheight(str) + 2;
    };
    va_end(argptr);
 }

//////////////////////////////////////////////////////////////////
//
// GPRINTXY: Usado como Gprinte excepto en el area donde en las coordenadas de
// la pantalla son pasados por un rango que pasa por la dirreción del puntero
//
//////////////////////////////////////////////////////////////////
 void gprintxy(int xloc, int yloc, char *fmt, ...)
  {
   va_list argptr;
   char  str[140];
   struct textsettingstype textinfo;
   va_start(argptr, fmt);
   vsprintf(str, fmt, argptr);
   gettextsettings(&textinfo);
   erasestr(xloc, yloc, str);
   outtextxy(xloc, yloc, str);
   va_end(argptr);
}
```

```
//////////////////////////////////////////////////////////////////////////////
//
//   conjunto de texto de rutinas de entrada y salida de texto para el modo grafico
//
//                                   GTEXT.CPP
//
//////////////////////////////////////////////////////////////////////////////

#include <graphics.h>
#include <stdarg.h>
#include <stdio.h>
```

```
int cnt;
struct fillsettingstype oldfill;          // actual fill setting
char userfillpattern[8];                  // actual patrón user-fill
va_start(argptr, fmt);                    // inicializa la función va_functions
cnt = vsprintf(str, fmt, argptr);         // imprime el string al buffer
if (str[0] == NULL) return(0);
                                                        // limpia el espacio cuando el texto es imprimido
getfillsettings(&oldfill);
if (oldfill.pattern == USER_FILL)
        getfillpattern(userfillpattern);
setfillstyle(SOLID_FILL, getbkcolor());
bar(xloc, yloc, xloc+textwidth(str), yloc+textheight("H")*5/4);
if (oldfill.pattern == USER_FILL)
        setfillpattern(userfillpattern,oldfill.color);
else
        setfillstyle(oldfill.pattern,oldfill.color);
outtextxy(xloc, yloc, str);               // Escribe el string a la pantalla
va_end(argptr);                           // termina va_functions
return(cnt);                              // retorna al conteo de conversión

                                                // Un grafico basado en la función printf(). Esto actualizará la actual
                                                // Posición. Asumiendo la justificación LEFT_TEXT y HORIZ_DIR.
int gprintf(char *fmt, ...)
{
va_list argptr;                           // puntero en el argumento list
char str[BUFSIZE];                        // almacenamiento para la construccion de una cadena
int cnt;                                  // resultado de la conversion del string
struct fillsettingstype oldfill;          // actual fill setting
char userfillpattern[8];                  // actual patrón user-fill
int xloc, yloc;                           // actual posición

va_start(argptr, fmt);
cnt = vsprintf(str,fmt,argptr);
if (str[0] == NULL) return(0);
                                                        // limpia el espacio cuando el texto es imprimido
xloc = getx(); yloc = gety();
getfillsettings(&oldfill);
if (oldfill.pattern == USER_FILL)
getfillpattern(userfillpattern);
setfillstyle(SOLID_FILL,getbkcolor());
bar(xloc, yloc, xloc+textwidth(str), yloc+textheight("H")*5/4);
if (oldfill.pattern == USER_FILL)
        setfillpattern(userfillpattern,oldfill.color);
else
        setfillstyle(oldfill.pattern,oldfill.color);
outtext(str);                             // escribe el string a la pantalla
va_end(argptr);                           // termina va_functions
return(cnt);                              // retorna al conteo de conversión
}
                                                // Un grafico basado en la función getche()
int ggetche(void)
{
  char ch;
  ch = getch();
  gprintf("%c",ch);
  return(ch);
}
                                                // Un grafico basado en la función puch()
int gputch(int c)
{
  char buffer[2];
  sprintf(buffer,"%c",c);
  gprintf(buffer);
  return(c);
}
                                                // Un grafico basado en la rutina de entrada de texto. Retornando el string completo.
                                                // Esto soporta el caracter backspace.
char *ggets(char *buffer)
{
 int currloc, maxchars, oldcolor;
 struct viewporttype view;
 char ch, charbuff[3];
 buffer[0] = '\0';
```

```
    gprintf(buffer);
    return(c);
}
                                        // Un grafico basado en la rutina de entrada de texto. Retornando el string completo.
                                        // Esto soporta el caracter backspace.
char *ggets(char *buffer)
{
 int currloc, maxchars, oldcolor;
 struct viewporttype view;
 char ch, charbuff[3];
 buffer[0] = '\0';
 currloc=0;
 getviewsettings(&view);
 maxchars = (view.right - getx() / textwidth("M") - 1;
 if (maxchars<=0) return(NULL);
 gprintfxy(getx(), gety(), "_");
 while ((ch+getch()) != CR) {
 if (ch == BS) {
        if (currloc > 0) {
          currloc--;
          if (currloc <= maxchars) {
            oldcolor = getcolor() ;
            setcolor(getbkcolor());
            sprintf(charbuff,"%c", buffer[currloc]);
            gprintfxy(getx() - textwidth(charbuff), gety(),"%c_", buffer[currloc]);
            setcolor(oldcolor);
            moveto(getx() - textwidth(charbuff), gety());
          }
        }
 }
 else {
 if (currloc < maxchars) {
        oldcolor = getcolor();
        setcolor(getbkcolor());
        gprintfxy(getx(), gety(), "_");
        setcolor(oldcolor);
        buffer[currloc] = ch;
        gputch(ch);
        currloc++;
        }
        else
          putch(0x07);
        }
        if (currloc < maxchars)
          gprintfxy(getx(), gety(), "_");
        }
        if (currloc <= maxchars) {
          oldcolor = getcolor();
          setcolor(getbkcolor());
 gprintfxy(getx(), gety(), "_");
          setcolor(oldcolor);
        }
        buffer[currloc] = '\0';
        return(buffer);
 }
                                        // Un grafico basado en la función scanf(). Esta actualiza la actual posición.
                                        // Asumiendo la justificación LEFT_TEXT y HORI_DIR.
 int gscanf(char *fmt, ...)
 {
 va_list argptr;
 char str[BUFSIZE];
 int cnt;
 va_start(argptr, fmt);
 ggets(str);
 cnt = vsscanf(str, fmt, argptr);       // convierte la entrada de acuerdo al formato del string
                                                                        // termina la función va_
 va_end(argtr);                  // termina el numero de sucesiones
 return(cnt);                    // entradas de conversión

 }
                                                // Un grafico basado en la función scanf(). Esto no afecta a la
actual
                                                // posición
```

```
        cnt = vsscanf(str, fmt, argptr);
        va_end(argptr);
        moveto(oldx, oldy);
        return(cnt);
}
```

```
////////////////////////////////////////////////////////////////////////
//
// iconed.cpp -- Es un editor de icono. Este programa habilita la creatividad
// de un icono, edita un existente, o salva un patron de icono en un archivo
// que puede ser usado por programa grafico. El mouse soporta los modelos:
// CGAIII, EGA y VGA.
//
////////////////////////////////////////////////////////////////////////
```

```
#include <stdio.h>
#include <graphics.h>
#include <stdarg.h>
#include <alloc.h>
#include <conio.h>
#include <process.h>
#include "mouse.h"                      // rutinas del mouse y del teclado
#include "kbmouse.h"
const int BIGICONLEFT = 20;                     // lado izquierdo del icono
const int BIGICONTOP = 50;              // lado superior del icono
const int BIGBITSIZE = 8;               // son del 8 pixel
const int ICONWIDTH = 16;               // el tamaño del icono es de 16 x 16
const int ICONLEFT = 400;
const int ESC = 27;

                                                        // Estas son las funciones de iconed.cpp
void draw_enlarged_icon(void);
void toggle_bigbit(int x, int y);
void toggle_icons_bit(int x, int y);
void init_bigbit(void);
void toggle_cursor(int x, int y);
void save_icon(void);
void read_icon(void);
void init_graphics(void);
void show_icon(void);

                                                        // Variables globales
void *bigbit;
unsigned char icon[ICONHEIGHT][ICONWIDTH];      // patrón de 16 x 16
kbdmouseobj mouse;
int aspect;

main()
 {
  int x, y, c;
  read_icon();
  init_graphics();
  mouse.init();                         // inicializa la pantalla
  draw_enlarged_icon();                 // con un patrón de iconos
  show_icon();                          // Dibuja el Icono cuando
  mouse.hide();                         // se usa el mouse. primero
                                                        // se turna en off antes de
                                                        // escribir
  outtextxy(BIGICONLEFT,10,"Presione ESC para finalizar ...");
  outtextxy(BIGICONLEFT,BIGICONTOP-20,"Iconoalargado");
  outtextxy(ICONLEFT,BIGICONTOP-20,"Actual Icono");
  mouse.show();                         // redibuja el mouse en la pantalla
  while ((c=mouse.waitforinput(LEFT_BUTTON)) !=ESC)
        {
                                                        // espera la entrada desde el mouse/teclado
                                                        // si la entrada es ESC. Sale el programa
         if (c < 0)                     // si la entrada < 0 entonces un boton del
          {                             // mouse se acciona.
           mouse.getcoords(x,y);  // si el botton <0 el mouse se activa
```

```
  outtextxy(BIGICONLEFT,10,"Presione ESC para finalizar ...");
  outtextxy(BIGICONLEFT,BIGICONTOP-20,"Iconoalargado");
  outtextxy(ICONLEFT,BIGICONTOP-20,"Actual Icono");
  mouse.show();                          // redibuja el mouse en la pantalla
  while ((c=mouse.waitforinput(LEFT_BUTTON)) !=ESC)
          {
                                                          // espera la entrada desde el mouse/teclado
                                                                  // si la entrada es ESC. Sale el programa
           if (c < 0)                          // si la entrada < 0 entonces un boton del
            {                                  // mouse se acciona.
            mouse.getcoords(x,y);  // si el botton <0 el mouse se activa
            toggle_bigbit(x,y);
            }
 }
 mouse.hide();
 closegraph();
 printf("Quiere salvar este icono en un archivo? (Y) ");
 if (getch() != 'n')              // Salva el icono a un archivo si se
  save_icon();                    // teclea "n"
 return(0);
//Esta rutina dibuja una vista de el patrón de iconos al ser editadas
//El icono alargado es dibujado con BIGICONLEFT, BIGICONTOP, al accionarse
//el bottom derecho.
void draw_enlarged_icon(void)
{
  int i, right, bottom;

  setlinestyle(DOTTED_LINE,0,NORM_WIDTH);
  right = BIGICONLEFT+ICONWIDTH*aspect*(BIGBITSIZE+NORH_WIDTH);
  bottom = BIGICONTOP+ICONHEIGHT*(BIGBITSIZE+NORH_WIDTH);
  mouse.hide();                        // dibuja las lineas verticales y
                                                          // horizontales al hacer el patron del icono
    for (i=0; i<=ICONHEIGHT; i++)
          line(BIGICONLEFT,BIGICONTOP+i*(BIGBTSIZE+NORM_WIDTH),right,BIGICONTOP+i*(BIGBTSIZE+NORM,WIDTH));
    for (i=0; i<=ICONWIDTH; i++)
          line(BIGICONLEFT+aspect*(i*(BIGBITSIZE+NORM,WIDTH)),BIGIContop,
BIGICONLEFT+aspect*(i*(BIGBITSIZE+NORH,WIDTH)),bottoh);
    mouse.show();
    init_bigbit();                     // Se crea la imagen big bit
}
                                        //crea la imagen de un simple bit. Esta imagen puede usarse al cruzarse bit antes

void init_bigbit(void)
{
 int bbx, bby, i, J;
 bbx = BIGICONLEFT;                    // crea la imagen en la esquina
 bby = BJGICONTOP:                     // izquierda superior de el icono
 mouse.hide();
 for (J=bby+1; J<=bby+BIGBITSIZE,1; j++)
  {
   for (i=bbx+1: i<=bbx+aspect*BIGBITSIZE: i++)
   putpixel(i,j,getmax<olor());
  }
 bigbit = malloc(imagesize(bbx,bby,bbx+aspect*BIGBITSIZE, bby+BIGBITSIZE));
 getimage(bbx+1,bby+1,bbx+aspect*BIGBITSIZE,bby+BIGBITSIZE,bigbit);
                                       // borra la imagen
 putimage(bbx+1,bby+1,bigbit,XOR_PUT);
 mouse.show();                         // torna el mouse en on
}

// cuando el usuario realiza un click en un patron de icono
// cruza el bit y el pixel en el patron de icono. Esta rutina acepta las
// coordenadas en la pantalla que especifican donde el boton del mouse
// es presionado. Los dos loop de prueba se ven cuales bit logicos en el
// patron de icono puede ser cruzado.

void toggle_bigbit(int x, int y)
{
 int i, J, line1, line2, col1, col2;

  for (J=0; j<ICONHEIGHT; j++)
     {
      line1 = BIGICONTOP+J*(BIGBITSIZE+NORM,WIDTH);
```

```
                mouse.show();
                toggle_icons_bit(i,j);
                return;
              }
          }
        }
      }
}

void toggle_icons_bit(int x, int y)
  {
   int i;
   mouse.hide();                                    //cambia el color de el pixel
   if (getpixel(asPect*x+ICONLEFT,BIGICONTOP+y) != BLACK)
     {
      for (i=0; i<aspect; i++)
        putpixel(aspect*x+i+ICONLEFT,BIGICONTOP+y,BLACK);
        icon[Y][x] = 0;
     }
   else
     {                                              // dibuja todos los pixeles con el
                                                    // maximo de colores
      for (i=0; i<aspect; i++)
        putpixe>(aspect*x+i+ICONLEFT.BIGICONTOP+y,getmaxcolor());
      iCon[Y][x] = 1;
     }
   mouse.show();
  }
                                                // Esta rutina escribe el patron de iconos en un archivo. El usuario se
                                                // muestra por el filename al escribir el archivo en el. el formato de el
                                                // archivo es dado al principio del programa

void save_icon(void)
{
  char filename[80];
  FILE *iconfile;
  int i, j;
  printf("\n Nombre del archivo donde se almacena el icono:");
  scanf("%s",filename);
  if ((iconfile = fopen(filename,"w")) += NULL)
     {
      printf("No se abre el archivo.\n");
      return;
     }
                                                // Escribe la cabezera del archivo:
  fprintf(iconfile,"%d %d\n",ICONWIDTH,ICONHEIGHT);
  for (1=0; J<1CONHEIGHT; j++)
     {                                          // escribe el patron del icono
      for (i=O; i<ICONWIDTH; i++)                       // en un archivo
      fprintf(iconfile,'%x ',icon[j][i]);
      fprintf(iconfile,'\n');
     }
  fclose(iconfile);
}
void read,icon(void)
{
  char filename1801:
  FILE *iconfile;
  int i. J. width. height:
  for IJ=O: J<ICONHEIGHT; J++) {   )) Initiali/e the icon a
    for (i=O; i<ICONWIDTH: i++)     /) to all /eroes
      iCOn(J](it = 0;
  )
  printf('\n\n----------  ICON EDITOR ----------\n\n');
  printf('Do you want to edit an existing icon? (Y) ');
  if (9etch() == 'n') return;
  printf('\nEnter the name of the file to read the icon froi
  scanf('%s'.filename):
  if ((iconfile = fopen(filename.'
  if (iCOn[yl[X1 == 1) {
    putimage(BIGICONLEFT+aspect'(x*(BIGBITSIZE+NORM,WIDTH)
          BIGICONTOP+y@(BIGBITSIZE+NORM,WIDTH)+1,
```

```
    iCOn(J)(il = 0;
  }
  printf('\n\n---------- ICON EDITOR ----------\n\n');
  printf('Do you want to edit an existing icon? (Y) ');
  if (getch() == 'n') return;
  printf('\nEnter the name of the file to read the icon fro
  scanf('%s',filename);
  if ((iconfile = fopen(filename,'
  if (iCOn[y][XI == 1) {
    putimage(BIGICONLEFT+aspect'(x*(BIGBITSIZE+NORM,WIDTH)
          BIGICONTOP+y@(BIGBITSIZE+NORM,WIDTH)+1,
          bigbit,XOR,PUT);
    toggle,icons,bit(x,y);
   }
}
```

```
////////////////////////////////////////////////////////////////
//
//                    ICONEDIT.CPP
//                  Icon_Image_Editor
//
////////////////////////////////////////////////////////////////
```

```
#include <conio.h>
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <graphics.h>
#include <\sys\stat.h>
#include <gprint.i>
#include <gcontrol.i>
typedef  int  Outline[10];
Mstatus  Position;
palettetype  OrgPalette;
viewporttype  VRef;
int   ActiveColor = WHITE, HSize = 31, VSize = 31, i, j,
      GridColor = BLUE, PixColor, GDriver, GMode, GError;
unsigned int  Image[32][32];
Button        ExtBtn,  ClrBtn, HighBtn, WideBtn,
                MonoBtn, InvBtn, SaveBtn, ReadBtn;
RadioButton   RButton[16];
void Fillsquare( int x, int y, int FillStyle, int Color )
{
  int  OrgColor = getcolor();
  Outline  outline;
  setcolor( GridColor );
  outline[0] = outline[6] = outline[8] = x;
  outline[1] = outline[3] = outline[9] = y;
  outline[2] = outline[4] = x+10;
  outline[5] = outline[7] = y+10;
  setfillstyle( FillStyle, Color );
  fillpoly( 5, outline );
  setcolor( OrgColor );
}
void Beep()
{
  sound( 220 ); delay( 100 ); nosound();
                              delay(  50 );
  sound( 440 ); delay( 100 ); nosound();
}
void PaintGrid()
{
  char * Temp1, Temp2;
  setcolor( WHITE );
  gmouse.Mshow( FALSE );
```

```
            {
              Fillsquare(i*10+300, j*10, INTERLEAVE_FILL, Image[i] [j] );
              putpixel(250+i, 10+j, Image[i] [j] );
            }
  gmouse.Mshow(TRUE);
}
void NoteColor()
  {
    char* const ColorNames[] =
            { "Black", "Blue", "Green", "Cyan", "Read", "Magents", "Brown", "LigthGray", "DarkGray", "LigthBlue",
              "LigthGreen", "LigthCyan", "LightRed", "LtMagents", "Yellow", "White" };
    setviewport(0, 310, 80, 330, TRUE);
    setcolor(ActiveColor);
    settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
    settextjustify( CENTER_TEXT, CENTER_TEXT );
    clearviewport();
    outtextxy( 40, 10, ColorNames[ActiveColor] );
    setviewport( 0, 0, getmaxx(), getmaxy(), TRUE );
  }
void ClearImage()
  {
    int i, j;
    for( i=0; i<=HSize; i++ )
          for( j=0; j<VSize; j++ )
            Image[i] [j] = 0;
    PaintGrid();
    ClrBtn.SetState( FALSE );
    if( InvBtn.GetState() ) InvBtn.SetState( FALSE );
  }
void InvertImage()
{
  int  i, j;

  for( i=0; i<=HSize; i++ )
    for( j=0; j<=VSize; j++ )
          Image[i][j] ^= 0x0F;
  PaintGrid();
}
void monochrome()
{
  int  i, j;
  if( InvBtn.GetState() )
  {
     for( i=0; i<=HSize; i++ )
          for( j=0; j<=VSize; j++ )
            Image[i][j] &= 0x0F;
    PaintGrid();
  }
  if( MonoBtn.GetState() )
  {
    getpalette( &OrgPalette );
     for( i=1; i<=15; i++ )
          setpalette( i, OrgPalette.colors[ ActiveColor] );
  }else setallpalette( &OrgPalette );
  delay( 100 );
}
int  ReadSize()
{
  char Ch, TempStr[4] = "";
  int  Done = FALSE;
  settextjustify( LEFT_TEXT, TOP_TEXT );
  settextstyle( DEFAULT_FONT,HORIZ_DIR, 1 );
  setviewport( 120, 330, getmaxx(), 340, TRUE );
  while( !Done )
  {
    clearviewport();
    gprintxy( 1, 1, "Size (10..32): %s", TempStr );
    Ch = getch();
    if( Ch == 0x0D ) Done = TRUE;
    else
     if( ( Ch >= '0' ) && ( Ch <= '9' ) )
          strncat( TempStr, &Ch, 1 );
  }
```

```
  setviewport( 120, 330, getmaxx(), 340, TRUE );
  while( !Done )
  {
    clearviewport();
    gprintxy( 1, 1, "Size (10..32): %s", TempStr );
    Ch = getch();
    if( Ch == 0x0D ) Done = TRUE;
    else
     if( ( Ch >= '0' ) && ( Ch <= '9' ) )
           strncat( TempStr, &Ch, 1 );
  }
  clearviewport();
  setviewport( 0, 0, getmaxx(), getmaxy(), TRUE );
  return( atoi( TempStr ) );
}
void SetWidth()
{
  int  Result = ReadSize();
  if( ( Result < 10 ) || ( Result > 32 ) ) Beep();
  else
   {
    HSize = Result-1;
    PaintGrid();
   }
  WideBtn.SetState( FALSE );
}
void SetHeight()
{
  int  Result = ReadSize();
  if( ( Result < 10 ) || ( Result > 32 ) ) Beep();
  else
  {
    VSize = Result-1;
    PaintGrid();
  }
  HighBtn.SetState( FALSE );
}
char* ReadName( char* Note )
 {
  char  Ch, TempStr[20] = "";
  int   Done = FALSE;
  setcolor( WHITE );
  settextjustify( LEFT_TEXT, TOP_TEXT );
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 ),
  setviewport( 120, 330, getmaxx(), 340, TRUE);
  while( !Done )
   {
   clearviewport();
   gprintxy( 1, 1, "%s%s", Note, TempStr );
   Ch = getch();
   if( Ch == 0x0D ) Done = TRUE;
   else
    if( ( ( Ch >= '0' ) && ( Ch <= '9' ) ) ||
        ( ( Ch >= 'A' ) && ( Ch <= 'Z' ) ) ||
        ( ( Ch >= 'a' ) && ( Ch <= 'z' ) ) ||
         ( Ch == '.' ) ) strncat( TempStr, &Ch, 1);
   }
  clearviewport();
  setviewport( 0, 0, getmaxx(), getmaxy(), TRUE );
  return( TempStr );
}
void ReportError( char* ErrMsg )
{
  settextjustify( LEFT_TEXT, TOP_TEXT );
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  setviewport( 120, 330, getmaxx(), 350, TRUE );
  clearviewport();
  setcolor( LIGHTRED );
  outtextxy( 1, 1, ErrMsg );
  setcolor( WHITE );
  getch();
  clearviewport();
  setviewport( 0, 0, getmaxx(), getmaxy(), TRUE );
```

```
    return;
  }
  outtextxy( 120, 330, FileName );
  if( ( handle = open( FileName, O_CREAT | O_TRUNC | O_BINARY, S_IREAD | S_IWRITE ) ) == -1 )
  {
  ReportError( "no se puede abrir el archivo" );
  return;
  }
  write( handle, &HSize, 1 );
  write( handle, &VSize, 1 );
CFlag = IMonoBtn.GetState();
if( CFlag )
 {
  CFlag = FALSE;
  for( i=0; i<=HSize; i++ )
    for( j=0; j<=VSize; j++ )
        if( Image[i][j] )
          if( !C1 ) C1 = Image[i][j];
          else
          if( C1 != Image[i][j] )
            CFlag = TRUE;
}
if( CFlag )
{                                  // Imagen multicolor
  write( handle, &C2, 1 );         // Bandera multicolor
  for( i=0; i<=HSize; i++ )
  {
    C2 = k = 0;
    for( j=0; j<=VSize; j++ )
    {
          k++;
          C2 <<= 4;
          C2 += Image[i][j];
          if( k==2 )
          {
            k = 0;
            write( handle, &C2, 1 );
          }
    }
    if( k )
     {
          C2 <<= 4;
          write( handle, &C2, 1 );
          }
  }
}
else
{                                  // Imagen monocromatico
  write( handle, &ActiveColor, 1 );        // palette color
  for( i=0; i<=HSize; i++ )
    for( j=0; j<=3; j++ )
          if( j*8 <= VSize )
          {
            C2 = 0;
            for( k=0; k<=7; k++ ) C2 <<= 1;
            if( j*8+k <= VSize )
              if( Image[i][j*8+k] ) C2++;
            write( handle, &C2, 1 );
          }
}
close( handle );
SaveBtn.SetState( FALSE );
}
void ReadImage()
{
  unsigned int C1 = 0, C2 = 0;
  int    i, j, k, handle;
  char*  FileName;
  strcpy ( FileName, ReadName( "Read Image: " ) );
  if( FileName == "" )
  {
    Beep();
    ReadBtn.SetState( FALSE );
```

```
}
void ReadImage()
{
 unsigned int  C1 = 0, C2 = 0;
 int     i, j, k, handle;
 char*  FileName;
 strcpy  ( FileName, ReadName( "Read Image: " ) );
 if( FileName == "" )
 {
   Beep();
   ReadBtn.SetState( FALSE );
   return;
 }
 outtextxy( 120, 330, FileName );
 if( ( handle =
  open( FileName, O_RDONLY | O_BINARY ) ) == -1 )
 {
    ReportError( "No se puede abrir el archivo" );
    return;
 }
 read( handle, &HSize, 1 );
 read( handle, &VSize, 1 );
 read( handle, &C1, 1 );
 if( C1 == 0xFF )
  {                                  // Imagen multicolor
    for( i=0; i<=HSize; i++ )
            for( j=0; j<=VSize/2; j++ )
            {
              read( handle, &C2, 1 );
              Image[i][j*2] = C2 >> 4;
              if( VSize >= j*2+1 )
                Image[i][j*2+1] = C2 & 0x0F;
            }
  }
  else
  {                                  // Imagen monocromatico
  for( i=0; i<=HSize; i++ )
    for( j=0; j<=3; j++ )
            if( j*8 <= VSize )
            {
              read( handle, &C2, 1 );
              for( k=0; k<=7; k++ )
              {
                      if( j*8+k <= VSize )
                       if( C2 & 0x80 ) Image[i][j*8+k] = C1;
                        else           Image[i][j*8+k] = 0;
                      C2 <<= 1;
              }
            }
 }
 close( handle );
  PaintGrid();
  ReadBtn.SetState( FALSE );
}
main()
{
  Mstatus  Mevent;
  int     Exit = FALSE;
  GDriver = DETECT;
  initgraph( &GDriver, &GMode, "C:\\LENGUAJE\\TC\\BGI");
  GError = graphresult();
  if( GError != grOk )
  {
    printf( "error Grafico: &s\n",
            grapherrormsg( GError ) );
    printf( "Program abortado..." );
    exit(1);
  }
if( !gmouse.Mreset() ) exit(1);
cleardevice();

//////////////////////////////////////////////////////////////////
//
```

```
MonoBtn.SetAll( SQUARE, 120,  80, 80, 20, LIGHTGREEN, "Mono"  );
HighBtn.SetAll( SQUARE, 120, 130, 80, 20, YELLOW,    "High"  );
WideBtn.SetAll( SQUARE, 120, 160, 80, 20, YELLOW,    "Wide"  );
SaveBtn.SetAll( SQUARE, 120, 210, 80, 20, LIGHTCYAN, "Save"  );
ReadBtn.SetAll( SQUARE, 120, 200, 80, 20, LIGHTCYAN, "Read"  );
NoteColor();
RButton[ActiveColor].SetState( TRUE );
for( i=0; i<=HSize; i++ )
  for( j=0; j<=VSize; j++ )
    Image[i][j] = 0;
paintGrid();

////////////////////////////////////////////////////////////////
//
//              Empiezo de la edición del programa
//
////////////////////////////////////////////////////////////////

while( !Exit )
{
  Mevent = gmouse.Mpos();
  Exit = ExtBtn.ButtonHit();
  if( mevent.button,status & 0x07 )
  do
  {
    if( Mevent.xaxis ( 120 )
    {
          for( i=0; i<=15; i++ )
            if( RButton[i].ButtonHit() )
            {
              gmouse.Mshow( FALSE );
              for( j=0; j<=15; j++ )
                    RButton[j].SetState( FALSE );
              RButton[i].SetState( TRUE );
              ActiveColor = i;
              NoteColor();
              gmouse.Mshow( TRUE );
            }
    }
    else
    if( Mevent.xaxis < 250 )
    {
      if( ClrBtn.ButtonHit() )  ClearImage();
      if( InvBtn.ButtonHit() )  InvertImage();
      if( MonoBtn.ButtonHit() ) Monochrome();
      if( HighBtn.ButtonHit() ) SetHeight();
      if( WideBtn.ButtonHit() ) SetWidth();
      if( SaveBtn.ButtonHit() ) SaveImage
      if( ReadBtn.ButtonHit() ) ReadImage
    }
    else
    {

    FixColor = ActiveColor;
    if( Mevent.button_status & 0x02 )
      PixColor = BLACK;
    i = ( Mevent.xaxis - 300 ) / 10;
    j = Mevent.yaxis / 10;
    if( ( i >= 0 ) && ( i <= HSize ) &&
          ( j >= 0 ) && ( j <= VSize ) )
      if( Image[i][j] != PixColor )
      {
            gmouse.Mshow( FALSE );
            Image[i][j] = PixColor;
            FillSquare( i*10+300,  j*10,
                        INTERLEAVE_FILL, PixColor );
            putpixel( 250+i, 10+j, PixColor );
            gmouse.Mshow( TRUE );
      }
    }
    Mevent = gmouse.Mpos();          // busca el estado del boton del mouse
}                                                  // continua si unicamente el boton esta abajo
```

```
            gmouse.Mshow( FALSE );
            Image[i][j] = PixColor;
            FillSquare( i*10+300,  j*10,
                        INTERLEAVE_FILL, PixColor );
            putpixel( 250+i, 10+j, PixColor );
            gmouse.Mshow( TRUE );
        }
    }
    Mevent = gmouse.Mpos();          // busca el estado del boton del mouse
}                                                  // continua si unicamente el boton esta abajo

    while( Mevent.button_status );
}
    closegraph();                  // almacena en modo texto y ...
    tmouse_Mreset();               // resetea el mouse para la operación de texto
}


//////////////////////////////////////////////////////////////
//
//              INCONTESST.CPP
//          PROGRAMA DEMO OBJETO DE ICONOS
//
//////////////////////////////////////////////////////////////


#include <fcntl.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graphics.h>

#include "mouse.i"
#include <gprint.i>
#include <gcontrol.i>
class Icon
{
    void  *ImgPtr;
    int   X, Y, XOfs, YOfs, HSize, VSize, State;
  public:
    Icon();
    ~Icon();
    void Create( char* FileName );
    void Show( int Status );
    void MoveTo( int xPos, int YPos );
    void Drag();
    int  MouseSelect();
};
Icon::Icon()
{
  HSize = VSize = XOfs = YOfs = X = Y = 0;
}
Icon::~Icon()
{
  if( State ) Show( FALSE );
  free( ImgPtr );
}
void Icon::Create( char* FileName )
{
  unsigned  int  C1 = 0, C2 = 0 ;
  int  i, j, k, handle;
  if( ( handle =
          open( FileName, O_RDONLY | O_BINARY )) == -1 )
  {
    gprintxy( 200, 200 ,
                "Cannot open as input file", FileName);
    getch();
    return;
```

```
                if( VSize >= j*2+1 )
                    putpixel( i, j*2+1, C2 & 0x0F );
                }
        }
        else
        {                               // Imagen monocromatica
          for( i=0; i<=HSize; i++ )
                for( j=0; j<=3; j++ )
                  if( j*8 <= VSize )
                  {
                        read( handle, &C2, 1 );
                        for( k=0; k<=7; k++ )
                        {
                          if( j*8+k <= VSize )
                                if( C2 & 0x80 ) putpixel( i, j*8+k, C1 );
                                else            putpixel( i, j*8+k, 0 );
                          C2 <<= 1;
                        }
                  };
        }
        close( handle );
        ImgPtr = (void *)malloc(sizeof(unsigned) );
        imagesize( 0, 0, HSize, VSize );
        getimage( 0, 0, HSize, VSize, ImgPtr );
        putimage( 0, 0, ImgPtr, XOR_PUT );
        State = FALSE;
}
void Icon::Show( int Status )
{
  if( Status != State )
  {
    gmouse.Mshow( FALSE );
    putimage( X, Y, ImgPtr, XOR_PUT );
    State = !State;
    gmouse.Mshow( TRUE );
  }
}
void Icon::MoveTo( int XPos, int YPos )
{
  if( State ) putimage( X, Y, ImgPtr, XOR_PUT);
  X = XPos;
  Y = YPos;
  putimage( X, Y, ImgPtr, XOR_PUT );
  State = TRUE;
}
int  Icon::MouseSelect()
{
 Mstatus  Mevent = gmouse.Mpos();
  if( ( Mevent.xaxis >= X ) && ( Mevent.xaxis <= X+HSize) &&
    ( Mevent.yaxis >= Y ) && ( Mevent.yaxis <= Y+VSize) )
  {
    XOfs = MouseX - X;              // posición relativa de icono
    YOfs = MouseY - Y;              // Para el sitio del cursor del mouse
    Drag();
    return( TRUE );
  }
  return( FALSE );
}
void Icon::Drag()
{
  Mstatus  Mevent;
  int      XPos, YPos, XOld = -1, YOld = -1, Done = FALSE;
  Mevent = gmouse.Mreleased( ButtonL );
  Show( FALSE );
  gmouse.Set_Cursor(GLOVE);
  do
  {
    Mevent = gmouse.Mpos();
    XPos = Mevent.xaxis;
    YPos = Mevent.yaxis;
    if( ( XPos != XOld ) || ( YPos != YOld ))
    {
          gmouse.Mshow(FALSE);
```

```
  Mevent = gmouse.Mreleased( ButtonL );
  Show( FALSE );
  gmouse.Set_Cursor(GLOVE);
  do
  {
     Mevent = gmouse.Mpos();
     XPos = Mevent.xaxis;
     YPos = Mevent.yaxis;
     if( ( XPos != XOld ) || ( YPos != YOld ))
     {
            gmouse.Mshow(FALSE);
            putimage( XPos-XOfs, YPos-YOfs, ImgPtr, XOR_PUT );
            putimage( XOld-XOfs, YOld-YOfs, ImgPtr, XOR_PUT );
            gmouse.Mshow( TRUE );
            XOld = XPos;
            YOld = YPos;
     }
     Mevent = gmouse.Mreleased( ButtonL );
     Done = Mevent.button_count;
  }
  while( !Done );
  X = XPos-XOfs;                  // actualización de las coordenadas del objeto
  Y = YPos-YOfs;                  // hacia la posición final
  gmouse.Set_Cursor(ARROW);       // restaurando el cursor del mouse
}
  //======== fin de la definición del objecto Icono =========//
void main()
{
  int     i,(far )GDriver = DETECT, (far )GMode, GError;
  Mstatus  Mevent;
  Button   ExitButton;
  Icon     PixImage[4];
  initgraph( &GDriver = DETECT, &GMode, "D:\\LENGUAJE\\TC\\BGI");
  GError = graphresult();
  if( GError != grOk )
  {
    printf( "Error Grafico: %s\n",
                grapherrormsg( GError ) );
    printf( "Programa abortado..." );
    exit(1);
  }
  if( !gmouse.Mreset() ) exit(1);
  cleardevice();
  PixImage[0].Create( "FILEFLDR.ICO");
  PixImage[1].Create( "OM.ICO" );
  PixImage[2].Create( "QUESTION.ICO" );
  PixImage[3].Create( "RAINBOW3.ICO" );
  for( i=0; i<=3; i++ )
     PixImage[i].MoveTo( 40 * i + 60, 200 );
  ExitButton.SetAll( ROUNDED, getmaxx()-80, 1,
                     80, 20, LIGHTRED, "Salida" );
  gmouse.Mshow( TRUE );

            //=============================================
            //   Comienzo del programa de demostración
            //=============================================

while( !ExitButton.ButtonHit() )
{
   Mevent = gmouse.Mpressed( ButtonL );
   if( Mevent.button_count )
     for( i=0; i<=3; i++ )
            if( PixImage[i].MouseSelect() ) /* ??? */;
}                   //=====================================//
  closegraph();                 // restaura modo texto y ...      //
  tmouse.Mreset();              // reinicializa el mouse para texto //
}                   //=====================================//
```

```
#define lower (x, y)  (x < y) ? x : y
#define upper (x, y)  (x > y) ? x : y
#define ButtonL    0
#define buttonR    1
#define ButtonM    2
#define SOFTWARE   0                    // tipo del cursor
#define HARDWARE   1
#define FALSE      0
#define TRUE       1
#define OFF        0
#define ON         1
union REGS inreg, outreg;                    // registros tipo statfe
typedef struct  {  int present,          // TRUE si present
                   buttons;                  // # of buttons
                } Mresult;
typedef struct
  { int button_status,                  // los bits 0-2 son ON si el boton esta abajo
         button_count,                       // a la vez que el boton le fue dado un click
         xaxis, yaxis;                       // con el mouse
  } Mstatus;
typedef struct
  {  int x_count,
         y_count;
  } Mmovement;
typedef struct
  {  unsigned  flag,
               button,
               xaxis, yaxis;
  } mouse_event;
typedef struct              // descripción grafica del mouse
  {  unsigned  int
         ScreenMask[16],
         CursorMask[16],
         xkey, ykey;
  } g_cursor;
static  g_cursor  ARROW =
        {  0x1FFF, 0x0FFF, 0x07FF, 0x03FF,
           0x01FF, 0x00FF, 0x007F, 0x003F,
           0x001F, 0x003F, 0x01FF, 0x01FF,
           0xE0FF, 0x00FF, 0xE8FF, 0xF8FF,
                                                                  // cursor
           0x0000,  0x4000, 0x6000,  0x7000,
           0x7800,  0x7CC0, 0x7E00,  0x7F00,
           0x7F80,  0x7C00, 0x4C00,  0x0600,
           0x0600,  0x0300, 0x0300,  0x0000,
           0x0001,  0x0001 };              // xkey, ykey
static  g_cursor CHECK =                // pantalla
        {
           0xFFF0,  0xFFE0, 0xFFC0,  0xFF81,
           0xFF03,  0x0607, 0x000F,  0x001F,
           0x803F,  0xC07F, 0xE0FF,  0xF1FF,
           0xFFFF,  0xFFFF, 0xFFFF,  0xFFFF,
                                                  // cursor
           0x0000,  0x0006, 0x000C,  0x0018,
           0x0030,  0x0060, 0x70C0,  0x3980,
           0x1F00,  0x0E00, 0x0400,  0x0000,
           0x0000,  0x0000, 0x0000,  0x0000,
           0x0000,  0x000A };              // xkey, ykey

static  g_cursor CROSS =                // pantalla
        {
           0xF01F,  0xE00F, 0xC007,  0x8003,
           0x0441,  0x0C61, 0x0381,  0x0381,
           0x0381,  0x0C61, 0x0441,  0x8003,
           0xC007,  0xE00F, 0xF01F,  0xFFFF,
                                                  // cursor
           0x0000,  0x07C0, 0x0920,  0x1110,
           0x2108,  0x4004, 0x4004,  0x783C,
           0x4004,  0x4004, 0x2108,  0x1110,
           0x0920,  0x07C0, 0x0000,  0x0000,
           0x0007,  0x0007 };              // xkey, ykey

static  g_cursor GLOVE =                // pantalla
```

```
            0x0441,  0x0C61, 0x0381,  0x0381,
            0x0381,  0x0C61, 0x0441,  0x8003,
            0xC007,  0xE00F, 0xF01F,  0xFFFF,
                                                    // cursor
            0x0000,  0x07C0, 0x0920,  0x1110,
            0x2108,  0x4004, 0x4004,  0x783C,
            0x4004,  0x4004, 0x2108,  0x1110,
            0x0920,  0x07C0, 0x0000,  0x0000,
            0x0007,  0x0007 };              // xkey, ykey

static  g_cursor GLOVE =                    // pantalla
          {
            0xF3FF,  0xE1FF, 0xE1FF,  0xE1FF,
            0xE1FF,  0xE049, 0xE000,  0x8000,
            0x0000,  0x0000, 0x07FC,  0x07F8,
            0x9FF9,  0x8FF1, 0xC003,  0xE007,
                                                  // cursor
            0x0C00,  0x1200, 0x1200,  0x1200,
            0x1200,  0x12B6, 0x1249,  0x7249,
            0x9249,  0x9001, 0x0001,  0x8001,
            0x4002,  0x4002, 0x2004,  0x1FF8,
            0x0004,  0x0000 };              // xkey, ykey

static  g_cursor IBEAM =                    // pantalla
          {
            0xF39F,  0xFD7F, 0xFEFF,  0xFEFF,
            0xFEFF,  0xFEFF, 0xFEFF,  0xFEFF,
            0xFEFF,  0xFEFF, 0xFEFF,  0xFEFF,
            0xFFFF,  0xFEFF, 0xFD7F,  0xF39F,
                                                  // cursor
            0x0C60,  0x0280, 0x0100,  0x0100,
            0x0100,  0x0100, 0x0100,  0x0100,
            0x0100,  0x0100, 0x0100,  0x0100,
            0x0100,  0x0100, 0x0280,  0x0C60,
            0x0007,  0x0008 };              // xkey, ykey
class Mouse
{
  int    Mview;                     // estado del cursor del mouse
  protected:
    Mouse();                        // constructor
    ~Mouse();                                   // destructor
  public:
    static mouse_event far *Mevents;
    Mmovement *Mmotion();
    Mresult  *Mreset();
    Mstatus   Mpos();
    Mstatus   Mpressed( int button );
    Mstatus   Mreleased( int button );
    void Mshow( int showstat );
    void Mmoveto( int xaxis, int yaxis );
    void Mxlimit( int min_x, int max_x );
    void Mylimit( int min_y, int max_y );
    void Mmove_ratio( int xsize, int ysize );
    void Mspeed( int speed );
    void Mconceal( int left, int top,
                   int right, int bottom );
};
class GMouse : public Mouse
{
  private:
    void set_cursor( int xaxis,
                     int yaxis,
                     unsigned mask_Seg,
                     unsigned mask_Ofs );
  public:
    void Set_Cursor( g_cursor ThisCursor );
    void Mlightpen( int set );
};
class TMouse : public Mouse
{
  public:
                                                  // figura del cursor
    void Set_Cursor( int cursor_type,
```

```
Mouse::Mouse()
{
}

Mouse::~Mouse()
{
}

//////////////////////////////////////////////////////////////////////
//
//      resetea el estado default, retornando el pintero al la estructura
//      Mresult indicando si el mouse instalado y si presenta número de
//      botones siempre que sea llamado durante la Inicialización
//
//////////////////////////////////////////////////////////////////////

Mresult *Mouse::Mreset()
{
  static Mresult  m;
  Mview = OFF;
  inreg.x.ax = 0;                    // mouse función 0 //
  call_mouse;
  m.present = outreg.x.ax;
  m.buttons = outreg.x.bx;
  if( m.present ) Mshow( TRUE );
  return ( &m );
}
void Mouse::Mshow( int showstat )
{
  if( showstat )
  {
    inreg.x.ax = 1;              // función 1  del mouse//
    if( !Mview ) call_mouse;      // muestra el cursor del mouse//
    Mview = ON;
  }
  else
  {
    inreg.x.ax = 2;              // función 2 del mouse //
    if( Mview ) call_mouse;       // oculta el cursor del mouse//
    Mview = OFF;
  }
}
Mstatus Mouse::Mpos()
{                                  // retorna el puntero a la
                                                        // structura Mstatus con la
                                                        // posición del cursor del
                                                        // mouse y el estado del botton
  static Mstatus  m;
  inreg.x.ax = 3;                 // función 3 del mouse
  call_mouse;
  m.button_status = outreg.x.bx;     // estado del boton
  m.xaxis = outreg.x.cx;             // coordenadas xaxis
  m.yaxis = outreg.x.di;             // coordenadas yaxis
  return(m);
}
void Mouse::Mmoveto( int xaxis, int yaxis )
{                                // el cursor del mouse a la nueva posición
  inreg.x.ax = 4;                  // function 4 del mouse
  inreg.x.cx = xaxis;
  inreg.x.dx = yaxis;
  call_mouse;
}
Mstatus Mouse::Mpressed( int button )
{
  static Mstatus  m;
  inreg.x.ax = 5;                  // función 5 del mouse
  inreg.x.bx = button;              // requerida para el boton
  call_mouse;
  m.button_status = outreg.x.ax;
  m.button_count = outreg.x.bx;
  m.xaxis = outreg.x.cx;
  m.yaxis = outreg.x.dx;
  return (m);
```

```
Mstatus Mouse::Mpressed( int button )
{
  static Mstatus  m;
  inreg.x.ax = 5;                  // función 5 del mouse
  inreg.x.bx = button;              // requerida para el boton
  call_mouse;
  m.button_status = outreg.x.ax;
  m.button_count = outreg.x.bx;
  m.xaxis = outreg.x.cx;
  m.yaxis = outreg.x.dx;
  return (m);
}
Mstatus Mouse::Mreleased( int button )
{                                  // retorna información acerca del boton //
  static Mstatus  m;
  inreg.x.ax = 6;                  // función 6 del mouse
  inreg.x.bx = button;              // requerida para el boton
  call_mouse;
  m.button_status = outreg.x.ax;
  m.button_count = outreg.x.bx;
  m.xaxis = outreg.x.cx;
  m.yaxis = outreg.x.dx;
  return (m);
}
                                                  // da un rango horizontal min/max para el cursor
                                                  // Mueve el cursor dentro de un rango cuando es
                                                  // llamado. Cambiando los valores de min_x y max_x

void Mouse::Mxlimit( int min_x, int max_x )
{
  inreg.x.ax = 7;                  // function 7 del mouse
  inreg.x.cx = min_x;
  inreg.x.dx = max_x;
  call_mouse;
}
void Mouse::Mylimit( int min_y, int max_y )
{
  inreg.x.ax = 8;                  // function 8 del mouse
  inreg.x.cx = min_y;
  inreg.x.dx = max_y;
  call_mouse;
}
void GMouse::set_cursor( int xaxis, int yaxis,
                         unsigned mask_Seg,
                         unsigned mask_Ofs )
{                                  // da la figura grafica del cursor
  struct SREGS  seg;

  inreg.x.ax = 9;                  // function 9 del mouse
  inreg.x.bx = xaxis;
  inreg.x.cx = yaxis;
  inreg.x.dx = mask_Ofs;
  seg.es     = mask_Seg;
  int86x( 0x33, &inreg, &outreg, &seg );
}
                                                  // da el tipo de cursor, 0 = software, 1 = hardware
void TMouse::Set_Cursor( int cursor_type,
                         unsigned s_start,
                         unsigned s_stop  )
{
  inreg.x.ax = 10;                 // función 10 del mouse
  inreg.x.bx = cursor_type;
  inreg.x.cx = s_start;
  inreg.x.dx = s_stop;
  call_mouse;
}
Mmovement* Mouse::Mmotion()
{                                  // reporta el movimiento del curso desde la ultima llamada
  static Mmovement  m;
  inreg.x.ax = 11;                 // función 11 del mouse
  call_mouse;
  m.x_count = _CX;
  m.y_count = _DX;
```

```
    else  inreg.x.ax = 14;          // función 14 OFF
  call_mouse;
}
void Mouse::Mmove_ratio( int xsize, int ysize )
{                                   // mueve el pixel con un radio de R/8
  inreg.x.ax = 15;                  // Default 16 vert                    g horiz //
  inreg.x.cx = xsize;               //          8 horiz
  inreg.x.dx = ysize;
  call_mouse;
}
void Mouse::Mconceal( int left, int top, int right, int bottom )
{
  inreg.x.ax = 16;
  inreg.x.cx = left;
  inreg.x.dx = top;
  inreg.x.si = right;
  inreg.x.di = bottom;
  call_mouse;
}
void Mouse::Mspeed( int speed )
{
  inreg.x.ax = 19;
  inreg.x.dx = speed;
  call_mouse;
}
void GMouse::Set_Cursor( g_cursor ThisCursor )
{
  set_cursor( ThisCursor.xkey,
              ThisCursor.ykey,
              _DS,
              (unsigned) ThisCursor.ScreenMask );
}
                                                  // muestra las definiciones para usarse en las aplicaciones
GMouse  gmouse;
TMouse  tmouse;
```

```
////////////////////////////////////////////////////////////////////
//
//                      MOUSEPTR.CPP
//                    Demo para objeto mouse
//          y la utilidad para crear el cursor del mouse
//
////////////////////////////////////////////////////////////////////
```

```
#ifdef _TINY_
#error demo grafico que podria no correr en el modelo TINY
#endif
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include <string.h>
#include <mouse.h>
Mstatus  Position;
g_cursor  NewCursor;
Int      Buttons, XIndex, YIndex, HotSpotX = 0, HotSpotY = 0, HotSpotSelect, Screen[16] [16], Cursor[16] [16];
int TPos( int TP, int Low, int High )
{
  return( ( TP >= Low ) && ( TP <= High ) );
}
void BoxItem( int x, int y, int w, int h, char* text )
{
  settextjustify( CENTER_TEXT, CENTER_TEXT);
  rectangle( x, y, x+w, y+h );
  outtextxy( x+(w/2), y+(h/2), text );
}
void FillSquare( int x1, int y1, int x2, int y2, int FillStyle,  int Color )
```

```
int TPos( int TP, int Low, int High )
{
  return( ( TP >= Low ) && ( TP <= High ) );
}
void BoxItem( int x, int y, int w, int h, char* text )
{
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  rectangle( x, y, x+w, y+h );
  outtextxy( x+(w/2), y+(h/2), text );
}
void FillSquare( int x1, int y1, int x2, int y2, int FillStyle,  int Color )
{
    int  outline[10];
    outline[0] = outline[6] = outline[8] = x1;
    outline[2] = outline[4] = x2;
    outline[1] = outline[3] = outline[9] = y1;
    outline[5] = outline[7] = y2;
    setfillstyle( FillStyle, Color );
    fillpoly( 5, outline );
}
void EraseSquare( int x1, int y1, int x2, int y2 )
{
    FillSquare( x1, y1, x2, y2, EMPTY_FILL, 0 );
}
void Beep()
{
 sound( 220 ); delay( 100 ); nosound();
                delay(  50 );
 sound( 400 ); delay( 100 ); nosound();
}
void MakeCursor()
{
 int  i, j;
 unsigned int TBit;
 NewCursor.xkey = HotSpotX;
 NewCursor.ykey = HotSpotY;
  for( i=0; i<=15; i++ )
  {
   NewCursor.ScreenMask[i] = 0x0000;
   NewCursor.CursorMask[i] = 0x0000;
    for( j=0; j<=15; j++ )
    {
          NewCursor.CursorMask[i] <<= 1;
           if( Cursor[j][i] ) NewCursor.CursorMask[i]++;
           NewCursor.ScreenMask[i] <<= 1;
           if( Screen[j][i] ) NewCursor.ScreenMask[i]++;
    }
  }
}
void UseNewCursor()
{
  MakeCursor();
  gmouse.Mshow( FALSE );
  gmouse.Set_Cursor( NewCursor );
  gmouse.Mshow( TRUE );
}
void PaintScreen( int X, int Y )
{
  int Color = WHITE;
  if  ( ( X == HotSpotX ) && ( Y == HotSpotY ) )
    Color = LIGHTRED;
  gmouse.Mshow( FALSE );
  if( Screen[X][Y] )
    FillSquare( X*15+18, Y*15+33, X*15+27, Y*15+92, SOLID_FILL, Color );
  else
  {
    EraseSquare( X*15+15, Y*15+30, X*15+30, Y*15+45 );
    FillSquare(  X*15+15, Y*15+30, X*15+30, Y*15+45, CLOSE_DOT_FILL, Color );
  }
  gmouse.Mshow( TRUE );
  setcolor( WHITE );
  }
void PaintCursor( int X, int Y )
```

```
                    (X+2) * 15 + 369, (Y+3) * 15, CLOSE_DOT_FILL, Color );
    gmouse.Mshow( TRUE );
    setcolor( WHITE );
}
void HotSpotComplete()
{
  PaintCursor( HotSpotX, HotSpotY );
  PaintScreen( HotSpotX, HotSpotY );
  gmouse.Mshow( FALSE );
  HotSpotSelect = FALSE;
  setcolor( WHITE );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  BoxItem ( 265, 320, 110, 20, "Set Hotspot" );
  gmouse.Mshow( TRUE );
}
void SetHotSpot()
{
  int X, Y;
  X = HotSpotX;
  Y = HotSpotY;
  HotSpotX = -1;
  HotSpotY = -1;
  PaintCursor( X, Y );
  PaintScreen( X, Y );
  gmouse.Mshow( FALSE );
  HotSpotSelect = TRUE;
  setcolor( RED );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  BoxItem( 265, 320, 110, 20, "set Hotspot" );
  setcolor( WHITE );
  gmouse.Mshow( TRUE );
}
void ScreenLayout()
{
  int i, j;
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  outtextxy( 135, 20, "Screen Mask" );
  outtextxy( 504, 20, "Cursor Mask" );
  HotSpotComplete();
                                          //Detalle de pantalla
  BoxItem(  15, 280,  60, 20, "Clear" );
  BoxItem(  85, 280,  60, 20, "Invert" );
  BoxItem( 155, 280, 140, 20, "Make from Cursor");
                                          // Detalle del cursor
  BoxItem( 564, 280,  60, 20, "Clear" );
  BoxItem( 494, 280,  60, 20, "Invert" );
  BoxItem( 344, 280, 140, 20, "Copy to Scree" );
                                          //opcion de controles
  BoxItem(  15, 320, 110, 20, "Use Pointer" );
  BoxItem( 140, 320, 110, 20, "Error Pointer" );
  BoxItem( 265, 320, 110, 20, "Set Hotspot" );
  BoxItem( 390, 320, 110, 20, "Save Pointer" );
  BoxItem( 515, 320, 110, 20, "Exit" );
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
    {
        Screen[i][j] = FALSE;
        PaintScreen( i, j );
        Cursor[i][j] = FALSE;
        PaintCursor( i, j );
    }
}
void ClearScreen()
{
  int  i, j;
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
        if( Screen[i][j] )
        {
          Screen[i][j] = FALSE;
          PaintScreen( i, j );
        }
}
```

```
void ClearScreen()
{
  int  i, j;
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
          if( Screen[i][j] )
          {
            Screen[i][j] = FALSE;
            PaintScreen( i, j );
          }
}
void ClearCursor()
{
  int  i, j;
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
          if( Cursor[i][j] )
          {
            Cursor[i][j] = FALSE;
            PaintCursor( i, j );
          }
}
void InvertScreen()
{
  int  i, j;
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
    {
          if( Screen[i][j] ) Screen[i][j] = FALSE;
                        else Screen[i][j] = TRUE;
          PaintScreen( i, j );
    }
}
void InvertCursor()
{
  int i, j;
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
    {
          if( Cursor[i][j] ) Cursor[i][j] = FALSE;
                        else Cursor[i][j] = TRUE;
     PaintCursor( i, j );
    }
}
void ScreenSet( int xaxis, int yaxis )
{
  int x, y;
  x = ( xaxis / 15 ) - 1;
  y = ( yaxis / 15 ) - 2;
  if( HotSpotSelect )
  {
    HotSpotX = x;
    HotSpotY = y;
    HotSpotComplete();
  }
  else
  {
    if( Screen[x][y] ) Screen[x][y] = FALSE;
                  else Screen[x][y] = TRUE;
    PaintScreen( x, y );
  }
}
void CursorSet( int xaxis, int yaxis )
{
  int x, y;
  x = ( xaxis - 384 ) / 15;
  y = ( yaxis / 15 ) - 2;
  if( HotSpotSelect )
  {
    HotSpotX = x;
    HotSpotY = y;
    HotSpotComplete();
  }
```

```
    for( j=0; j<=15; j++ )
    {
            Screen[i][j] = Cursor[i][j];
            PaintScreen( i, j );
    }
}
void ScreenFromCursor()
{
  int i, j, x, y, Test;
  for( i=0; i<=15; i++ )
    for( j=0; j<=15; j++ )
    {
            Test = TRUE;
            for( x=-1; x<=1; x++ )
              for( y =-1; y<=1; y++ )
                if( ( TPos( i+x, 0, 15 ) ) &&
                        ( TPos( j+y, 0, 15 ) ) &&
                        ( Cursor[i+x][j+y] ) ) Test = FALSE;
            Screen[i][j] = Test;
            PaintScreen( i, j );
    }
}
int SavePointer()
{
  int  i, Done = FALSE;
  char Ch, CursorName[8]="", FileName[12]="";
  FILE *CF;
  strcpy( CursorName, "........" );
  setviewport( 269, 0, 369, 42, TRUE );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  gmouse.Set_Cursor( IBEAM );
  gmouse.Mmoveto( 277, 30 );
  i = 0;
  do
  {
    gmouse.Mshow( FALSE );
    clearviewport();
    setcolor( LIGHTRED );
    rectangle( 0, 0, 100, 40 );
    outtextxy( 50, 10, "Save As" );
    outtextxy( 50, 20, "File Name?" );
    outtextxy( 50, 30, CursorName);
    gmouse.Mmoveto( 277+i*8,30 );
    gmouse.Mshow( TRUE );
    Ch = getch();
    if( Ch == 0x0D ) Done = TRUE;
    else if( ( Ch == 0x08 ) && ( i > 1 ) ) i-=2;
    else if( i > 7 ) { Beep(); i-=1 ;}
    else CursorName[i] = Ch;
    i++;
  }
  while( !Done );
  gmouse.Set_Cursor( ARROW );
  gmouse.Mshow( FALSE );
  clearviewport();
  setviewport( 0, 0, getmaxx(), getmaxy(), TRUE );
  gmouse.Mshow( TRUE );
  for( i=7; i>=0; i-- )
   {
            if( CursorName[i] == '.' ) CursorName[i] = '\0';
            if( CursorName[i] == ' ' ) CursorName[i] = '\0';
   }
  if( strlen( CursorName ) == 0 ) { Beep(); return(0); }
  MakeCursor();
  strcpy( FileName, CursorName );
  strcat( FileName, ".CUR" );
  outtextxy( 320, 10, FileName );
  CF = fopen( FileName, "w" );
  fprintf( CF, "static  g_cursor %s = \n", CursorName );
  fprintf( CF,
            "           0x%04X, 0x%4X, 0x%04X, 0x%04X,\n",
            NewCursor.ScreenMask[0], NewCursor.ScreenMask[1],
            NewCursor.ScreenMask[2], NewCursor.ScreenMask[3] );
```

```
  if( strlen( CursorName ) == 0 ) { Beep(); return(0); }
  MakeCursor();
  strcpy( FileName, CursorName );
  strcat( FileName, ".CUR" );
  outtextxy( 320, 10, FileName );
  CF = fopen( FileName, "w" );
  fprintf( CF, "static g_cursor %s = \n", CursorName );
  fprintf( CF,
           "        0x%04X, 0x%4X, 0x%04X, 0x%04X,\n",
           NewCursor.ScreenMask[0], NewCursor.ScreenMask[1],
           NewCursor.ScreenMask[2], NewCursor.ScreenMask[3] );
  for( i=1; i<=3; i++ )
        fprintf( CF,
           "        0x%04X, 0x%04X, 0x%04X, 0x%04x,\n",
           NewCursor.ScreenMask[i*4],
           NewCursor.ScreenMask[i*4+1],
           NewCursor.ScreenMask[i*4+2],
           NewCursor.ScreenMask[i*4+3]);
  for( i=0; i<=3; i++ )
        fprintf( CF,
        "        0x%04X, 0x%04X, 0x%04X, 0x%04x,\n",
           NewCursor.ScreenMask[i*4],
           NewCursor.ScreenMask[i*4+1],
           NewCursor.ScreenMask[i*4+2],
           NewCursor.ScreenMask[i*4+3]);
  fprintf( CF,
           "        0x%04X, 0x%04X,\n",  NewCursor.xkey,  NewCursor.ykey );
  fprintf( CF, "\n" );
  fclose( CF );
  return( 1 );
}
main()
{
  int GDriver = DETECT, GMode, GError,
    Exit = FALSE, i, j;
  Mresult* Result;
  initgraph( &GDriver, &GMode, "C:\\LENGUAJE\\TC\\BGI");
  GError = graphresult();
  if( GError != grOk )
  {
    printf( "Error Grafico: %s\n",
            grapherrormsg(GError) );
    printf( "Programa abortado...\n" );
    exit(1);
  }
  cleardevice();
  ScreenLayout();
  Result = gmouse.Mreset();
  setwritemode( COPY_PUT );
  if( Result->present )
  {
    do
    {
        Position = gmouse.Mpressed( ButtonL );
        if( Position.button_count )
        {
          if( TPos( Position.yaxis, 30, 270 ) )
          {
            if( TPos( Position.xaxis,  15, 255 ))
                  ScreenSet( Position.xaxis,
                             Position.yaxis );
            if( TPos( Position.xaxis, 384, 624 ))
                  CursorSet( Position.xaxis,
                             Position.yaxis );
          } else
          if( TPos( Position.yaxis, 280, 300 ) )
                                          // pantalla o comandos del cursor
  {
   if(TPos( Position.xaxis,  15,  75 ) )
          ClearScreen(); else
   if(TPos( Position.xaxis,  85,  145 ) )
          InvertScreen(); else
   if(TPos( Position.xaxis,  155,  295 ) )
```

```
            UseNewCursor(); else
    if(TPos( Position.xaxis,  140,  250 ) )
            gmouse.Set_Cursor(ARROW); else
    if(TPos( Position.xaxis,  265,  375 ) )
            SetHotSpot(); else
    if(TPos( Position.xaxis,  390,  500 ) )
            SavePointer(); else
    if(TPos( Position.xaxis,  515,  625 ) )
            Exit = TRUE;
     ) ) )
      while( !Exit );
    }
    tmouse.Mreset();
    tmouse.Set_Cursor(HARDWARE, 11,12);
    Beep();
}
```

CONCLUSIONES

Este trabajo de tesis ha sido desarrollado con el fin de proponer la aplicación de una de las más actuales areas de las ciencias de la computación, la **NEUROCOMPUTACION**, que sin lugar a dudas se presenta como una nueva herramienta para la resolución de problemas, y, como ya se ha mencionado, viene a complementar a las demás técnicas de programación, no a sustituírlas. Es importante resaltar el hecho de que las redes neurales son sólo una herramienta más, que deberán usarse en conjunto con las técnicas tradicionales (análisis y diseño estructurados, sistemas de bases de datos relacionales, programación estructurada, etc.), en aplicaciones en donde dichas técnicas tengan un alcance limitado o sea totalmente impráctico su uso. También es importante mencionar que las redes neurales no pueden aplicarse en donde sea y para cualquier propósito, ya que tienen sus propias limitaciones.

Por otro lado, a diferencia de las redes neurales, la **Programación Orientada a Objetos** (OOP), es una nueva herramienta que sí viene a sustituír a la tradicional técnica de la programación estructurada. La OOP, como se explicó en el capítulo 2, es un enfoque muy diferente al tipo de estructuras de datos clásicos de la programación estructurada, esta técnica construye las estructuras de datos necesarias para manejar la información con una concepción semejante a como lo hace la mente humana, es decir, de una manera más natural; maneja la información como entes (objetos) de cierto tipo (clase), a los que sólo se les puede aplicar un estricto conjunto de operaciones específicas (métodos de la clase en cuestión), a partir de los cuales se pueden especificar o generalizar otros entes de un tipo similar pero características especiales (herencia), y que están protegidos para que no los puedan afectar entes de otro tipo diferente (encapsulación). Por tanto, la OOP tiene un mayor alcance que la programación estructurada, debido principalmente a que ésta última comienza a ser deficiente y presentar desventajas cuando se tiene que manejar una cantidad muy grande de información, situación que se presenta actualmente con mucha

frecuencia en la mayoría de las aplicaciones.

Así entonces, la OOP presenta características especiales y esenciales para la implementación (y la más lógica) de los algortimos de las redes neurales, de una manera más apropiada, ya que ambas técnicas tienen la tendencia de tratar de simular y/o emular la forma de concebir y procesar la información del ser humano.

Otro punto a favor de la OOP es su potencial en el desarrollo de medios ambientes gráficos, que son mucho muy importantes cuando se trabaja con redes neurales, debido a que en la mayoría de los casos los usuarios finales de los sitemas desarrollados para la resolución de problemas, no son necesariamente expertos en el área de aplicación, y mucho menos en redes neurales, por lo que es esencial y obligatorio una interfaz lo más amigable posible para que estos usuarios no tengan mayor complicación en el uso de los sistemas. Además, cada vez se vuelve más imprescindible tener una ayuda a través de menús desplegables y/o a través de íconos. Algo importante que se tiene que especificar, es que dichos ambientes gráficos pueden ser (pero no necesariamente) a su vez redes neurales, en donde los íconos y las funciones o herramientas del sistema son las neuronas (nodos), y las relaciones (pesos) entre estas funciones y sus correspondientes íconos son las conecciones de la red.

Al reunir las características de las redes neurales con las de la OOP y las de los ambientes gráficos, se tiene una diversidad enorme de aplicaciones, tanto en la ingeniería, que es el área de interés, como en otras areas como podría ser el análisis financiero o la predicción en el comportamiento de sistemas biológicos.

En este trabajo de tesis se han presentado las técnicas y procedimientos para poder aplicar las redes neurales a la ingeniería de control en específico, no porqué sea el área de mejor aprovechamiento de las redes neurales, sino con el propósito de presentar esta nueva herramienta a un área en que difícilmente se intenta aplicar nuevas técnicas, principalmente a que la gente que maneja las industrias, por lo general, no quiere arriesgarse o simplemente no quiere invertir más capital del indispensable. Sin embargo, la aplicación de redes neurales realmente no presenta una inversión costosa ni arriesgada. En las aplicaciones presentadas en el capítulo 5, se mostró que en la mayoría de los casos prácticos, la implementación de una red neural para propósitos de control, no requiere de más inversión que el desarrollo del sistema (software) y probablemente de algún equipo de cómputo especial (por ejemplo, una computadora con procesamiento en paralelo), ya que se pueden aprovechar los sistemas de control que operan actualmente en las industrias, como es el caso de los PLC's o cualquier otro tipo de sistema automatizado, principalmente si éste es digital.

La intención de esta tesis es despertar el interés en la investigación y en el desarrollo de aplicaciones de redes neurales, presentandose casos prácticos y utilizando las técnicas más nuevas

en programación, como lo es la OOP, los ambientes gráficos, y muchos productos software desarrollados especialmente para las redes neurales. No fué nuestro propósito, en ningún momento, desarrollar nuevos algoritmos o mejorar los ya existentes en redes neurales, que como se vió en el capítulo 1, se tienen situaciones algo complejas en cuanto a la convergencia de los métodos o en los modelos matemáticos abstractos, por lo que intentar presentar algo nuevo en redes neurales saldría de los alcances que se pretendían, además que dichos problemas son practicamente más viables de ser resueltos en tesis de maestría.

Por último, las redes neurales empiezan a tener una atracción fascinante, a causa de sus posibles y muy variadas aplicaciones, para los desarrolladores de sistemas en países de Europa como Francia y Alemania, y en Estados Unidos. En México actualmente se usan, por lo general, sólo en investigaciones. Es por esto que nuestro principal interés es invitar a la gente que desarrolla sistemas a tomar muy en cuenta el uso de las redes neurales y de la OOP en la resolución de problemas, que con las grandes cambios que necesita México, en cuanto a una reconversión industrial con tendencia a una mayor productividad y mejores y más adecuados sistemas de información, posiblemente estas nuevas técnicas y herramientas sean de gran ayuda.

BIBLIOGRAFIA

REDES NEURONALES

[ADAM 92] ADAM BLUM, NEURAL NETWORKS IN C++, AN OBJECT-ORIENTED FRAMEWORK FOR BUILDING CONNECTIONIST SYSTEMS. WILEY PROFESSIONAL COMPUTING,1992

[INVESTIGACION 81] INVESTIGACION Y CIENCIA, EDICION ESPAÑOL, 1981

[MEMORIA 91] SIMPOSIUM NACIONAL DE COMPUTACION 91, INSTITUTO POLITECNICO NACIONAL. NOVIEMBRE 6,7 Y 8 1991

NEURAL COMPUTING: THEORY AND PRACTICE.

[AI EXPERT 90] AI EXPERT. NEURAL NETWORK SPECIAL REPORT. MAGAZINE OF ARTIFICIAL INTELLIGENCE IN PRACTIQUE, 1990

[TANK 86] DAVID W. TANK AND JONH J. HOPFILED. ARTICULO: SIMPLE "NEURAL" OPTIMIZACION NETWORKS.TRANSACTIONS ON CIRCUITS AND SYSTEMS, VOL CAS-33, No 5, MAY 1986

[HOPFIELD 86] JOHN J. HOPFIELD AND DAVIED W. TANK. ARTICULO COMPUTING WITH NEURAL NETWOKS. TRANSACTIONS ON CIRCUITS AND SYSTEMS, VOL CAS-33, No 5, AUGUST 1986

[AI EXPERT 89] AI EXPERT. ARTICULO : RECENT DISCOVERIES HERALD A NEW DIRECTION IN NEURAL-NETWORK RESEARCH, DECEMBER 1989.

[CAUDILL 88] AI EXPERT. ARTICULO: NEURAL NETWORKS PRIMER, JUNE 1988

[RUMELHART 89] RUMELHART, DAVID E., AND McCLELLAND, JAMES L., PARAREL DISTRIBUITED PROCESSING. THE MIT PRESS, 1989

[SIMPSON 90] SIMPSON, PATRICK K, "ASSOCIATIVE MEMORY SYSTEMS," PROCEEDINGS OF THE INTERNATIONAL JINT CONFERENCE ON NEURAL ETWORKS, JANUARY 1990

PROGRAMACION ORIENTADA A OBJETOS

[HERBERT 91] HERBERT SCHILDT, APLIQUE TURBO C++. OSBORNE MCGRAW-HILL, 1991.

[ADAMS 92] LEE ADAMS, LEE ADAMS' SUPERCHARGED C++ GRAPHICS. WINDCREST/MCGRAW-HILL, 1992.

[WEISKAMP 92] KEITH WEISKAMP, POWER GRAPHICS USING TURBO C++, 1992

[CEBALLOS 90] FCO. JAVIER CEBALLOS, CURSO DE PROGRAMACION CON C MICROSOFT C. MACROBIT, 1990

[SMITH 90] SMITH, JERRY D., REUSABILITY & SOFTWARE CONSTRUCTION IN C & C++. NEW YORK: JOHN WILEY & SONS, 1990

INGENIERIA EN CONTROL

[CREUS 81] ANTONIO CREUS SOLE, INSTRUMENTACION INDUSTRIAL. PUBLICACIONES MARCOMBO, 1981

[I&CS 90] CONTROL TECHONOLOGY FOR ENGINEERS AND ENGINEERING MANAGEMENT. SPECIAL REPORT ON: DITRIBUTED CONTROL, 1990