UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FAC. DE INGENIERIA

ESTUDIO DE LAS PROPIEDADES DE GENERALIZACIÓN DE UN NEUROCONTROLADOR

T E S I S
QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA: JOSÉ LUIS NAVA GONZÁLEZ
DIRECTOR DE TESIS: M. C. ALBERTO HERRERA BECERRA

CIUDAD UNIVERSITARIA 1998

AGRADECIMIENTOS

A MIS PADRES:

Por llamarme la atención cuando era necesario, por motivarme y darme la confianza en el momento justo, por ser mis amigos antes nada. Y por muchas otras cosas que no se pueden expresar con palabras. GRACIAS

A MIS ABUELOS:

Por creer en mi y apoyarme siempre que los necesité. Espero nunca defraudarlos.

A LESLIE:

Por darme animo y escucharme siempre, por ser una gran hermana.

A ALBERTO, GRISEL, JOSE LUIS Y SERGIO:

Por la paciencia que tuvieron conmigo, por darme las facilidades para la realización de la tesis, por ser amigos además de maestros.
GRACIAS.

ÍNDICE

Introducción I
Capítulo 1. Antecedentes 1
1.1 Modelos de neuronas artificiales 2
1.1.1 Modelo simplificado de MC Colluch-Pitts 2
1.2 Tipos de redes neuronales artificiales 4
1.2.1 Redes tipo perceptrón multicapa 5
1.3 Métodos de entrenamiento 7
1.3.1 Regla de aprendizaje del perceptrón 8
1.3.2 Método del gradiente 9
1.3.3 Adaptación de una neurona sigmoidal empleando el método del gradiente 10
1.3.4 Método del gradiente para redes 11
1.3.4.1 Retropropagación 12
1.3.4.2 Adaptación por el método del gradiente 13
1.3.4.3 Momentum 15
1.3.5 Retropropagación en el tiempo 15
1.4 Validación y generalización 17
Capítulo 2. Caso de estudio: Control de un péndulo invertido 20
2.1 Descripción del sistema físico 20
2.2 Problema de control 25
2.3 Descripción del neurocontrolador 27
Capítulo 3. Resultados 29
3.1 Introducción 29
3.2 Esquema de entrenamiento del neurocontrolador 29
3.3 Tiempo de Procesamiento 37
3.4 Entrenamiento 40
3.5 Validación 41
3.6 Generalización 46
Conclusiones 52
Prospectivas 55
Apéndice A. Manejo del Sistema Simulador 56
Bibliografía 61

INTRODUCCIÓN

Los métodos convencionales de diseño de sistemas de control involucran la construcción de un modelo matemático que describa el funcionamiento dinámico de la planta a ser controlada y la aplicación de una técnica analítica a este modelo para derivar una ley de control. Generalmente, el modelo matemático consiste de un conjunto de ecuaciones diferenciales o en diferencias lineales o no lineales, la mayoría de las cuales se obtienen mediante alguna forma de aproximación o simplificación. Esas técnicas convencionales fallan cuando un modelo es difícil de obtener o cuando el modelo producido viola alguna de las suposiciones de las leyes de control de la técnica de diseño. También, el modelado de un sistema físico para un control retroalimentado involucra una interacción entre la simplicidad del modelo y la aproximación del funcionamiento del sistema físico.

Dos aproximaciones son empleadas generalmente para realizar satisfactoriamente el desempeño de una planta dinámica vagamente conocida. Una de estas aproximaciones es el controlador robusto. En esta aproximación, si el sistema físico actual es miembro de una clase de sistemas que son parecidos a la planta nominal, un controlador robusto garantiza estabilizarlo. A partir de un controlador fijo se espera controlar un conjunto completo de plantas, el precio puede ser que el controlador así diseñado es altamente complejo comparado con la complejidad requerida para estabilizar cualquier planta simple de esa clase.

El control adaptivo es otra aproximación al problema de control para plantas complejas. Los parámetros del controlador adaptivo son ajustados de acuerdo con algún algoritmo adaptivo para obtener el desempeño del sistema a un nivel deseado. En general, la aproximación adaptiva es aplicable cuando se tienen muchas

incertidumbres, pero los controladores robustos son más simples de implementar, y no se requiere ajustar parámetros para variaciones de la planta.

Un sistema de control adaptivo mide cierto índice de desempeño usando las entradas, los estados y las salidas del sistema dinámico bajo control. De la comparación de la medición del índice de desempeño con el deseado, el mecanismo de adaptación modifica los parámetros del controlador para mantener la respuesta de la planta tan cercana a la respuesta deseada.

La necesidad de controlar sistemas complejos bajo incertidumbres significantes ha provocado una revaluación de las metodologías de control existentes. La evolución en los paradigmas de control se ha basado en dos hechos significativos: la necesidad de trabajar cada vez con sistemas más complejos, y la necesidad de obtener sistemas cada vez más demandantes con menor conocimiento de la planta y el ambiente que le rodea. En esa situación, es casi imperativo para los esquemas de control imponer el aprendizaje adaptivo. La tendencia del control adaptivo, sin embargo puede ser dirigida al desarrollo de aproximaciones más generales y ser utilizada en tantas aplicaciones como sea posible.

Para poder manejar la incertidumbre de la planta dinámica y su ambiente, el controlador tiene que estimar la información desconocida durante su operación. Si esta información estimada se aproxima gradualmente a la información real conforme avanza el tiempo, entonces el controlador así diseñado puede aproximar un controlador óptimo. Debido a la mejora gradual del desempeño producido por la mejora en la estimación de la información, este controlador puede ser visto como un controlador adaptivo de aprendizaje. El controlador aprende la información desconocida durante la operación, y esta información es usada como experiencia para futuras decisiones y controles.

La necesidad de la capacidad de aprendizaje en sistemas de control de sistemas complejos operando en presencia de incertidumbres significantes ha abierto caminos para técnicas de control aparentemente nuevas. Las redes neuronales, con su paralelismo masivo y su habilidad para aprender, ofrecen posibilidades excitantes para la introducción de técnicas de control muy superiores para situaciones complejas. Sin embargo, debe de considerarse que las redes neuronales no son la panacea, y no proporcionan soluciones a todos los problemas.

El campo de las redes neuronales artificiales no es nuevo, pero recientemente se ha convertido en una área activa de investigación. Algunos de los trabajos pioneros en el campo se deben a McCulloch y Pitts en 1943, cuando publicaron un modelo abstracto de una neurona simple. Entonces se consideró que mediante la conexión de muchos de esos dispositivos simples podía ser posible modelar el cerebro humano. A pesar de que su modelo era muy pobre para aproximar las habilidades humanas, fue una influencia para usar redes neuronales o elementos lógicos para construir lo que ahora se conoce como computadoras digitales.

El siguiente desarrollo importante ocurrió en 1949 cuando Hebb propuso un mecanismo de aprendizaje en el cerebro. Él postuló que el cerebro aprende, cambia su patrón de conectividad. La idea de mecanismo de aprendizaje fue incorporada primero en una red neuronal por Rosenblatt en 1959. Él combinó la neurona simple de McCulloch-Pitts con los pesos sinápticos ajustables basados en el esquema de aprendizaje propuesto por Hebb para formar la primera red neuronal con capacidad de aprendizaje.

En 1960, mediante la introducción del algoritmo de error mínimo cuadrático (LMS), Widrow y Hoff desarrollaron un modelo que aprende más rápida y eficientemente.

Muchas de las recientes aplicaciones de redes neuronales han sido en áreas de procesamiento de señales, tales como reconocimiento de patrones, interpretación de imágenes, etc. En sistemas de control, existen también intensas aplicaciones, tales como control e identificación de sistemas en tiempo real. Los sistemas tradicionales de control que operan con incertidumbres típicamente dependen de la intervención de operadores humanos para funcionar apropiadamente. Sin embargo la intervención humana es inaceptable en muchas aplicaciones autónomas de tiempo de real, técnicas de inteligencia artificial para manejar incertidumbres, tales como sistemas expertos, han sido empleados como alternativas a este problema. Las redes neuronales con su procesamiento distribuido puede ser una mejor alternativa para el control en tiempo real debido a su alta velocidad y adaptación a los cambios en los sistemas.

La mayor característica de las redes neuronales es su habilidad para aproximar funciones arbitrarias no lineales. Un neurocontrolador, en general, realiza una forma

específica de control adaptivo. La red neuronal define el problema de control como un mapeo de una señal medida (entradas, salidas, error) para calcular una acción.

El objetivo principal de este trabajo es mostrar la importancia de establecer la capacidad de generalización de las redes neuronales artificiales. Nuestro trabajo se enfocó a investigar hasta que punto una red neuronal puede responder adecuadamente a patrones que nunca fueron presentados durante el entrenamiento y que pueden ser alcanzados por el sistema modificando ciertas condiciones de operación. Para ejemplificar el análisis de sensibilidad de parámetros de una red neuronal se eligió el problema del control del péndulo invertido, ya que es un problema que se emplea comúnmente para evaluar estrategias de control.

CAPÍTULO 1

ANTECEDENTES

Actualmente podemos construir computadoras y otras máquinas que pueden realizar el procesamiento de la información y el aprendizaje mediante principios tomados del funcionamiento de los sistemas nerviosos biológicos, aunque esto no significa que tengamos que intentar copiar el cerebro parte por parte.

Una neurona biológica es una célula capaz de realizar algún tipo de procesamiento de señales. Esta es estimulada por una o más entradas, y genera una salida que es enviada a otras neuronas. La salida depende de las características de cada una de las entradas y de la naturaleza de cada conexión de entrada, llamada sinapsis. Algunas sinapsis pueden ser tales que una entrada tienda a excitar la neurona, esto es, que tienden a incrementar la salida. Otras pueden ser inhibitorias; una entrada tal que la sinapsis tienda a reducir la salida de la neurona.

La relación entre las entradas a la neurona y su salida puede ser enormemente compleja. Por ejemplo, puede haber retardos de tiempo entre la aplicación del estímulo de entrada y la generación de la señal de salida. Una neurona puede no responder siempre de la misma manera a las mismas entradas, eventos aleatorios pueden influir en la operación de una neurona, etc.. Afortunadamente, una gran cantidad de investigadores han mostrado que aún modelos simples de neurona pueden describir de manera suficientemente buena muchos fenómenos neuronales.

Cuando la salida de la neurona es considerada como una salida binaria o como una función continua, es posible describir matemáticamente la relación entre las entradas de la neurona y su salida.

1.1 MODELOS DE NEURONAS ARTIFICIALES

A partir de las abstracciones que se hacen de las neuronas se han construido diferentes tipos de modelos de neuronas artificiales. En la teoría de redes neuronales artificiales se tienen dos tipos de redes, una en la cual los patrones de entrada se pueden mantener todo el tiempo que sea necesario para que la red procese esas entradas, a este tipo de redes se les conoce como redes estáticas. Y existe otro tipo de redes en donde los patrones de entrada son funciones del tiempo y, por lo tanto, no se pueden mantener las entradas; éstas son conocidas como redes dinámicas.

A continuación se presentan algunos de los modelos elementales que han sido desarrollados como abstracciones de las neuronas biológicas.

1.1.1 MODELO SIMPLIFICADO DE McCULLOCH-PITTS

La neurona artificial básica puede ser modelada como un dispositivo procesador de señales no lineal multientrada con interconexiones ponderadas w_i, también llamadas pesos sinápticos. El cuerpo de la célula (soma) es representado por un limitador no lineal o una función de umbral $\Psi(u)$. Esto es, el modelo más simple de una neurona artificial es una suma de las n entradas ponderadas y el resultado es empleado como argumento de una función no lineal de la forma:

$$y = \Psi\left(\sum_{i=1}^{n} w_i x_i + \Theta\right) \tag{1.1}$$

donde Ψ es la función de umbral, Θ es un valor de umbral ($\Theta \in R$), también llamado offset o bias, w_i son los pesos sinápticos, x_i son las entradas (i=1,2,3,...,n), donde n es el número total de entradas y y representa la salida. Generalmente el valor de umbral Θ, es incluido en la sumatoria empleando una entrada adicional x_0=+1 y el correspondiente peso w_0 igual al valor de umbral, esto es, $w_0 = \Theta$. Así la ecuación se puede reescribir como:

$$y = \Psi\left(\sum_{i=0}^{n} w_i x_i\right) \tag{1.2}$$

Normalmente la función de activación puede ser representada por una función no lineal más general y, consecuentemente, la salida de la neurona y puede también asumir un valor de un conjunto discreto {-1,1} o variar continuamente entre -1 y 1, y generalmente, entre y_{min} y y_{max}. En el modelo básico de neurona artificial, la señal de salida está usualmente determinada por una función sigmoide aplicada a la suma ponderada de las señales de entrada. La función sigmoide es una función acotada, monótonamente creciente, con la primera derivada continua, no nula, y con un único punto de inflexión. Esta función puede ser descrita por alguna de las siguientes ecuaciones:

$$y = \tanh(u) = \frac{1 - e^{-2u}}{1 + e^{-2u}} \quad \text{ó} \tag{1.3}$$

$$y = \frac{1}{1 + e^{-u}} \tag{1.4}$$

$$\text{donde} \quad u = \sum_{i=0}^{n} w_i x_i \tag{1.5}$$

En comparación con la función de activación binaria, la función de activación sigmoidal es usualmente más conveniente para implementaciones de hardware analógico. En la figura 1.1 se muestra un diagrama del modelo simplificado de McCulloch-Pitts.

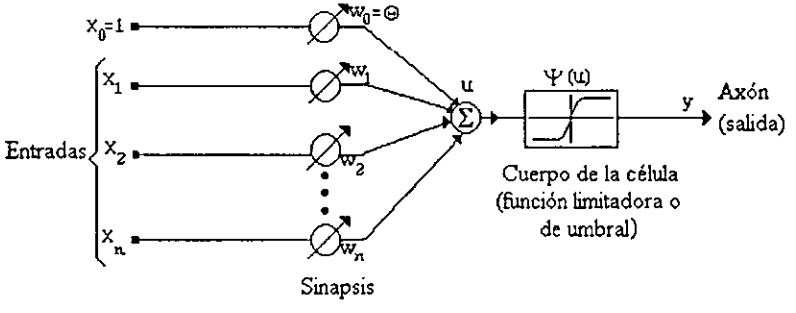


Figura 1.1. Modelo neuronal simplificado de McCulloch-Pitts

1.2 TIPOS DE REDES NEURONALES ARTIFICIALES

De manera general, las redes neuronales artificiales son sistemas de cómputo formados por un número elevado de unidades de procesamiento elementales altamente interconectadas, las cuales cumplen con las siguientes características [Cichocki & Unbehauen 1993]:

1. El procesamiento de la información y la memoria están distribuidos en toda la red.
2. El estado de una neurona afecta el estado de un gran número de otras neuronas a las cuales está interconectada.
3. Los pesos de conexión son, usualmente, los elementos adaptivos de la red.
4. Las redes neuronales formadas por un número elevado de neuronas exhiben una gran robustez al ruido en los valores de los patrones de entrada.

Existen muchas maneras diferentes de interconectar redes neuronales artificiales. Esas diferentes maneras de interconexión entre neuronas son llamadas arquitecturas.

Las arquitecturas de redes neuronales artificiales pueden ser divididas en dos grandes categorías:

- Redes multicapa o redes "Feedforward"
- Redes recurrentes o redes con retroalimentación

En las redes feedforward, las neuronas están arreglas de manera tal que el procesamiento de las señales externas se realiza en una sola dirección. Esto es, cada neurona recibe entradas del ambiente externo y/o de otras neuronas, pero no se permite retroalimentación entre ellas.

Por otro lado, las redes con retroalimentación (llamadas recurrentes) usan neuronas dinámicas y operan en modo de retroalimentación. Las propiedades dinámicas de tales sistemas son descritas por ecuaciones diferenciales o en diferencias. Una red feedforward es, de hecho, un mapeo estático; mientras que las redes recurrentes son representadas por sistemas dinámicos no lineales.

1.2.1 REDES TIPO PERCEPTRÓN MULTICAPA

Un perceptrón multicapa es una red feedforward donde las neuronas están arregladas en capas diferentes. Generalmente todas las neuronas en una capa están conectadas a todas las neuronas en las capas siguientes a través de enlaces unidireccionales. En un perceptrón de tres capas las neuronas están agrupadas en capas conectadas secuencialmente; cada capa es numerada 1,2, ó 3. Las neuronas de la capa 1 (llamada capa de entrada) no realizan cálculo alguno, sólo proporcionan la alimentación de las señales de entrada a la capa 2, llamada primera capa oculta o capa intermedia. La última capa (capa 3, en el caso de un arreglo de tres capas) es la capa de salida, la cual proporciona la respuesta de la red. Las capas entre las neuronas de entrada y las de salida se llaman capas ocultas o capas intermedias. En principio, el número de capas intermedias no tiene límite, pero usualmente en la práctica se usan sólo una o dos capas intermedias. Se ha mostrado experimentalmente que es suficiente emplear un máximo de tres capas (dos intermedias y una de salida) para resolver un problema complejo de clasificación de patrones. En este tipo de redes sólo se permite la conexión de neuronas de una capa a la inmediata siguiente, no se permite la conexión de neuronas en la misma capa.

Para calcular la respuesta de la red en términos de las entradas que recibe se procede como se indica a continuación. La figura 1.2 representa una función actuando sobre un vector $\mathbf{R}^{n+1}$ de la forma $(1,x_1,...,x_n)$, donde cada x_i es un numero real y $n \geq 1$. La red neuronal mapea el vector **X** al vector **Y**; donde $\mathbf{Y}=(y_1,...,y_m)$ en $\mathbf{R}^m$. Así, la red feedforward puede ser representada como: $\mathbf{Y}=\mathbf{F}(\mathbf{X})$.

La acción de esta función está determinada de una manera especifica. Para una red con N nodos de entrada, H nodos intermedios y m nodos de salida, los valores de y_k están dados por:

$$y_k = g\left(\sum_{j=1}^{H} w_{jk}^{o} h_j\right), k = 1,...,m \qquad (1.6)$$

Aquí w_{jk}^{o} es el peso de salida del nodo intermedio j al nodo de salida k, y g es la función de activación de la neurona. Los valores de los nodos intermedios H_j, j=1,...,h están dados por:

$$h_j = sgm(\sum_{i=1}^{N} w_{ij}^I x_i + w_j^T), j = 1,...,h \quad (1.7)$$

Aquí, w_{ij}^I es el peso de entrada para el nodo de entrada *i* al nodo intermedio *j*, w^T es el peso de umbral del nodo de bias de entrada al nodo intermedio j; x_i es el valor al nodo de entrada i, y *sgm* es la llamada función sigmoide.

La función *g* en la ecuación 1.6 puede ser la misma que la función de activación de la capa intermedia o puede ser una función diferente. En algunas implementaciones, se permite que la función *g* sea *sgm* o la función identidad (que es una función de salida lineal).

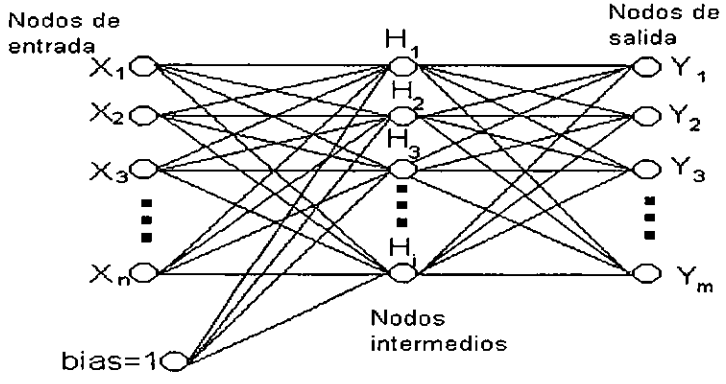


Figura 1.2 Esquema de una red tipo perceptrón multicapa

La acción de la red feedforward está determinada por dos cosas; la arquitectura (*n,h,m*), y los valores de los pesos w_{ij}^0, w_{ij}^I. El número de entradas *n* y salidas *m* están determinados por la aplicación y por lo tanto son fijos. El número de nodos intermedios *h* es una variable que puede ser ajustada por el usuario. Hasta ahora, este ajuste ha sido poco más o menos un "arte", a pesar de que varios métodos para fijar el número de nodos intermedios, o "recortar" nodos innecesarios, han sido propuestos en la literatura [Reed y Marks II, 1995].

En la siguiente sección se mostrarán algunos de los métodos de ajuste de parámetros para una neurona aislada y posteriormente se mostrará el método de retropropagación para entrenar una red neuronal multicapa.

1.3 MÉTODOS DE ENTRENAMIENTO

Una de las propiedades más importantes de las redes neuronales es su capacidad de "aprendizaje", es decir, de ajustarse automáticamente para realizar adecuadamente una tarea especifica. El término de aprendizaje o de entrenamiento se emplea debido a que el funcionamiento de la red se ajusta a partir de ejemplos conocidos.

Una red neuronal es ajustada de una de dos maneras principales. La más común es el llamado entrenamiento supervisado. En este tipo de entrenamiento colectamos muchas muestras que sirven como ejemplo; en cada muestra de este conjunto se especifican completamente las entradas, así como las salidas que se desean obtener cuando se presentan esas entradas. Entonces seleccionamos un subconjunto de entrenamiento y se le presentan las muestras de ese subconjunto una a una a la red. Por cada muestra, comparamos las salidas obtenidas por la red con la salida que deseamos obtener. Después de que todo el subconjunto de entrenamiento ha sido procesado se actualizan los pesos de conexión de las neuronas de la red. Esta actualización se hace de tal manera que se reduzca la medida de error que resulta de la red. A cada iteración en el subconjunto de muestras de entrenamiento, junto con la actualización de los pesos de la red, se le denomina un *epoch*. El número de muestras en el subconjunto es llamado *tamaño del epoch*. En algunos casos es conveniente usar un epoch de tamaño de uno, esto significa que los pesos son actualizados después de que cada caso de entrenamiento es presentado. Cuando el tamaño del epoch es menor que el conjunto de entrenamiento completo, es importante que el subconjunto sea seleccionado aleatoriamente cada vez, ya que pueden ocurrir grandes oscilaciones en la función de error. Los epochs son repetidos hasta que el funcionamiento de la red sea el satisfactorio.

El otro método principal de entrenamiento es el no supervisado. Como en el entrenamiento supervisado, tenemos una colección de muestras de entrada, pero no proporcionamos a la red las salidas para esas muestras. Típicamente suponemos que cada entrada es originada de una de varias clases y la salida de la red es una identificación de la clase a la que pertenece la entrada. El proceso de entrenamiento de la red consiste de tratar de descubrir las características sobresalientes del conjunto de entrenamiento, y usar esas características para agrupar las entradas en clases que la red pueda distinguir.

Existen dos clases principales de algoritmos adaptivos en línea. La primera clase está formada por las reglas de corrección de error lineales, las cuales alteran los pesos de una red para corregir el error en la capa de respuesta de una manera proporcional a ese error. La segunda clase incluye a las reglas del gradiente, que alteran los pesos de la red siguiendo la dirección en la cual el gradiente de la función de error se aproxima a un mínimo.

1.3.1 REGLA DE APRENDIZAJE DEL PERCEPTRÓN

El dispositivo adaptable con umbral es mostrado en la figura 1.3. La adaptación con la regla del perceptrón hace uso del llamado cuantizador de error ξ_k. La cual se define como la diferencia entre la respuesta deseada d_k y la salida del cuantizador y_k.

$$\xi_k = d_k - y_k \tag{1.8}$$

La regla del perceptrón, algunas veces llamado procedimiento de convergencia del perceptrón, no adapta los pesos si la salida y_k es correcta, es decir, si ξ_k =0. Si la salida no coincide con la respuesta binaria deseada d_k, la adaptación es efectuada sumando el vector de entrada con el vector de pesos cuando el error ξ_k es positivo, o sustrayendo el vector de entrada del vector de pesos cuando el error ξ_k es negativo. En otras palabras, se tiene que:

$$\mathbf{W}_{k+1} = \begin{cases} \mathbf{W}_k + \mathbf{X}_k & si \quad \xi > 0 \\ \mathbf{W}_k - \mathbf{X}_k & si \quad \xi < 0 \\ \mathbf{W}_k & si \quad \xi = 0 \end{cases} \tag{1.9}$$

La regla del perceptrón se detiene cuando los patrones de entrenamiento son separados correctamente. Se ha demostrado que la regla del perceptrón permite separar cualquier conjunto de patrones de entrenamiento que sean linealmente separables [Tou y Gonzalez, 1990].

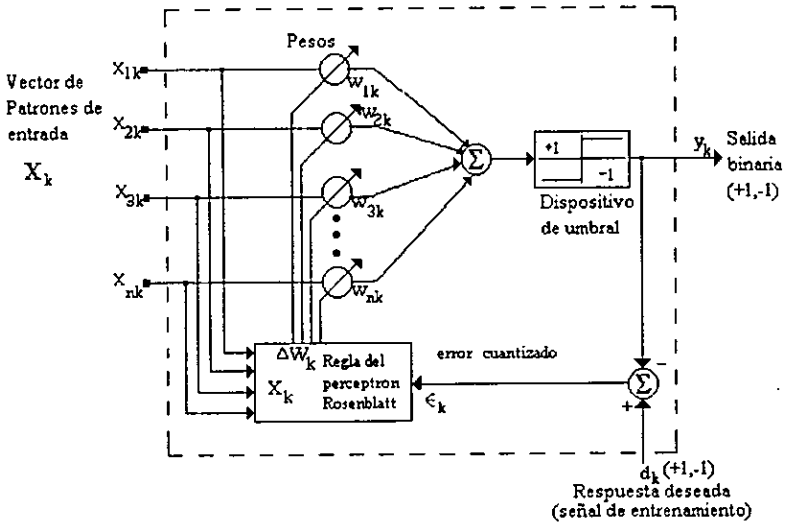


Figura 1.3 El elemento adaptivo de umbral del perceptrón.

1.3.2 MÉTODO DEL GRADIENTE

El objetivo de la adaptación es reducir el error promedio en el conjunto de entrenamiento. La función de error más común es el error medio cuadrático (MSE), quizá en algunas situaciones otro criterio de error puede ser más apropiado. La más popular aproximación para reducir el MSE tanto en un sólo elemento como en redes multielementos están basados en el método de la pendiente descendente. La adaptación de una red mediante la pendiente descendente inicia con un valor arbitrario W_0 para el vector de pesos del sistema. El gradiente de la función MSE es medido y el vector de pesos es alterado en dirección correspondiente al negativo del gradiente medido. El procedimiento es repetido, causando que el MSE sea reducido sucesivamente y causando que el vector de pesos se aproxime al valor optimo local. El método de la pendiente descendente puede ser descrito por la siguiente relación.

$$W_{k+1} = W_k + \mu(-\nabla_k) \tag{1.10}$$

donde μ es un parámetro que controla la estabilidad y rapidez de convergencia, y ∇_k es el valor del gradiente al MSE cuando $W=W_k$.

1.3.3 ADAPTACIÓN DE UNA NEURONA SIGMOIDAL EMPLEANDO EL MÉTODO DEL GRADIENTE

El método que se mostró en la sección anterior puede ser extendido a neuronas con salida continua, por ejemplo, las neuronas sigmoidales. En este caso es necesario calcular el gradiente instantáneo del error cometido por la neurona y, con él, tratar de minimizar el error. Un gradiente instantáneo es obtenido en cada presentación del vector de entrada y el método de la pendiente descendente es usado para minimizar el error.

El gradiente instantáneo estimado durante la presentación del k-ésimo vector de entrada X_k está dado por:

$$\overline{\nabla}\xi_k = \frac{\partial(\xi_k)^2}{\partial W_k} = 2\xi_k \frac{\partial \xi_k}{\partial W_k} \tag{1.11}$$

Haciendo algunas sustituciones e igualdades tenemos:

$$\overline{\nabla}_k = 2\xi_k sgm'(s_k)X_k \tag{2.12}$$

Al usar este gradiente estimado con el método del gradiente se obtiene un medio para minimizar el error medio cuadrático de la forma siguiente:

$$W_{k+1} = W_k + \mu(-\overline{\nabla}_k) = W_k + 2\mu\xi_k sgm'(s_k)X_k \tag{1.13}$$

La implementación del algoritmo es ilustrada en la figura 1.4.

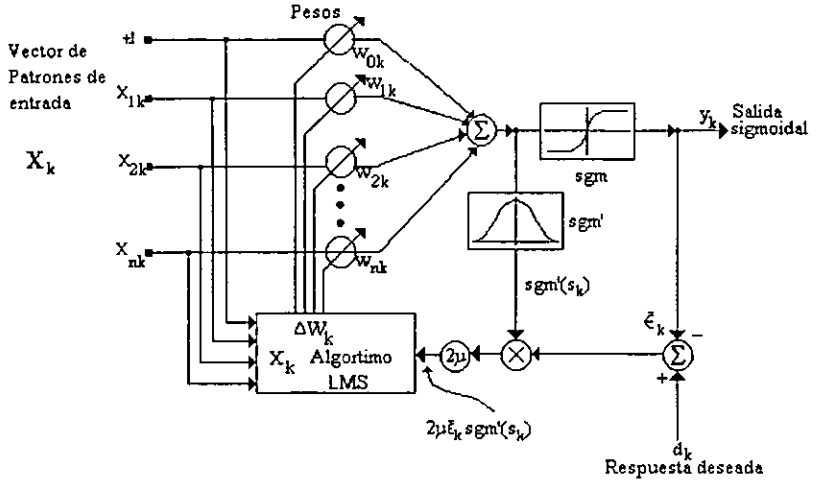


Figura 1.4 Implementación del algoritmo de retropropagación para una neurona con salida sigmoidal.

Si la función sigmoidal seleccionada es la tangente hiperbólica, entonces la derivada sgm'(s_k) esta dada por:

$$\begin{aligned} sgm'(s_k) &= \frac{\partial(\tanh(s_k))}{\partial s_k} \\ sgm'(s_k) &= 1-(\tanh(s_k))^2 = 1-y_k^2 \end{aligned} \tag{1.14}$$

y sustituyendo en la ecuación (1.13) se obtiene la expresión

$$W_{k+1} = W_k + 2\mu\xi_k(1-y_k^2)X_k \tag{1.15}$$

La cual es la que finalmente empleamos para realizar el ajuste de la neurona sigmoidal.

1.3.4 MÉTODO DEL GRADIENTE PARA REDES

Comúnmente se le denomina a este método de entrenamiento como retropropagación, pero retropropagación es sólo un método para calcular derivadas de manera exacta y eficientemente en grandes sistemas compuestos de subsistemas elementales.

1.3.4.1 RETROPROPAGACIÓN

Dentro del método del gradiente retropropagación se emplea para obtener las derivadas parciales de la función de error respecto a cada uno de los pesos de conexión de la red.

El concepto clave en el método del gradiente es la sensibilidad del error de la red a cambios en sus pesos. En términos matemáticos, necesitamos conocer las derivadas parciales del error con respecto a los pesos.

La derivada del error con respecto a un peso se calcula multiplicando una serie de términos que corresponden a la medida en que el peso contribuye al error. Los términos son como eslabones de una cadena que va del peso al error.

Por ejemplo, considere como un peso de un nodo intermedio afecta el error. Primero, el peso afecta directamente la suma ponderada de las entradas a ese nodo. Esta suma afecta la salida sigmoidal del nodo. El valor es pasado a todos los nodos de salida y es parte de la suma ponderada de las entradas en cada nodo de salida, las cuales influyen en su salida sigmoidal. Finalmente, esa salida es comparada con la salida deseada para determinar el error.

Para obtener la derivada de la función de error con respecto a un nodo de salida tenemos que considerar tres términos:

$$\frac{\partial \xi}{\partial b} = \frac{\partial \xi}{\partial z} \cdot \frac{\partial z}{\partial v} \cdot \frac{\partial v}{\partial b} \tag{1.16}$$

En ésta ecuación la derivada del error con respecto a un peso de un nodo de salida ($\frac{\partial \xi}{\partial b}$) es el producto de :

I. La derivada del error con respecto a la salida de ese nodo $\frac{\partial \xi}{\partial z}$

II. La derivada de la salida de ese nodo con respecto a su suma ponderada de las entradas $\frac{\partial z}{\partial v}$

III. La derivada de esa suma con respecto al peso en cuestión $\frac{\partial v}{\partial b}$

Las derivadas del error de la red con respecto a un nodo intermedio se encuentran también multiplicando tres términos:

$$\frac{\partial \xi}{\partial a} = \frac{\partial \xi}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial a} \tag{1.17}$$

La derivada de error con respecto a un peso de un nodo intermedio ($\frac{\partial \xi}{\partial a} = \frac{\partial \xi}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial a}$), es el producto de:

I. La derivada del error con respecto a la salida de ese nodo $\frac{\partial \xi}{\partial y}$

II. La derivada de la salida de ese nodo con respecto a su suma ponderada de las entradas $\frac{\partial y}{\partial u}$

III. La derivada de esa suma con respecto al peso en cuestión $\frac{\partial u}{\partial a}$

El conjunto de derivadas parciales forma el gradiente instantáneo de la función de error, el cual es empleado para realizar la adaptación de los pesos.

1.3.4.2 ADAPTACIÓN POR EL MÉTODO DEL GRADIENTE

La descripción del método del gradiente está basado en el trabajo de Rumelhart, Hilton y Williams (1986a, 1986b).

Cada epoch todos los nodos cambian sus pesos basados en el valor del gradiente de la función de error. Como el nombre de gradiente implica, esos pesos se muevan en la dirección en que el error desciende más rápidamente.

Para ver como se hace esto, piense en la red moviéndose a través de una serie de puntos en el espacio de pesos. El espacio de pesos de la red tiene una dimensión por cada peso, cada punto en el espacio de pesos corresponde a un conjunto completo de valores de pesos para la red. Durante el proceso de entrenamiento la red recorre una serie de estos puntos, tratando de alcanzar el punto en que la función de error se minimiza.

Para un punto dado en el espacio de pesos, nuestro principal problema es determinar en que dirección decrementa mas rápidamente el error, y el segundo problema es decidir que tanto nos movemos en esa dirección para ajustar los pesos.

Para determinar la dirección y la magnitud del ajuste se debe calcular el error cuadrático para cada una de las muestras de entrenamiento, se obtiene el porcentaje del error a través de todas las muestras y el gradiente que se emplea es el producido por el error promedio obtenido.

Cuando se aplica una regla del gradiente, la dirección en la cual los pesos son ajustados es el negativo del gradiente de la función de error y la cantidad de cambio es proporcional a cada componente del vector. La relación entre la dirección y la cantidad de cambio es controlada por un parámetro llamado épsilon.

Si tenemos que W_i representa el vector de pesos después del epoch i

$$W_{i+1} = W_i + c_i \tag{1.18}$$

donde c_i es el cambio en los pesos W al final del epoch i. Este cambio se define como:

$$c_i = -\varepsilon d_i \tag{1.19}$$

donde ε es el parámetro controlando la proporción del cambio, el signo negativo hace el cambio en dirección en la que el error decae más rápidamente, d_i es la suma de las derivadas de la función de error de todas las muestras con respecto a los pesos.

$$d_i = \sum_{n=1}^{N} \left(\frac{\partial \xi}{\partial W_i} \right)_n \tag{1.20}$$

El valor del parámetro ε es definido por el usuario, diferentes valores trabajan mejor en diferentes conjuntos de datos. La única forma segura de descubrir el mejor valor para usar es tratar diferentes valores y observar que pasa. Esta aproximación de prueba y error es una de las desventajas del método del gradiente.

1.3.4.3 MOMENTUM

Si los pesos son cambiados después de cada muestra, la red puede perder tiempo saltando en la superficie de error alrededor del mínimo sin llegar a él. El momentum es una variación del método del gradiente. En lugar de promediar el error, el momentum promedia el cambio de pesos mismo. Primero calculamos el cambio que correspondería a la dirección del gradiente en el punto actual del vector de pesos. Pero el cambio que se realiza es el promedio entre éste cambio indicado y el último cambio realizado. Matemáticamente este método puede escribirse como

$$c_i = \mu c_{i-1} - (1-\mu)\varepsilon d_i \tag{1.21}$$

donde $0 \leq \mu \leq 1$, el momentum puede ser empleado cuando se realiza el entrenamiento por epoch o por ejemplo.

1.3.5 RETROPROPAGACIÓN EN EL TIEMPO

El algoritmo de retropropagación para redes feedforward (Rumelhart, Hinton and Williams, 1986) ha sido aplicado exitosamente a una gran variedad de problemas. Sin embargo los problemas que pueden ser resueltos por una red feedforward son sólo mapeos estáticos de un vector de entrada. Para modelar funciones dinámicas es esencial utilizar un sistema capaz de almacenar estados internos e implementar dinámicas complejas.

Una red recurrente es capaz de transformar una secuencia de entradas en alguna otra secuencia de salida. Esto puede ser usado como un filtro no lineal (Kenji Doya, 1989), un controlador no lineal (Kenji Doya, 1992), o una máquina finita de estado (Giles et al, 1992).

El algoritmo de aprendizaje para redes recurrentes para realizar tareas en el tiempo es similar al caso de las redes feedforward, una red que proporciona una salida deseada para una entrada dada. Se define una función de error y se calcula el gradiente con respecto a los pesos. Sin embargo, la principal diferencia es que las entradas y las salidas no son vectores estáticos, sino secuencias en el tiempo.

En estudios recientes, el algoritmo estándar de retropropagación fue adaptado para entrenar redes recurrentes, considerándolas como redes feedforward. Sin embargo, este esquema de aprendizaje aproximado toma en cuenta la dinámica de la red para sólo un paso en el tiempo.

En una red recurrente, un cambio en un peso puede afectar el funcionamiento futuro de la red completa. Los algoritmos de aprendizaje que toman en cuenta este efecto recurrente han sido obtenidos tanto para modelos discretos (Rumelhart et al 1986; Williams and Zipser, 1989) como para modelos continuos (Pearlmutter 1989; Doya and Yoshizawa, 1989; Rowat and Selverston, 1991). El principio básico es obtener una versión linealizada de la red y estimar el efecto de un cambio pequeño en los pesos dentro de la función de error. Los algoritmos para obtener un modelo discreto pueden ser derivados mediante la transformación de una red recurrente en una red multicapa (Rumelhart et al 1986). En este esquema, la iteración T de una red recurrente es considerada como un barrido en la capa T de la red feedforward con idénticos pesos de conexión entre capas sucesivas.

Primero, iniciamos con un modelo discreto. Denotamos el estado de la neurona i como y_i y el peso de conexión de la neurona j a la neurona i por w_{ij}. Tanto las entradas externas u_i y las entradas recurrentes y_j son representadas como z_i por conveniencia.

$$y_i(t+1) = f\left(\sum_{j=1}^{n+m} w_{ij} z_j(t)\right) i = 1,\ldots,n$$
$$z_j(t) = \begin{cases} y_j(t) & j \le n \\ u_{j-n} & j > n \end{cases} \tag{1.22}$$

El error del gradiente puede ser derivado en la misma manera que en la retropropagación estándar, excepto que el error de salida no sólo está dado en la última capa, sino también es agregado en cada capa.

$$\frac{\partial E}{\partial y_i(t)} = \sum_{j=1}^{n} \frac{\partial E}{\partial y_i(t+1)} f'(x_i(t)) w_{ij} + \mu(t)(y_i(t) - d_i(t)) \tag{1.23}$$
$$i = 1,\ldots n$$

Debido a que el error E es independiente del estado al tiempo t > T, la condición límite para la ecuación (1.23) esta dada al final del paso como:

$$\frac{\partial E}{\partial y_i(T+1)} = 0 \qquad i = 1,\ldots n$$

Así, la ecuación (1.23) puede ser iterada inversamente en el tiempo de t = T a 1.

De la solución $\partial E / \partial y_i$; el error del gradiente está dado por:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial y_i(t)} f'(x_i(t-1)) z_j(t-1) \tag{1.24}$$

y los pesos son actualizados en lote por la siguiente ecuación

$$w_{ij}(t+1) = w_{ij}(t) - \varepsilon \sum_{j=1}^{n} \mu(t)(y_i(t) - d_i(t)) \frac{\partial y_i(t)}{\partial w_{ij}} \tag{1.25}$$

La ventaja del algoritmo es que tenemos que resolver sólo un sistema *n* dimensional para ajustar todos los pesos.

1.3 VALIDACIÓN Y GENERALIZACIÓN

Sería arriesgado entrenar una red e inmediatamente ponerla en funcionamiento. Primero debe ser evaluada su confiabilidad. Este proceso es llamado validación. El procedimiento usual es separar los casos conocidos en dos conjuntos separados. Uno es el conjunto de entrenamiento, el cual es usado para entrenar la red. El otro es el conjunto de validación, el cual es usado para probar a la red entrenada.

En muchos aspectos, la validación es más importante que el propio entrenamiento. Es muy fácil observar un error pequeño en el conjunto de entrenamiento y, erróneamente, concluir que el funcionamiento de la red es el correcto. Si el modelo tiene muchos parámetros libres, relativos al número de casos en el conjunto de entrenamiento, se puede producir un sobreentrenamiento de los datos. En lugar de aprender la estructura básica de los datos, empleada para generalizar, se aprenden los detalles irrelevantes de los casos individuales. Naturalmente, esperamos que el error del conjunto de validación exceda ligeramente al error de conjunto de entrenamiento. Pero si la diferencia es grande, podemos sospechar que alguno de los dos conjuntos no es representativo de la misma población, o que el modelo ha sido sobreentrenado. En cualquier caso, la disparidad de errores es una señal de alarma

que no debe ser ignorada. Es imperativo que el conjunto de validación no sea usado como parte del proceso de entrenamiento de ninguna manera. Casi nadie permite que parte o todo el conjunto de validación caiga en el conjunto de entrenamiento. Es un error muy obvio, pero puede ocurrir un error mucho más fino si se realiza más de un ciclo de entrenamiento-validación. Suponga que la red entrenada inicialmente funcionó pobremente en el conjunto de validación. El proceso de entrenamiento puede hacerse de nuevo, esta vez iniciando con una configuración de pesos aleatorios diferente. Es posible que la red entrenada resultante tenga una configuración final de pesos muy diferente que la primera red entrenada. La nueva red es probada ahora con el conjunto de validación. Si responde bien se acepta, de otra forma se realiza un nuevo entrenamiento. Esto se hace repetidamente hasta que el funcionamiento con el conjunto de validación sea aceptable. Este proceso ha usado esencialmente el conjunto de validación como conjunto de entrenamiento, siendo inapropiado para la validación.

¿Porqué el proceso anterior no es válido? Por que el propósito de la fase de validación es proporcionarnos una indicación de que clase de funcionamiento podemos esperar de la red cuando sea usada con la población restante. Incluyendo el conjunto de validación en el ciclo de entrenamiento, lo hemos predispuesto. Y hemos seleccionado nuestra red final especialmente para responder mejor al conjunto de validación, lo cual nos priva de una confirmación independiente.

¿Qué debemos hacer si nuestra red se desempeña pobremente en el conjunto de validación? Primero debemos darnos cuenta que información importante debe encontrarse en el conjunto de validación, información que la red no ha aprendido. Puede suceder que una pobre configuración de los pesos aleatorios de inicio sea la responsable. Pero es más probable que el conjunto de entrenamiento haya sido diseñado de una manera incompleta. Sin embargo el primer paso es combinar los conjuntos de validación y de entrenamiento en un gran conjunto de entrenamiento. La red es entonces reentrenada hasta que el desempeño de su conjunto de entrenamiento aumentado sea aceptable. Entonces colectamos un nuevo conjunto de validación. Probablemente esto sea caro (en términos de tiempo de procesamiento ó en tiempo de experimentación) pero no tan caro como poner en operación una red incompetente.

La generalización es una propiedad de las redes neuronales artificiales y se refiere a la capacidad que las redes neuronales poseen para responder adecuadamente ó de forma aceptable a condiciones desconocidas para la red, es decir, a situaciones que no se presentaron en la etapa de entrenamiento.

CAPÍTULO 2
CASO DE ESTUDIO:
CONTROL DE UN PÉNDULO INVERTIDO

El péndulo invertido es un problema clásico de control, el cual ha sido analizado por casi todas las técnicas de control conocidas, esto se debe a las características inherentes de inestabilidad que presenta. Ahora este problema será analizado empleando una red neuronal como controlador, la función que la red neuronal tendrá como controlador es predecir las condiciones del péndulo en el siguiente intervalo de tiempo para determinar si es necesario aplicar una fuerza que permita mantener balanceado el péndulo. Esto es parecido a balancear una escoba sobre la palma de la mano, el controlador, en este caso la persona que balancea la escoba, observa las condiciones en las que se encuentra la escoba y si considera que es necesario aplicar una fuerza correctiva moverá la mano en la dirección en que la escoba se incline. Bajo este principio se construirá la red neuronal que permitirá el control del modelo de péndulo invertido que se ha seleccionado. Cabe hacer notar que la propuesta del controlador es original de Charles Anderson, y lo que se demostrará en este trabajo son las propiedades de las redes neuronales, tales como la generalización, aproximación de funciones y los pasos que se deben seguir en la construcción de una red neuronal, como es el entrenamiento y la validación.

2.1 DESCRIPCIÓN DEL SISTEMA FÍSICO

El sistema dinámico consiste de un carro en el cual está montado un bastón (un péndulo invertido) con un soporte esférico casi sin fricción. El bastón puede caer libremente sobre el eje del pivote (sobre el plano XY). El sistema es de gran interés, ya

que representa la idealización de un sistema mecánico inestable, el cual es monitoreado de tiempo en tiempo para ser controlado. La posición del ángulo θ del bastón es medida con el objeto de usar esta señal para manejar el carro y balancear el bastón. Para el análisis, suponemos que el pivote esta completamente libre de fricción, y que las ruedas del carro no patinan sobre la superficie en la cual está montado el carro.

Para obtener las ecuaciones de movimiento se deben escribir las ecuaciones de equilibrio que describen el balance de fuerzas que debe existir para el sistema, también se deben escribir las relaciones de compatibilidad que describen que elementos del sistema se interrelacionan debido a la manera en que se conectan.

Además hay que tomar en cuenta ciertas consideraciones para la obtención de las ecuaciones. Una es la selección de las variables físicas que describen el estado del sistema y en términos de que variables se estudiará el sistema.

En sistemas mecánicos es de gran ayuda el empleo de los diagramas de cuerpo libre, ya que podemos observar el estado de equilibrio del sistema.

A continuación procederemos a la obtención de las ecuaciones del sistema, para ello haremos uso de los dos diferentes diagramas de cuerpo libre mostrados en las figuras 2.1 y 2.2; empleando el principio de D'Alembert[1] para obtener las ecuaciones de equilibrio del sistema.

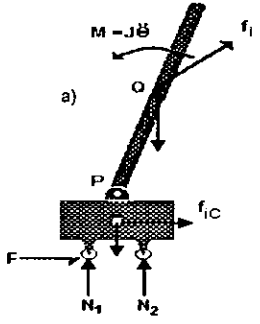


Figura 2.1 Diagrama de cuerpo libre del sistema completo

[1]El principio de D'Alembert establece que las fuerzas aplicadas a un elemento, junto con las fuerzas de inercia forman un sistema en equilibrio.

Si en el diagrama de cuerpo libre del sistema completo (Ver Figura 2.1) empleamos el principio de D'Alembert aplicado a sistemas mecánicos traslacionales en la dirección del eje X tenemos la siguiente ecuación:

$$\sum f_x = (-m_c\ddot{x}) + (-m\mathbf{a}^Q)_x + F(t) = 0 \tag{2.1}$$

donde Σf_X es la ecuación de equilibrio de fuerzas en dirección del eje X, $(-m_c\ddot{x})$ es la fuerza debida a la masa del carrito, $(-m\mathbf{a}^Q)_x$ es la fuerza debida a la masa del bastón y F(t) es la fuerza externa de tracción aplicada entre el carrito y el piso.

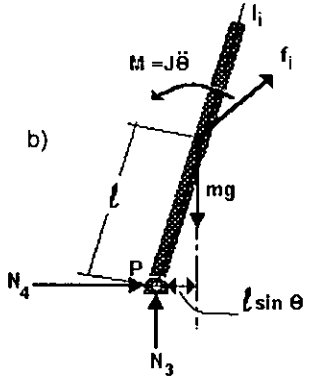


Figura 2.2 Diagrama de cuerpo libre del péndulo invertido

Si en el diagrama de cuerpo libre del péndulo (Ver figura 2.2) empleamos el principio de D'Alembert aplicado a sistemas mecánicos rotacionales en el pivote P tenemos la siguiente ecuación:

$$\sum M_P^{\cdot} = 0 = M_i - [m\ell(\ddot{x}\cos\theta + \ell\ddot{\theta})] - mg\ell\sin\theta \tag{2.2}$$

donde ΣM_P es la ecuación de equilibrio de fuerzas de rotación que actúan sobre el pivote, M_i es la fuerza de rotación debida a la inercia, $\left[m\ell(\ddot{x}\cos\theta + \ell\ddot{\theta})\right]$ es la componente virtual de D'Alembert para sistemas rotacionales, $mg\ell\sin\theta$ es la fuerza de rotación debida a la fuerza de gravedad.

Por definición sabemos que:

$$M_i = J\ddot{\theta}$$

donde J es el momento de inercia del bastón al rededor de su centro de masa: $J=m\ell^2/3$ para un bastón de longitud 2ℓ

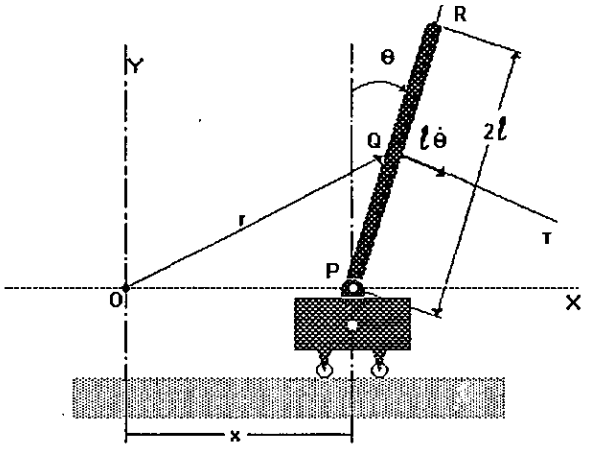


Figura 2.3 Diagrama de posición del sistema dinámico.

Como podemos ver en la figura 2.3, el sistema se muestra en una posición arbitraria. Para obtener las ecuaciones de movimiento del sistema se necesita la aceleración del centro de masa del bastón Q, para emplearla en la ecuación (2.1).

Para el bastón podemos obtener la aceleración requerida con una diferenciación escalar trabajando con el eje coordenado XY mostrado en la figura 2.3. Primero escribimos el vector de posición **r** del centro de masa del bastón Q en el eje coordenado XY:

$$\begin{aligned}\mathbf{r}^{Q} &= \mathbf{r}^{P} + \mathbf{r}^{Q/P} \\ &= \mathrm{l}_x x + (\mathrm{l}_x \sin\theta + \mathrm{l}_y \cos\theta)\end{aligned} \tag{2.3}$$

donde $\mathbf{r}^{Q}$ es el vector de posición del centro de masa del bastón con respecto al origen del sistema, $\mathbf{r}^{P}$ es el vector de posición del pivote con respecto al origen y $\mathbf{r}^{Q/P}$ es el vector de posición de centro de masa del bastón con respecto al pivote.

Debido a que el eje coordenado XY no se rota, podemos diferenciar la ecuación anterior por componentes para obtener $\mathbf{v}^Q$:

$$\mathbf{v}^Q = \dot{\mathbf{r}}^Q = \mathbf{l}_x(\dot{x} + \ell\dot{\theta}\cos\theta) + \mathbf{l}_y(-\ell\dot{\theta}\sin\theta) \tag{2.4}$$

Y derivamos nuevamente para obtener $\mathbf{a}^Q$:

$$\mathbf{a}^Q = \dot{\mathbf{v}}^Q = \mathbf{l}_x(\ddot{x} + \ell\ddot{\theta}\cos\theta - \ell\dot{\theta}^2\sin\theta) + \mathbf{l}_y(-\ell\ddot{\theta}\sin\theta - \ell\dot{\theta}^2\cos\theta) \tag{2.5}$$

donde $\mathbf{a}^Q$ es la aceleración del bastón y $\mathbf{v}^Q$ es la velocidad del bastón.

El sistema dinámico trabajará en un rango muy pequeño de ángulos, por lo que la componente en Y tiende a ser constante y si derivamos dos veces esta componente veremos que para estos valores de ángulo esta componente tiende a ser igual a cero, así que podemos discriminar la componente que se encuentra en el eje Y de la ecuación (2.5).

Sustituyendo la ecuación (2.5) en la ecuación (2.1), se obtiene el siguiente sistema:

$$\begin{aligned} &m_c x'' + mx'' + m\ell\theta''\cos\theta - m\ell\theta'^2\sin\theta = F(t) \\ &J\theta'' + m\ell(x''\cos\theta + \ell\theta'') - mg\ell\sin\theta = 0 \end{aligned} \tag{2.6}$$

Si despejamos $\ddot{x}$ de la primera ecuación del sistema 3.6 y la sustituimos en la segunda ecuación y finalmente despejamos $\ddot{\theta}$ de esta ultima ecuación tenemos el siguiente sistema de ecuaciones:

$$\begin{aligned} \ddot{x} &= \frac{F(t) + m\ell[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta]}{m_C + m} \\ \ddot{\theta} &= \frac{(m_C + m)g\sin\theta - \cos\theta[F(t) + m\ell\dot{\theta}^2\sin\theta]}{4/3\ell(m_C + m) - m\ell\cos^2\theta} \end{aligned} \tag{2.7}$$

donde m_C es la masa de carro, m es la masa del bastón y ℓ es la longitud del bastón desde el pivote hasta el centro de masa.

Ahora hacemos una redefinición de las literales de la siguiente manera:

$m = m_C + m$ (m es igual a la masa combinada del péndulo y el carro);

$m_b = m$ (m_b es la masa de péndulo)

Y finalmente tenemos el siguiente sistema de ecuaciones:

$$\ddot{x} = \frac{F(t) + m_b \ell[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta]}{m}$$
$$\ddot{\theta} = \frac{mg\sin\theta - \cos\theta[F(t) + m_b \ell \dot{\theta}^2 \sin\theta]}{4/3\ell m - m_b \ell \cos^2\theta} \qquad (2.8)$$

donde θ es el ángulo del péndulo (en radianes) con respecto a la vertical; $\dot{\theta}$ es su velocidad angular (rad/s); x es la posición del carro (en metros) y $\dot{x}$ es la velocidad del carro (m/s); F(t) es la magnitud de la fuerza externa aplicada al carro al tiempo t (en newtons [N]): por último, las constantes m, ℓ, m_b y g; corresponden a la masa combinada del carro y el péndulo, la longitud del péndulo (del punto del pivote al centro de masa), la masa del péndulo y la constante gravitacional, respectivamente. En las simulaciones estas constantes se ajustaron de la manera siguiente: m=1.1kg, ℓ=0.5m (donde la longitud total de péndulo es de 1m), m_b=0.1kg y g=9.8m/s^2.

Así de esta manera terminamos el estudio de la dinámica del sistema.

2.2 PROBLEMA DE CONTROL

El esquema de control que se emplea se muestra a continuación

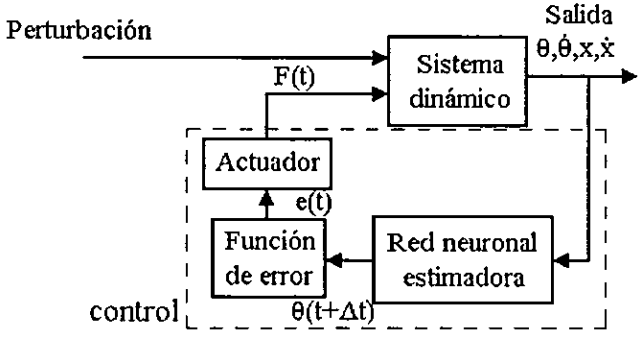


Figura 2.4 Diagrama del esquema de control empleado.

Se emplea a la red neuronal para obtener el valor estimado del ángulo de inclinación del péndulo a un intervalo de tiempo posterior, si se activa la función de error (ecuación 2.9) se aplica una fuerza correctiva al sistema por medio del actuador para mantener el sistema en equilibrio (ecuación 2.10).

Para efectos de simulación del sistema se ha considerado que el carrito se encuentra montado sobre un riel de longitud fija. Ahora bien, se considera que a ocurrido una falla cuando alguno de los extremos del carro alcanza el límite del riel o cuando el ángulo de la barra con respecto de la vertical excede algún valor límite. De estas manera, hemos definido la señal de falla del sistema como sigue:

$$e(t) = \begin{cases} 1, & si\ |\theta(t)| > \text{ángulo de tolerancia} \quad o \quad |x(t)| > \text{long. del riel} \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{2.9}$$

y la ecuación del actuador está dada por:

$$F(t) = \begin{cases} F, & si\ e(t) = 1 \quad y \quad \theta(t + \Delta t) > 0 \\ -F, & si\ e(t) = 1 \quad y \quad \theta(t + \Delta t) < 0 \\ 0, & \text{en cualquier otro caso} \end{cases} \tag{2.10}$$

El sistema de control propuesto está basado en una red neuronal que proporciona una estimación del estado futuro del sistema a partir del conocimiento de su estado actual. Un estado se define por el valor instantáneo de las variables θ (t), $\dot{\theta}$ (t), x(t) y $\dot{x}$ (t). Así, una vez conocido el estado futuro del sistema, se puede tomar una acción de control adecuada para asegurar que el péndulo se mantenga balanceado. El esquema de control propuesto establece la dirección del forzamiento que se debe aplicar al péndulo en el instante de tiempo t+Δt a partir del conocimiento de su estado en el instante de tiempo t, considerando adicionalmente que la magnitud de la fuerza de control aplicada es constante.

Ahora se tienen que hacer ciertas consideraciones antes de diseñar la red neuronal. Como se puede ver el sistema dinámico es no lineal y continuo, con lo cual se debería de emplear una red neuronal dinámica, estas redes son más complejas que las estáticas, así que si consideramos que el sistema continuo puede ser aproximado como un sistema discreto, aplicándole una frecuencia de muestreo adecuada, es posible emplear una red neuronal estática.

El sistema de ecuaciones 3.8 es empleado con el método de Euler de tal manera que la forma de obtener los valores de las variables de estado es la siguiente:

$$\begin{aligned} \dot{\theta}(t+\Delta t) &= \dot{\theta}(t) + \Delta t\ddot{\theta}(t) \\ \theta(t+\Delta t) &= \theta(t) + \Delta t\dot{\theta}(t) \\ \dot{x}(t+\Delta t) &= \dot{x}(t) + \Delta t\ddot{x}(t) \\ x(t+\Delta t) &= x(t) + \Delta t\dot{x}(t) \end{aligned} \qquad (2.11)$$

donde Δt es el periodo de muestreo del sistema.

2.3 DESCRIPCIÓN DEL NEUROCONTROLADOR

La red neuronal propuesta es un perceptrón multicapa formado por cinco neuronas en la capa de entrada, N en la intermedia (donde N será definido en la fase de construcción de la red) y una en la de respuesta. Ahora bien, ya que el estado del sistema se define por las variables $\theta(t)$, $\dot{\theta}(t)$, $x(t)$ y $\dot{x}(t)$, entonces cuatro de las neuronas de la capa de entrada están asociadas a dichos valores, mientras que la restante se asocia a la fuerza aplicada. Adicionalmente, la neurona de salida se asocia al valor del ángulo en el intervalo de tiempo siguiente $(t+\Delta t)$ resultante de aplicar la fuerza bajo las condiciones de la entrada.

La arquitectura de red empleada en el controlador es la siguiente:

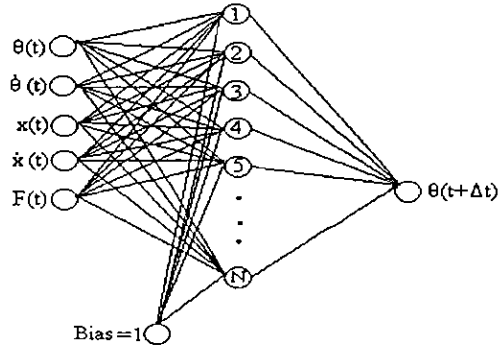


Figura 2.5 Arquitectura de red empleada para el análisis del controlador

El entrenamiento se realiza con una variación del algoritmo de retropropagación, esta variación es conocida como retropropagación en el tiempo y fue creada por Paul Werbos [1994].

Retropropagación en el tiempo se ha usado con mayor frecuencia en predicción de series en el tiempo y para aproximar sistemas dinámicos. Este tipo de retropropagación es más precisa ya que empleamos el conocimiento de lo que pasó en tiempos anteriores. La red neuronal puede realizar mejor su tarea si puede saber los cambios en el sistema del tiempo t-1 al tiempo t. En donde la representación del sistema al tiempo t esta basado en un ajuste de la representación al tiempo t-1; para esto se requiere la memoria de las variables internas de la red en el tiempo t-1.

Una vez que hemos definido el problema y la forma que será analizado, es necesario realizar las simulaciones correspondientes para comprobar el esquema planteado. En el capítulo siguiente se mostrarán los resultados de las simulaciones realizadas y se especificará la forma definitiva de la red neuronal que se consideró como la mejor controladora.

CAPÍTULO 3
RESULTADOS

3.1. INTRODUCCIÓN

El entrenamiento es de gran importancia ya que de ello depende que nuestra red funcione adecuadamente o no. Primero tenemos que definir cual será las forma de generar el archivo de entrenamiento y las condiciones de validación y generalización de las redes obtenidas.

Debido a que este sistema físico ha sido estudiado ampliamente, algunos autores han mostrado que el sistema es controlable para ángulos pequeños (entre 10° y 15°), por eso se definió el ángulo de tolerancia en 12°, además se delimitó la longitud del riel a ±2.4m y el periodo de muestreo se ajustó a 0.02s, ya que estos valores han sido usados por algunos autores tales como Anderson.

3.1. ESQUEMA DE ENTRENAMIENTO DEL NEUROCONTROLADOR

Aquí debido a que se desea aproximar un sistema dinámico, en lugar de definir un conjunto de entrenamiento se toma el sistema de ecuaciones 2.11 considerando que el péndulo se encuentra en el centro del riel y el ángulo de la barra con respecto a la vertical igual a cero. A continuación se le aplica una fuerza aleatoria (tanto en magnitud como en sentido) en cada periodo de muestreo y empleando el sistema 2.11, se calcula el valor de cada una de las variables de estado en el siguiente periodo de muestreo, el objetivo es tener los valores máximos absolutos de las velocidades tanto del carro como angular, ya que la posición y el ángulo de tolerancia máximos son fijos para la condición de error, si el ángulo excede en ángulo de tolerancia o si el

carrito toca un extremo del riel, se coloca el carrito nuevamente en la posición inicial y se calcula un nuevo estado. Esto se realiza tantas veces como número de patrones se hayan definido. Es importante remarcar que no se almacenan todos los estados que se generan, sino sólo los valores que se tuvieron como máximos para cada una de las variables de estado. En la figura 3.1 se presenta un diagrama de flujo del algoritmo que se empleo para generar el archivo de entrenamiento.

Primero se tiene que definir el número de patrones de entrenamiento, en el algoritmo tradicional estos n patrones son los mismos para todas las iteraciones que se requieran; pero en esta adaptación para cada iteración de n patrones, estos son diferentes; es decir que el conjunto de entrenamiento es diferente cada vez.

Para realizar el entrenamiento, se inicializa el conjunto de pesos de la red con valores aleatorios, se genera un valor aleatorio para cada una de las variables, la condición es que el valor se encuentre dentro de los siguientes limites:

$$12^{\circ} \leq \theta(t) \leq 12^{\circ}$$
$$-\dot{\theta}_{max} \leq \dot{\theta}(t) \leq \dot{\theta}_{max}$$
$$-2.4m \leq x(t) \leq 2.4m$$
$$-\dot{x}_{max} \leq \dot{x}(t) \leq \dot{x}_{max}$$
$$-10N \leq F(t) \leq 10N$$

Esto es con el objeto de garantizar que el estado que se genero pertenezca a los que el sistema puede alcanzar durante su funcionamiento. A continuación se obtiene la salida que la red nos proporciona con el conjunto de pesos de la red actual, al mismo tiempo se calcula el estado real del sistema por medio del sistema de ecuaciones (2.9). Con esto podemos calcular el error cuadrático entre la salida deseada y la proporcionada por la red (ecuación 2.10). Esta medida de error es comparada con el error de umbral, si este error es mayor que el de umbral se aplica el algoritmo de retropropagación, y si es menor seguimos con el siguiente patrón aleatorio. El proceso de entrenamiento se detiene hasta que los n patrones que se hayan generado en esa iteración se encuentren por debajo del error de umbral.

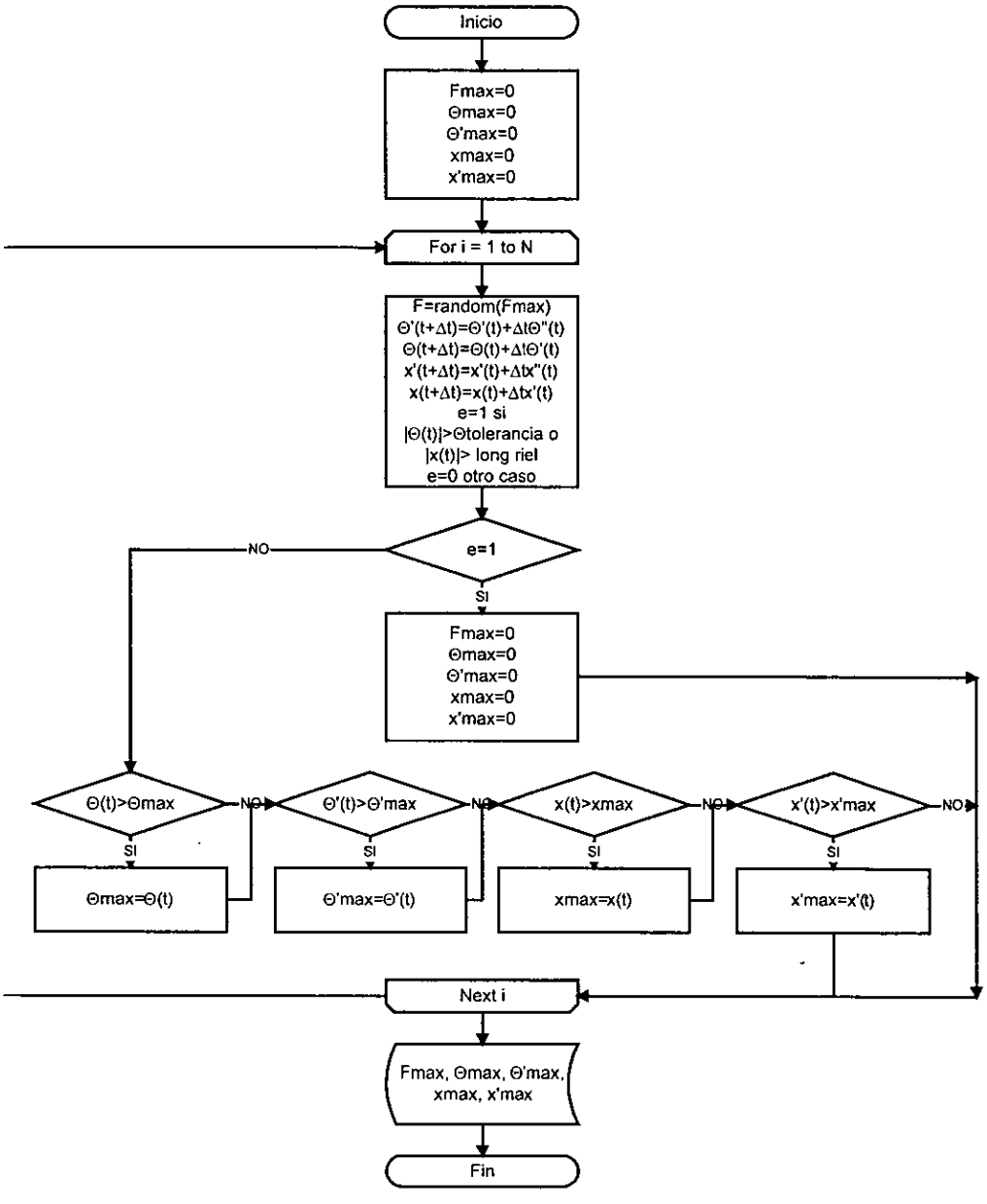


Figura 3.1 Diagrama de flujo empleado para generar el archivo de entrenamiento.

La ecuación del error cuadrático esta definida como:

$$e_r = \frac{1}{2}(s_d - s_r)^2 \tag{3.1}$$

donde e_r es el error cuadrático de la red, s_d es la salida deseada (proporcionada por el sistema de ecuaciones 2.11) y s_r es la salida proporcionada por la red neuronal.

A continuación se presentan los algoritmos empleados para la implementación y simulación del problema de control planteado.

Generación del conjunto de entrenamiento.

/*Se coloca el péndulo en el centro del riel y el ángulo de inclinación igual a cero*/

$\ddot{\theta}(t) = 0$

$\ddot{x}(t) = 0$

$\theta(t) = 0$

$\dot{\theta}(t) = 0$

$\dot{x}(t) = 0$

$x(t) = 0$

/*Se asigna el estado actual como estado máximo*/

$\dot{\theta}_{max} = \dot{\theta}(t)$

$\dot{x}_{max} = \dot{x}(t)$

$\theta_{max} = 12°$

$x_{max} = 2.4m$

F_{max} =fuerza_max

/*Se realiza el proceso n veces*/

For i=1 to terminos_de_entrenamiento

/*se calcula una fuerza aleatoria*/

F = fuerza_max × (2 × random() - 1)

$$\ddot{x} = \frac{F + m_b \ell[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta]}{m}$$

$$\ddot{\theta} = \frac{mg\sin\theta - \cos\theta[F + m_b \ell \dot{\theta}^2 \sin\theta]}{4/3\ell m - m_b \ell \cos^2\theta}$$

/*Se calcula el estado que se produce al siguiente intervalo de tiempo*/

$\dot{\theta}(t+\Delta t) = \dot{\theta}(t) + \Delta t\ddot{\theta}(t)$

$\theta(t+\Delta t) = \theta(t) + \Delta t\dot{\theta}(t)$

$\dot{x}(t+\Delta t)=\dot{x}(t) + \Delta t\ddot{x}(t)$

$x(t+\Delta t)=x(t) + \Delta t\dot{x}(t)$

/*Si el estado actual del sistema hace que la ecuación (2.10) tome valor de 1, el sistema se coloca en la posición inicial*/

if $|\theta(t+\Delta t)| > 12°$ OR $|x(t+\Delta t)| > 2.4$ Then

$\ddot{\theta}(t) = 0$

$\ddot{x}(t) = 0$

$\theta(t) = 0$

$\dot{\theta}(t) = 0$

$x(t) = 0$

$\dot{x}(t) = 0$

End if

/*se buscan los valores máximos para las variables de estado*/

If $|\theta(t)| > \theta_{max}$ then

$\theta_{max} = |\theta(t)|$

End if

If $|\dot{\theta}(t)| > \dot{\theta}_{max}$ then

$\dot{\theta}_{max} = |\dot{\theta}(t)|$

End if

If $|x(t)| > x_{max}$ then

$x_{max} = |x(t)|$

End if

If $|\dot{x}(t)| > \dot{x}_{max}$ then

$\dot{x}_{max} = |\dot{x}(t)|$

End if

End For

/*Se escriben los valores máximos al archivo de entrenamiento*/

Escribe θ_{max}, $-\theta_{max}$, $\dot{\theta}_{max}$, $-\dot{\theta}_{max}$, F_{max}, $-F_{max}$, x_{max}, $-x_{max}$, $\dot{x}_{max}$, $-\dot{x}_{max}$ a un archivo de texto

End function

Entrenamiento por retropropagación.

/*Lee los valores máximos del archivo de entrenamiento*/

Lee θ_{max}, $-\theta_{max}$, $\dot{\theta}_{max}$, $-\dot{\theta}_{max}$, F_{max}, $-F_{max}$, x_{max}, $-x_{max}$, $\dot{x}_{max}$, $-\dot{x}_{max}$ de un archivo de texto

/*Se inicializa la matriz de pesos*/

```
For i = 1 to 6
        For j to no_nodos_intermedios
                pesosint(i,j) = 2 × random -1
        Next j
Next i
For i = 1 to no_nodos_intermedios
        pesossal(i) = 2 × random -1
Next i
```

/*Se realiza el proceso hasta que los n patrones cumplan con el criterio de entrenamiento*/

```
Do
        No_error = 0
```

/*se generan y validan los n patrones de entrenamiento*/

```
        For i = 1 to terminos_de_entrenamiento
```

/*Se obtiene un estado aleatorios dentro de los limites máximos de las variables de estado*/

$F = fuerza\_max \times (2 \times random() - 1) - fuerza\_max$

$\dot{\theta} = \dot{\theta}_{max} \times (2 \times random() - 1) - \dot{\theta}_{max}$

$\theta = \theta_{max} \times (2 \times random() - 1) - \theta_{max}$

$\dot{x} = \dot{x}_{max} \times (2 \times random() - 1) - \dot{x}_{max}$

$x = x_{max} \times (2 \times random() - 1) - x_{max}$

/*Se calcula la salida de la red para la entrada y matriz de pesos actuales*/

$sal\_red = FeedForward(\dot{\theta}, \theta, x, \dot{x}, F)$

/*Se calcula la salida con el sistema de ecuaciones dinámicas*/

$$\ddot{x} = \frac{F + m_b \ell [\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta]}{m}$$

$$\ddot{\theta} = \frac{mg\sin\theta - \cos\theta[F + m_b \ell \dot{\theta}^2 \sin\theta]}{4/3\ell m - m_b \ell \cos^2\theta}$$

$\dot{\theta}(t+\Delta t) = \dot{\theta}(t) + \Delta t\ddot{\theta}(t)$

$\theta(t+\Delta t) = \theta(t) + \Delta t\dot{\theta}(t)$

$\dot{x}(t+\Delta t) = \dot{x}(t) + \Delta t\ddot{x}(t)$
$x(t+\Delta t) = x(t) + \Delta t\dot{x}(t)$
sal_deseada = θ $(t+\Delta t)$
/*Se calcula el valor del error*/
error = ((sal_deseda - sal_red) ^ 2) / 2
/*si el error es mayor que el error de umbral definido se realiza retropropagación y se incrementa la variable que cuenta el numero de patrones que no son clasificados de forma correcta*/
If error > error_threshold Then
No_error = No_error + 1
Backpropagation()
End if
End For
loop While No_error > 0
end function

Calcula la salida de la red.
FeedForward(θ ,$\dot{\theta}$,x,$\dot{x}$,F)
/*Esta función calcula la salida de la red para el estado y matriz de pesos presente*/
/*Toma los valores de entrada a la red*/
ent(1) = $\dot{\theta}$
ent(2) = θ
ent(3) = $\dot{x}$
ent(4) = x
ent(5) = F
ent(6) = 1
/*calcula la salida de cada uno de los nodos intermedios*/
for i = 1 to no_nodos_intermedios
wx = 0
for j = 1 to 6
wx = wx + pesos(i,j) × ent(j)
end for
nodo_inter(i) = 1 / (1 + exp(-wx))
end for

```
/*Se calcula la salida de la red a partir de los nodos intermedios*/
wh = 0
for j = 1 to no_nodos_intermedios
        wh = wh + pesossal(j) × nodointer(j)
end for
nodo_salida = 1 / (1 + exp(-wh))
end FeedForward
```

Se realiza el ajuste de los pesos por el algoritmo de retropropagación.

```
Backpropagation()
/*Calcula el factor de cambio para el nodo de salida*/
delta_sal = sal_deseada - sal_red
/*Calcula el factor de cambio para los nodos intermedios*/
for i = 1 to no_nodos_intermedios
        delta_inter(i) = delta_sal × pesossal(i) × sal_red × (1 - sal_red)
end for
/*Corrige los pesos de conexión de los nodos de entrada a los intermedios*/
for i = 1 to 6
   for j = 1 to no_nodos_intermedios
        wt_cambio = delta_inter(i) × ent(i) - sal_nodo_inter(j) × (1 - sal_nodo_inter(j))
        pesosint(i,j) = pesosint(i,j) + learning_rate × ((1 - alpha) × wt_cambio + alpha ×
                                momentum_int(i,j))
                momentum_int(i,j) = wt_cambio
        end for
end for
/*corrige los pesos de conexión de los nodos intermedios al nodo de salida*/
for i = 1 to no_nodos_intermedios
        wt_cambio = delta_sal × sal_nodo_inter(i) × sal_red × (1-sal_red)
        pesossal(i) = pesossal(i) + learning_rate × ((1 - alpha) × wt_cambio + alpha ×
                        momentum_sal(i))
        momentum_sal(i) = wt_cambio
end for
end Backpropagation
```

3.3 TIEMPO DE PROCESAMIENTO

Como primer paso para la obtención de la red neuronal óptima se determinará el número adecuado de patrones necesarios para entrenar una red. Una medida interesante a considerar es el número de operaciones de punto flotante que son necesarias para entrenar una red. Se considera como operación de punto flotante a las operaciones aritméticas (sumas, restas, multiplicaciones y divisiones con reales), lógicas (comparaciones entre números reales) y la generación de números aleatorios. En la sección anterior se presentó el pseudocódigo que se emplea para generar el archivo de entrenamiento, y ahí podemos ver que se emplean 41 operaciones de punto flotante por cada patrón que se genera. Por lo que para generar el archivo de entrenamiento se deben realizar :

$$np * 41 \tag{3.2}$$

operaciones de punto flotante, donde *np* es el número de patrones. La ecuación (3.2) se ilustra en la figura 3.2.

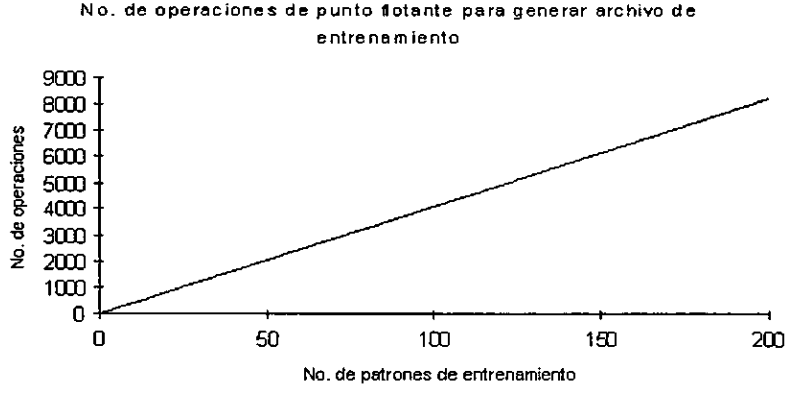


Figura 3.2 En ésta gráfica podemos ver que el número de operaciones necesarias para generar el archivo de entrenamiento crece de manera lineal dependiendo del número de patrones requeridos.

Como podemos ver en la figura 3.2, el número de operaciones de punto flotante necesarias para generar el archivo de entrenamiento crece de forma lineal y a medida en que aumenta el número de patrones de entrenamiento, el número de operaciones

realizadas por la computadora se incrementa también. De esta manera podemos advertir que es más conveniente emplear un conjunto de patrones reducido

En la sección anterior se presentó el pseudocódigo que se emplea para entrenar una red neuronal y de éste se obtiene la siguiente formula:

$$nop = \sum_{i=1}^{iter} np * (de + feed + error + (nerror_i * retro)) \qquad (3.3a)$$

donde *iter* es el número de iteraciones requeridas para entrenar a la red. *np* es el número de patrones, *de* es el número de operaciones para obtener la entrada dinámica aleatoria para la red (117 operaciones), *feed* es el número de operaciones que realiza la red para dar la salida al patrón de entrada (724 operaciones para la red seleccionada), *error* es el número de operaciones para calcular el error cuadrático (5 operaciones), $nerror_i$ es el número de patrones que presentaron error superior al de umbral en la i-ésima iteración en el conjunto de *np* patrones y finalmente *retro* es el número de operaciones que se realizan en la retropropagación (3000 operaciones). También del pseudocódigo de *Entrenamiento por retropropagación*, *Calcula la salida de la red* y *Ajuste de los pesos por el algoritmo de retropropagación* podemos obtener los siguientes valores: *de* = 117 operaciones, *feed* = 724 operaciones para la red seleccionada, *error* = 5 operaciones y finalmente *retro* = 3000 operaciones

Sustituyendo estos valores en la ecuación (3.3a) tenemos:

$$nop = \sum_{i=1}^{iter} np * (846 + nerror_i * 3000) \qquad (3.3b)$$

$nerror_i$ e *iter* se obtuvieron de forma experimental entrenando varias redes para cada número de patrones diferentes y se registro en un archivo de texto el número de patrones con error por iteración. De esta manera podemos tener varias mediciones para cada número de patrones empleados. A continuación se presenta una tabla en la que podemos ver el número de iteraciones empleadas en cada red. Es importante resaltar que el número de iteraciones y el número de patrones con error por iteración dependen en gran medida del valor aleatorio de los pesos iniciales, ya que si el método converge de manera rápida se considera que los pesos iniciales son buenos.

Número de iteraciones necesarias para entrenar una red

No. de patrones	Red 1	Red 2	Red 3	Red 4
40	31	42	49	59
50	25	33	44	51
60	38	59	60	67
70	28	30	40	56
80	49	50	54	66
90	26	44	51	67
100	60	73	77	83
110	42	55	78	80
120	29	64	75	84
150	60	68	71	73
200	51	77	100	127

Tabla 3.1 Relación entre el número de iteraciones necesarias para entrenar una red y el número de patrones empleados para ello.

Ahora bien, si sustituimos los valores anteriores, junto con el número de errores por iteración en la ecuación 3.3b tendremos el número de operaciones de punto flotante necesarias para entrenar a las redes de acuerdo al número de patrones empleados durante el entrenamiento. Es decir, se tienen cuatro diferentes valores para cada conjunto de patrones de entrenamiento que se manejaron durante la etapa de entrenamiento. Posteriormente se agruparon en una gráfica de dispersión para ver el comportamiento del número de operaciones de punto flotante necesarias para entrenar a la red.

Número de operaciones de punto flotante necesarias para entrenar una red ($x10^7$)

No. de patrones	Red 1	Red 2	Red 3	Red 4
40	2.24	3.67	4.83	5.77
50	2.67	3.72	4.24	4.01
60	6.34	9.07	12.84	13.00
70	5.51	5.2	6.15	8.81
80	9.66	8.37	10.75	14.30
90	7.67	10.9	10.63	13.58
100	18.14	14.12	20.10	24.58
110	13.65	14.72	19.51	22.92
120	10.58	18.27	21.44	22.89
150	26.02	24.79	30.94	28.25
200	37.52	47.22	45.77	60.97

Tabla 3.2. Número de operaciones de punto flotante necesarias para entrenar una red.

Esos datos se ilustran ver en la figura 3.3.

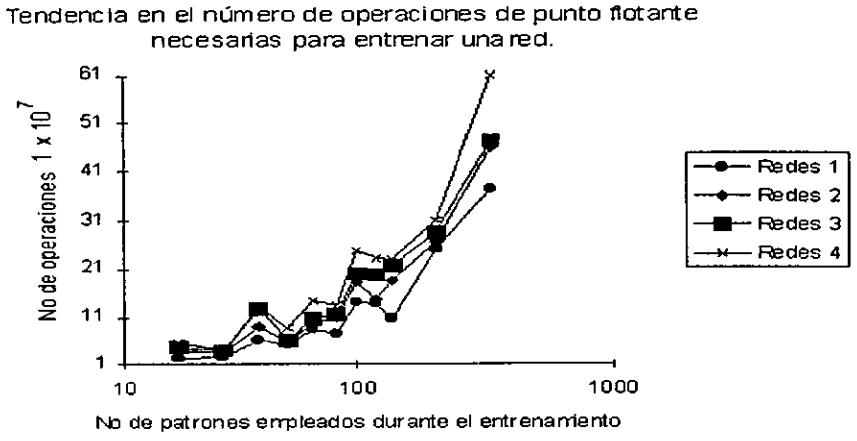


Figura 3.3. Tendencia en el número de operaciones para entrenar una red neuronal.

Como podemos observar en la figura 3.3, el número de operaciones de punto flotante necesarias para entrenar una red tiende a crecer de manera exponencial conforme aumenta el número de patrones. Con lo cual comprobamos que una red con un conjunto de entrenamiento pequeño requiere un tiempo considerablemente menor para entrenarse que la red con un conjunto de entrenamiento grande.

3.3 ENTRENAMIENTO

Para seleccionar el número de patrones de entrenamiento no sólo se debe de tomar en cuenta el tiempo de procesamiento requerido durante la etapa de entrenamiento, lo más importante para seleccionar el tamaño adecuado del conjunto de entrenamiento es el desempeño que la red nos ofrece.

Durante la fase de entrenamiento se registró el número de errores por iteración que presentaba la red para un conjunto de n patrones de entrenamiento. A continuación se establece experimentalmente el número de patrones con error por iteración para diferentes redes hasta que el método ha convergido.

El entrenamiento se realiza presentando patrones aleatorios y corrigiendo para cada patrón que no cumpla con el error de umbral. El error de umbral está definido por la ecuación 3.1 y para el entrenamiento de las redes se consideró que su magnitud es 0.01; el entrenamiento se detiene cuando los n patrones que se seleccionaron están por debajo del error de umbral.

El la figura 3.4 se ilustran los resultados obtenidos para el conjunto de redes entrenadas con cuarenta patrones de entrenamiento. En ésta figura se puede ver que el método converge en más o menos iteraciones dependiendo de la configuración aleatoria de pesos iniciales.

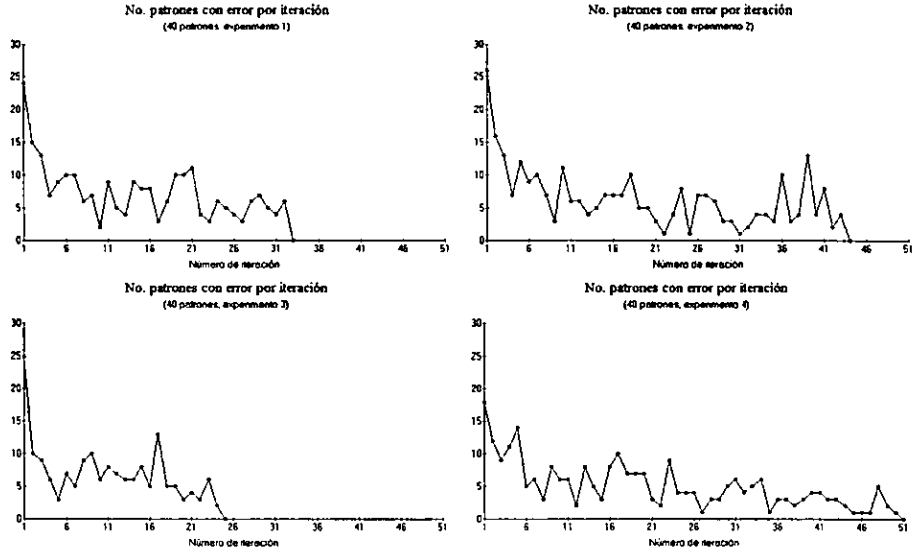


Figura 3.4 Método de entrenamiento para diferentes configuraciones de pesos iniciales (40 patrones)

3.4 VALIDACIÓN

Como mencionamos anteriormente, durante la fase de entrenamiento se generaron varias redes, variando el número de patrones de entrenamiento con el objeto de probar el comportamiento de las redes. A continuación se explora el desempeño de las redes ya entrenadas.

Debido a la forma en que se definió el controlador, cuando la red neuronal esta controlando al péndulo la magnitud proporcionada por el actuador es fija y lo que varia es el sentido de la fuerza aplicada; y cuando la red neuronal es entrenada la fuerza no se restringe a esta condición, por lo que para realizar la validación de la red neuronal se consideró fijo el ángulo de tolerancia (0.2094 rad) y se validó para varias fuerzas (desde 1 hasta 10N). Adicionalmente se consideraron diferentes condiciones de la perturbación inicial (también se emplearon fuerzas de 1 a 10N).

Las siguientes gráficas corresponden al desempeño de cada una de las redes obtenidas con 40 nodos intermedios, ya que las redes con un número menor de nodos ofrecían un desempeño muy pobre. También se muestra solamente el desempeño de las redes entrenadas con 100, 80, 60 y 40 patrones, ya que no se obtuvo una diferencia notable entre estas y las entrenadas con mayor número de patrones de entrenamiento y como se vio anteriormente se requiere menor tiempo para entrenarlas.

Desempeño de las redes entrenadas con 40 patrones

(variando la fuerza de control, la magnitud de la pertubación inicial y con un ángulo de tolerancia de 0.02094 rad)

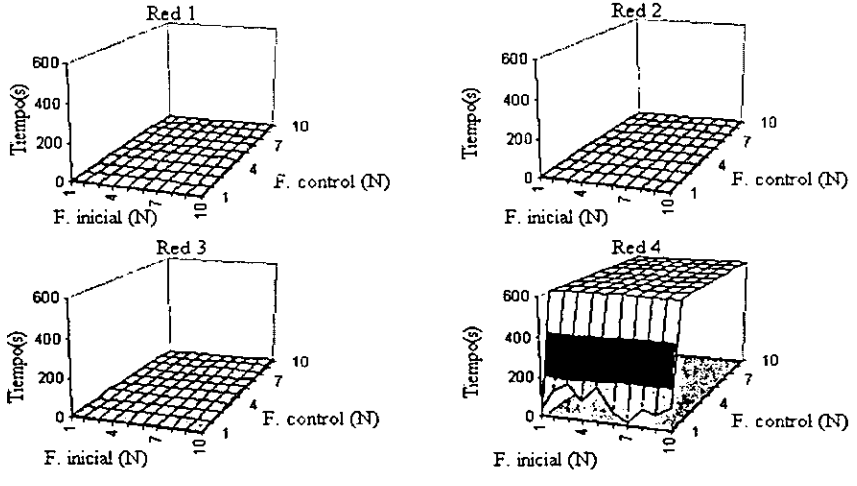


Figura 3.5 Comportamiento de las redes entrenadas con 40 patrones

Desempeño de las redes entrenadas con 60 patrones

(variando la fuerza de control, la magnitud de la pertubación inicial y con un ángulo de tolerancia de 0.02094 rad)

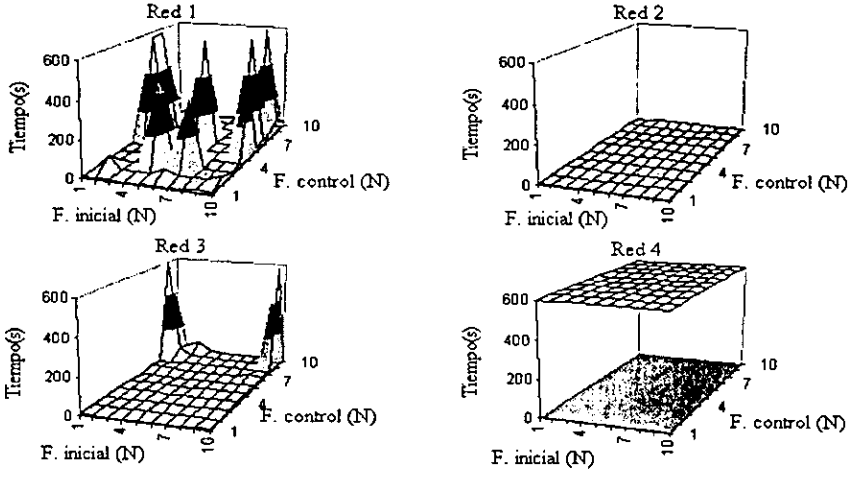


Figura 3.6 Comportamiento de las redes entrenadas con 60 patrones

Desempeño de las redes entrenadas con 80 patrones

(variando la fuerza de control, la magnitud de la pertubación inicial y con un ángulo de tolerancia de 0.02094 rad)

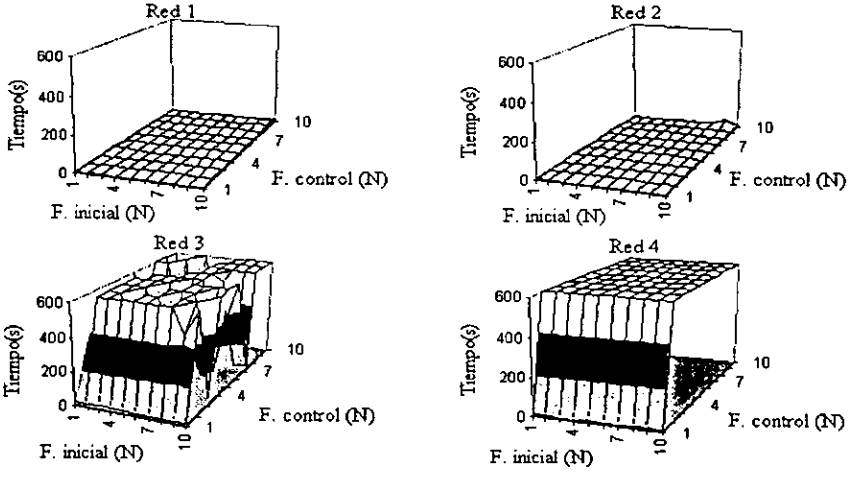


Figura 3.7 Comportamiento de las redes entrenadas con 80 patrones

Desempeño de las redes entrenadas con 100 patrones

(variando la fuerza de control, la magnitud de la pertubación inicial y con un ángulo de tolerancia de 0.02094 rad)

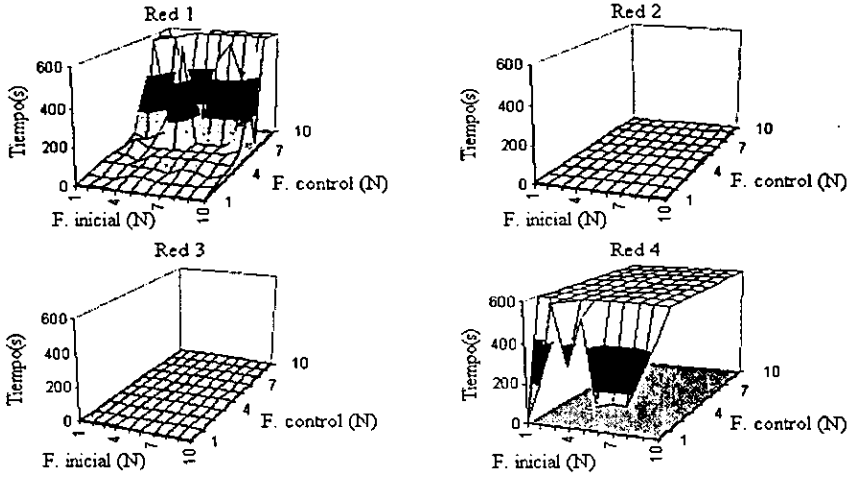


Figura 3.8 Comportamiento de las redes entrenadas con 100 patrones

Para considerar una red como bien entrenada seleccionamos aquellas que nos proporcionan un tiempo de control alto en todo el intervalo de fuerzas o en una gran parte de ese intervalo (considerando como tiempo suficiente 600s). De las gráficas anteriores podemos observar que de cada conjunto de redes entrenadas con la misma cardinalidad se desechan varias redes, ya que estas no cumplen con el desempeño esperado. Por ejemplo, en el caso de las redes de 80 patrones encontramos dos redes que nos dan aparentemente un buen desempeño, mientras que en los otros conjuntos obtuvimos sólo una red aceptable por cada uno de ellos.

En el conjunto de figuras 3.9 se muestran redes entrenadas con diferente configuración de pesos inicial, en ellas podemos ver el comportamiento del controlador y de la posición del carrito y el ángulo del péndulo con respecto al tiempo para unas redes que se consideraron como bien entrenadas y otra que no lo fue.

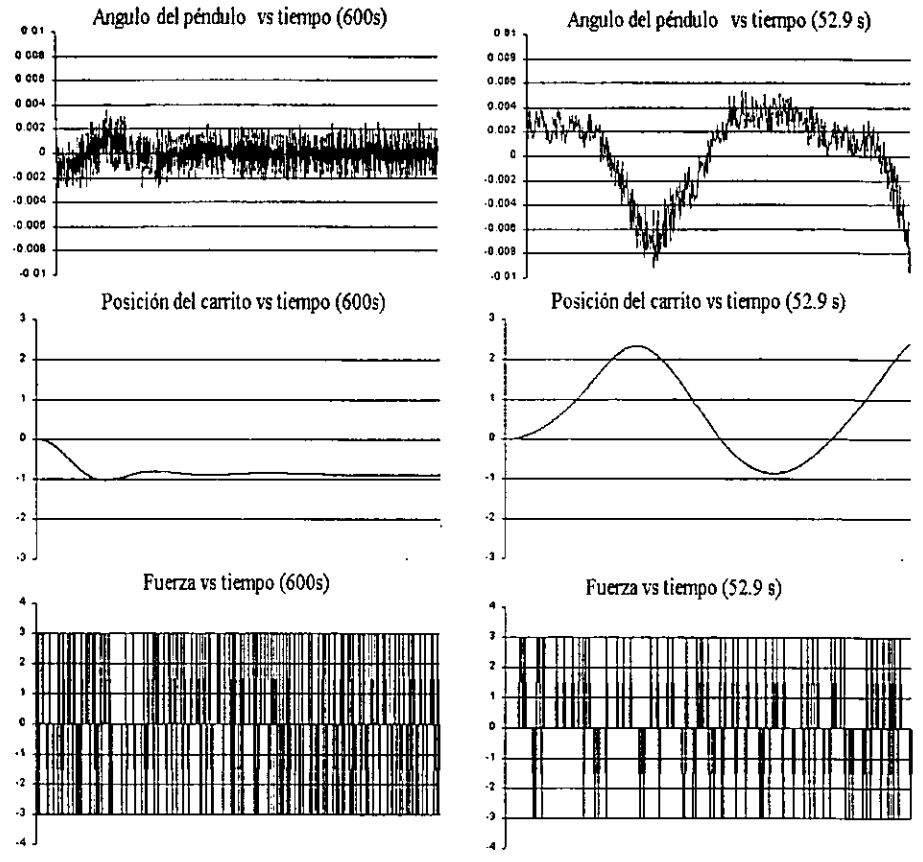


Figura 3.9a Comportamiento de las variables de estado para dos redes diferentes

En las gráficas de la figura 3.9 podemos ver el comportamiento entre diferentes redes. Las cuales fueron entrenadas a partir de diferentes conjuntos de pesos iniciales, por lo cual el conjunto de pesos resultante es diferente. En estas gráficas podemos ver redes que presentan comportamientos diferentes, por ejemplo, podemos ver una red en la cual el péndulo oscila mucho pero el carrito casi no se mueve a lo largo del riel, en otro ejemplo en carrito recorre el riel de un lado a otro y el péndulo oscila también, así como un ejemplo en el cual la red no es capaz de mantener balanceado el péndulo.

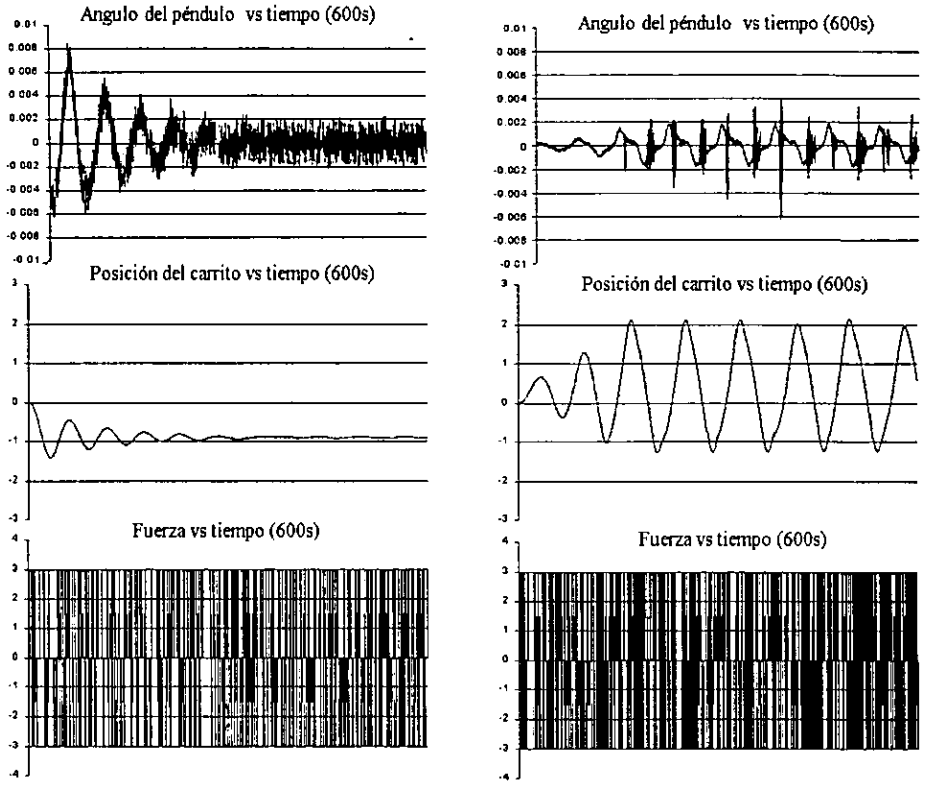


Figura 3.9b Comportamiento de las variables de estado para dos redes diferentes

3.3 GENERALIZACIÓN

Hasta ahora el desempeño de las redes entrenadas con diferente cardinalidad es similar, por lo que necesitamos hacer pruebas que nos permitan tomar una mejor selección del número de patrones de entrenamiento. Para ello hemos decidido hacer un barrido en el intervalo de la magnitud de la fuerza así como en la magnitud del ángulo de tolerancia. Para hacer estas pruebas se emplearon sólo las redes que se consideraron como buenas en la fase de validación, empleando una magnitud de 5N para la perturbación aleatoria inicial.

Las gráficas siguientes nos dan el desempeño de dichas redes.

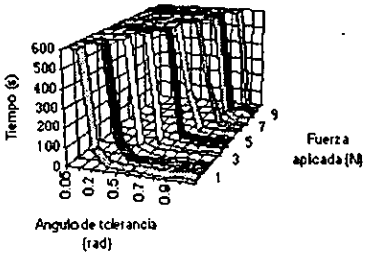


Figura 3.10 red entrenada con 40 patrones

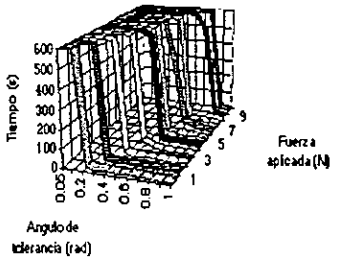


Figura 3.11 red entrenada con 60 patrones

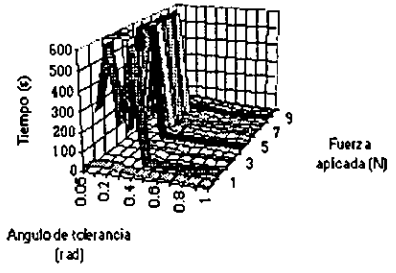


Figura 3.12 red 3 entrenada con 80 patrones

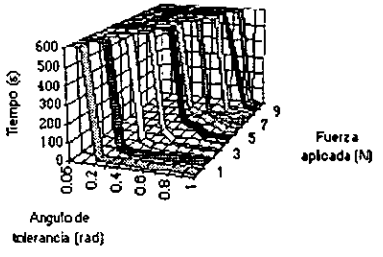


Figura 3.13 red 4 entrenada con 80 patrones

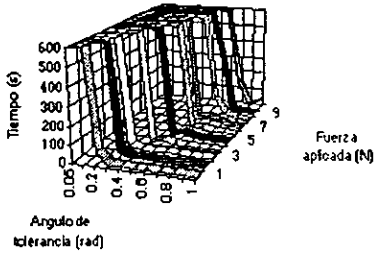


Figura 3.14 red entrenada con 100 patrones

Como podemos observar, aunque una red neuronal sea buena bajo condiciones similares a las del entrenamiento no siempre puede generalizar el desempeño a condiciones desconocidas, esto se ejemplifica en la red 3 que se obtuvo con 80 patrones, que aunque en la primera gráfica que obtuvimos de su desempeño era considerada como aceptable, cuando tratamos de generalizar notamos que su comportamiento era muy irregular y que no controlaba adecuadamente al péndulo.

A continuación presentamos una gráfica del comportamiento de las variables de estado para una misma red variando el ángulo de tolerancia para el controlador.

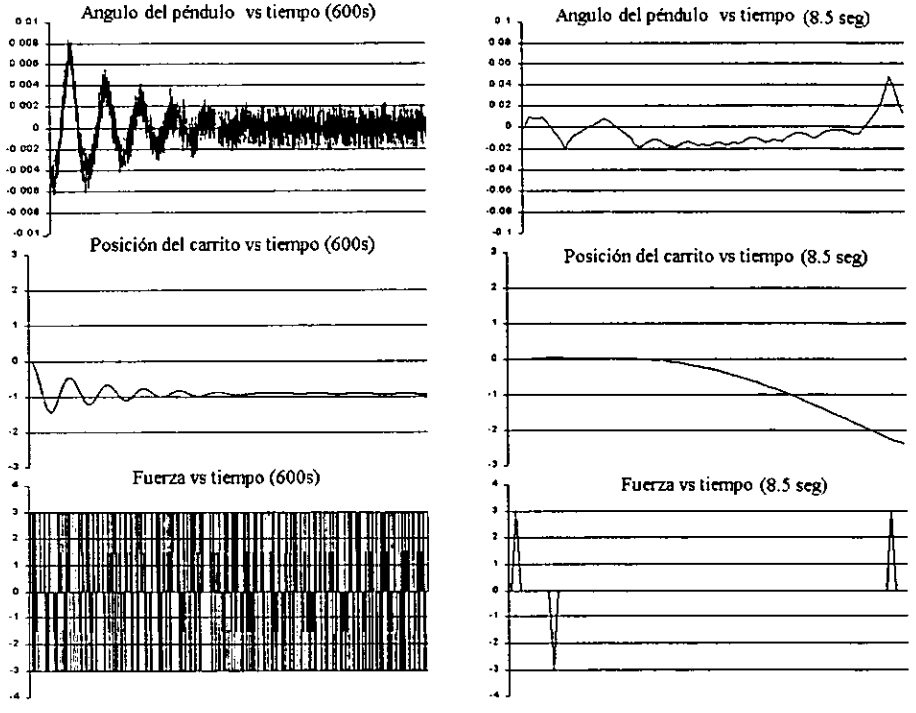


Figura 3.15 .Comportamiento de la red variando el ángulo de tolerancia

En las gráficas de la figura 3.15 podemos ver que una misma red no funciona siempre de la misma manera, ya que si modificamos el ángulo en el cual debe de responder el controlador la red ya no es funcional.

Los resultados obtenidos de las redes restantes indican que el neurocontrolador, en general, responde mejor cuando la magnitud de fuerza aplicada es grande o cuando el ángulo de tolerancia es pequeño. Estos resultados son muy parecidos entre las cuatro redes de entrenadas con diferente número de patrones. Sin embargo, el tiempo de procesamiento con el conjunto más pequeño es significativamente menor, como podemos ver en la tabla 3.2. Debido a la disminución en el tiempo de entrenamiento, además de que el comportamiento es muy similar entre las redes con un mayor número de patrones de entrenamiento justifican que en las simulaciones siguientes se empleen redes neuronales entrenadas con sólo cuarenta patrones.

Como pudimos ver de las gráficas de generalización del desempeño de las redes el comportamiento que se obtiene a magnitudes grandes es superior al que se obtiene con una magnitud pequeña. Esto se debe a que en el entrenamiento se considera una fuerza que abarca el intervalo completo de fuerzas (-10 a 10N) y al ser aplicadas de manera aleatoria no sabemos exactamente que patrones fueron presentados, pero si reducimos el intervalo de la magnitud de la fuerza esperamos reforzar el desempeño de la red a magnitudes pequeñas y con ello esperamos obtener eventualmente una red que tenga un desempeño más robusto en todo el intervalo de fuerzas.

En la siguiente gráfica se manejaron magnitudes durante el entrenamiento en el intervalo de -6 a 6N y como se puede ver el desempeño es muy similar al obtenido con la magnitud de 10N aunque en la magnitud de la fuerza igual a 1 aumento un poco el ángulo de tolerancia y el funcionamiento a las magnitudes grandes no se degradó significativamente.

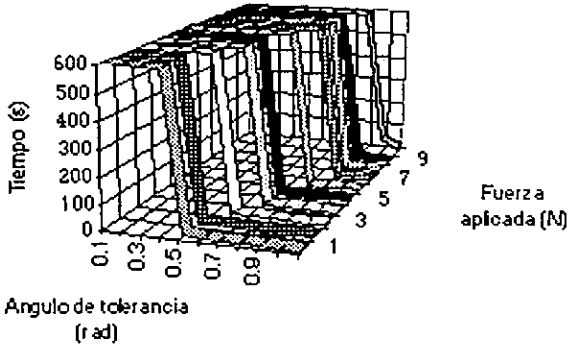


Figura 3.16 .Comportamiento de la red entrenada con fuerzas en el intervalo de –6 a 6 N

En la siguiente gráfica podemos observar el comportamiento de la red entrenada con fuerzas en el intervalo -3 a 3N. En este caso, podemos ver que el neurocontrolador presenta un mejor comportamiento para fuerzas de magnitud pequeña y ángulos de tolerancia también pequeños; como podemos ver el desempeño es casi uniforme en todo el intervalo de valores y aunque durante el entrenamiento no fueron presentados patrones en los que se consideraran fuerzas de magnitud mayor a 3N, la red neuronal fue lo suficientemente apta para responder adecuadamente ante estas situaciones.

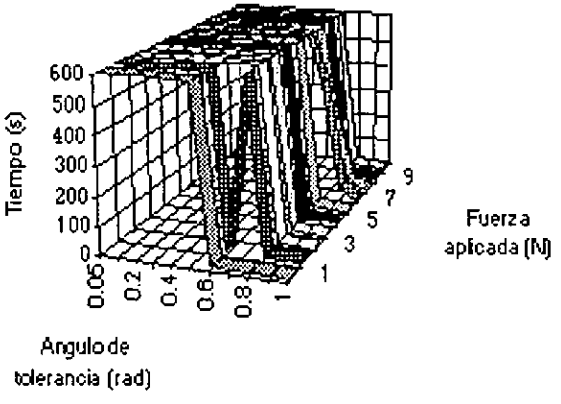


Figura 3.17. Comportamiento de la red entrenada con fuerzas en el intervalo –3 a 3 N

Sin embargo, si durante el entrenamiento de la red se consideran fuerzas de magnitud muy pequeña (-1 a 1N), el comportamiento del neurocontrolador se degrada, es decir, ya no responde adecuadamente a las fuerzas de magnitud grande, ya que el ajuste de los pesos de la red no le permitió generalizar a tales magnitudes de fuerza.

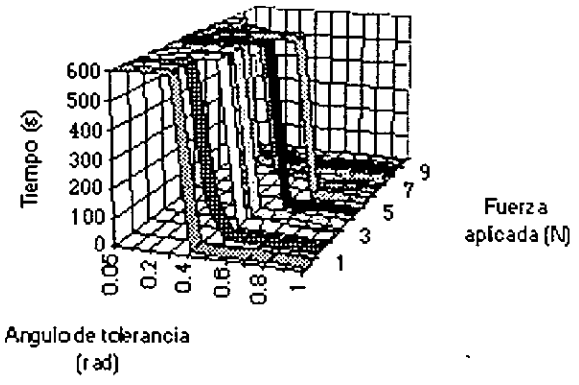


Figura 3.18 Comportamiento de la red entrenada con fuerzas en el intervalo –1 a 1 N

Para llegar obtener estas gráficas de desempeño se realizó nuevamente el proceso de entrenamiento y validación de varias redes a las cuales se modificaron las condiciones de entrenamiento. Una vez que se seleccionaron las redes funcionales en la etapa de validación se realizo el proceso de generalización para poder obtener las gráficas mostradas anteriormente.

Es importante recalcar que se realizó el mismo proceso descrito anteriormente para redes con diferente número de nodos intermedios, Se generó el proceso para redes con 10, 15, 20, 25, 30, 35 y 40 nodos intermedios presentándose en todos ellos la misma tendencia exponencial al momento de crecer el número de patrones de entrenamiento. Pero aquí sólo se presentan los resultados que se obtuvieron con redes con 40 nodos intermedios debido a que finalmente ese fue el número de nodos intermedios con el que funcionó adecuadamente la red.

CONCLUSIONES

En el capítulo 3 se presentaron algunos de los aspectos más importantes a considerar en el entrenamiento de una red neuronal. Como primer punto se midió el tiempo de procesamiento para entrenar una red neuronal, ya que es importante generar una red neuronal funcional en el menor tiempo posible.

Para determinar el tiempo de procesamiento requerido para entrenar una red neuronal se debe de considerar el tamaño del conjunto de entrenamiento, el tamaño de la red neuronal, el criterio de convergencia y el método de entrenamiento.

Como pudimos observar, entre mayor sea el conjunto de entrenamiento se realizan más operaciones de punto flotante. Si hacemos la comparación del conjunto mayor con respecto al conjunto menor, en promedio tarda diez veces más que el menor. Por eso debemos de tratar que el tamaño del conjunto de entrenamiento sea lo menor posible y que éste conjunto refleje las características significativas del sistema que se este aproximando para que al realizar el entrenamiento la red funcione adecuadamente.

También es deseable que el tamaño de la red sea pequeño, el número de nodos de entrada y de salida están definidos por el sistema, y por lo tanto son fijos, pero el número de nodos intermedios es determinado por el diseñador de la red. Aunque no existe una forma estricta para determinar el número de nodos intermedios, algunos autores recomiendan comenzar con pocos nodos intermedios, entrenar y probar la red, si no funciona adecuadamente se incrementa el número de nodos intermedios y se repite el proceso hasta que la red responde adecuadamente. En este trabajo el primer entrenamiento se realizó con diez nodos intermedios y se fue incrementando hasta llegar a cuarenta. Con cuarenta nodos intermedios obtuve resultados aceptables con la red, y ya no aumente el número de nodos.

Otro parámetro que interviene en el número de operaciones de punto flotante es el valor aleatorio de los pesos iniciales, ya que si con ésta configuración el método converge en pocas iteraciones, se dice que la configuración inicial de pesos es buena. Este es un parámetro que no es controlado por el diseñador, ya que nunca sabemos si un conjunto de pesos iniciales es bueno, sino hasta después de realizar el entrenamiento.

El criterio de convergencia está determinado por el diseñador de la red neuronal, ya que puede ser como en éste caso, que el entrenamiento se detiene cuando todos los patrones estén por debajo del error de umbral o cuando el error promedio sea el que este por debajo del error de umbral, además de que el error de umbral se puede reducir tanto como se quiera. Por otro lado el ajuste de la matriz de pesos se puede hacer por ejemplo o por epoch, es decir a cada presentación de un patrón o al finalizar un recorrido por todos los patrones de una iteración.

Todos estos parámetros mencionados en los párrafos anteriores influyen tanto en el tiempo de procesamiento, como en el desempeño de la red neuronal, y no existe una regla que nos diga que combinación de todos éstos parámetros nos dará una buena red neuronal, aunque como pudimos ver en el capítulo 3, el tamaño del conjunto de entrenamiento no es tan importante como el que éste sea representativo de la población.

Una vez que hemos entrenado una red neuronal debemos probarla, es decir verificar que realice correctamente la función para la que fue adaptada, una forma de hacerlo es presentar los patrones de un conjunto de validación previamente separados, y en este caso particular la validación se realiza poniendo a funcionar a la red neuronal como controladora del sistema físico bajo condiciones parecidas a las del entrenamiento. Como podemos ver en las gráficas obtenidas en el capítulo 3 (figuras 3.4 a 3.7) no todas la redes que cumplieron el criterio de convergencia en la fase de entrenamiento resultan aptas para controlar al péndulo invertido y por lo tanto no podemos poner a funcionar una red neuronal inmediatamente después de que ha sido entrenada, ya que debemos comprobar primero que ha sido ajustada correctamente.

Una característica importante de las redes neuronales es la capacidad de generalizar, es decir, la capacidad de responder adecuadamente a situaciones que no fueron presentadas en la fase de entrenamiento o de validación. En esta caso, la generalización se estudio variando la magnitud del ángulo de tolerancia del péndulo

con respecto a la vertical y también la magnitud de la fuerza de control. Con esto se le dio un poco más de juego al péndulo y se hizo más difícil el control. Pudimos comprobar que varias de las redes respondieron muy bien a estas condiciones desconocidas y otras no lo hicieron.

Hasta ahora se ha hablado en general de las redes neuronales, pero a continuación se hará una breve revisión de los resultados obtenidos con el controlador.

Todo el proceso de entrenamiento, validación y generalización, se realizó para redes con diferentes características. Primero se fue modificando el número de nodos intermedios, hasta que se ajusto a cuarenta nodos intermedios. En el trabajo no se presentan estos resultados, ya que con redes con menos nodos nunca obtuve un controlador aceptable. Posteriormente se entrenaron redes variando en número de patrones de entrenamiento y como se explicó en el capítulo de resultados al final se eligió entrenar más redes con sólo cuarenta patrones de entrenamiento. Y por último se entrenaron más redes variando la magnitud de la fuerza de entrenamiento, se realizó el entrenamiento para redes con magnitud máxima de la fuerza de 10N que fue con las que se validaron las redes anteriores, después con realizo el mismo procedimiento empleando una magnitud máxima de 6, 3 y 1N. Los resultados que obtuvimos con la fuerza de entrenamiento de 10N muestran que el controlador neuronal es muy sensible a fuerzas de magnitud pequeña, para las cuales no resulta ser un buen controlador. Sin embargo, para fuerzas de magnitud grande, el controlador mantiene vertical al péndulo y dentro del riel por periodos de tiempo muy largos, aún para tolerancias grandes. Sin embargo pudimos notar que es de vital importancia el elegir el conjunto de entrenamiento adecuado, ya que como se puede ver en la figura 3.13 el comportamiento es parecido al obtenido con las redes de 10N (figuras 3.8 a 3.12), sin embargo tiende a mejorar un poco en la región que fallaba anteriormente. El la figura 3.14, la red entrenada con 3N proporciona un comportamiento casi uniforme en todo el intervalo de fuerzas aplicadas y en la figura 3.15 se observa que los patrones que se le presentaron a la red no fueron suficientes para poder controlar adecuadamente al sistema y la parte que anteriormente funcionaba bien ya no lo hace, aunque en la región de fuerzas pequeñas el desempeño es aceptable.

PROSPECTIVAS

El presente trabajo ha estudiado algunas de las propiedades de las redes neuronales y ha mostrado que el proceso de entrenamiento es más que un simple algoritmo de ajuste. Pero aún quedan muchas cosas que pueden ser estudiadas posteriormente. Algunas de las cosas que se pueden hacer son variar el periodo de muestreo del sistema físico, que pasa si aumenta o disminuye, si las redes entrenadas siguen funcionando o es necesario realizar un nuevo entrenamiento. También podemos cambiar la estructura del conjunto de entrenamiento, es decir que sea el mismo para todas las iteraciones que realice el método antes de alcanzar el criterio de convergencia, podemos mover el error de umbral, realizar el ajuste de los pesos por epoch en lugar de como de hace actualmente, que es por ejemplo. Esto es sólo para la parte de diseño de la red, y no son todas las posibilidades de estudio del sistema empleado, esto puede crecer tanto como se desee.

APÉNDICE A

Manejo del Sistema Simulador

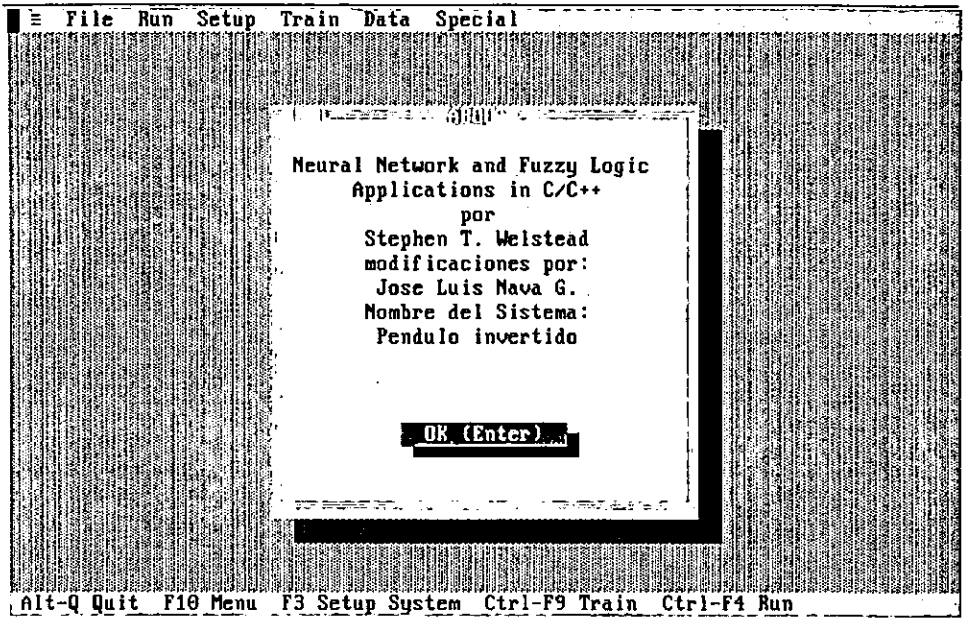
El sistema simulador fue tomado del libro *Nueral Networss and Fuzzy Logic Applications in C/C++* de Stephen T. Welstead, dicho sistema tiene la capacidad de entrenar redes neuronales tipo perceptrón multicapa a partir un conjunto de entrenamiento, el sistema puede modificar el número de nodos en cada una de las capas y los parámetros relativos al entrenamiento; además de contar con una animación simple del sistema considerado para el estudio.

En general el sistema original de Welstead cumple con los requerimientos para generar las simulaciones deseadas, pero se le hicieron pequeñas modificaciones para poder tener los valores del error de la red (eq 3.1) obtenido para cada uno de los patrones en todas las iteraciones necesarias para entrenar a la red, además de poder modificar algunos parámetros del sistema que eran fijos.

A continuación se presenta un pequeño manual de como funciona el sistema.

Para iniciar el sistema es necesario estar en MS-DOS y teclear lo siguiente:
C:> FFBRM

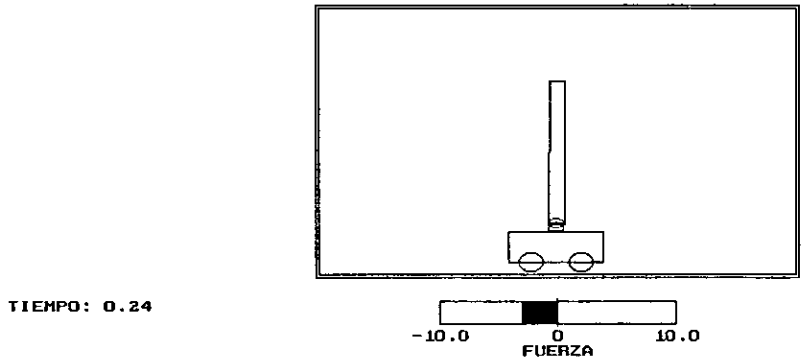
A continuación aparecerá la siguiente pantalla.



Después de Mostrar los créditos aparecerá un menú principal con varias opciones.

En el menú de Archivo podemos cambiar el nombre del archivo en el cual se escribirán los pesos de la red, así como la opción de salir.

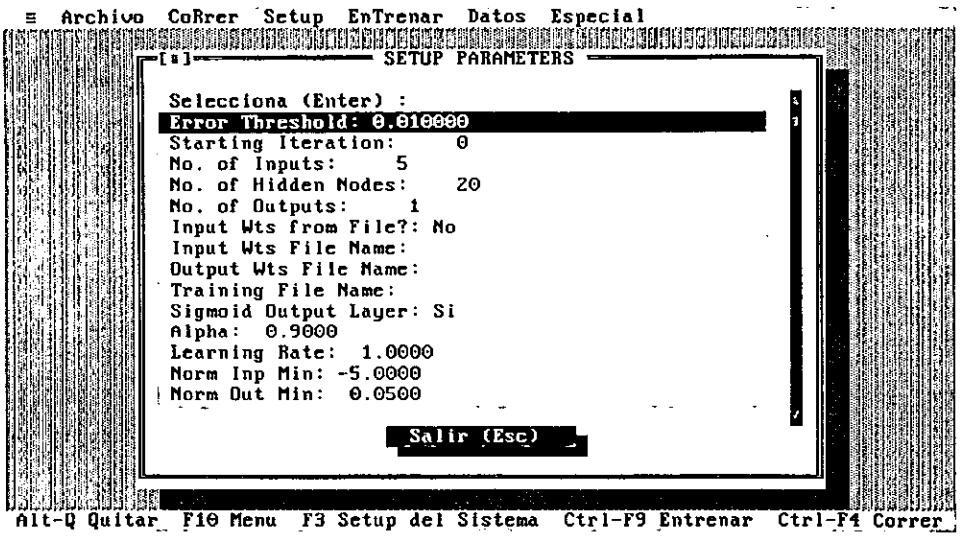
En la opción de correr se ejecutará la simulación del sistema; en ésta opción podemos guardar en un archivo el comportamiento del sistema físico (tanto en el ángulo, como en la posición de carrito) y durante la simulación aparece la animación como se muestra en la siguiente figura.



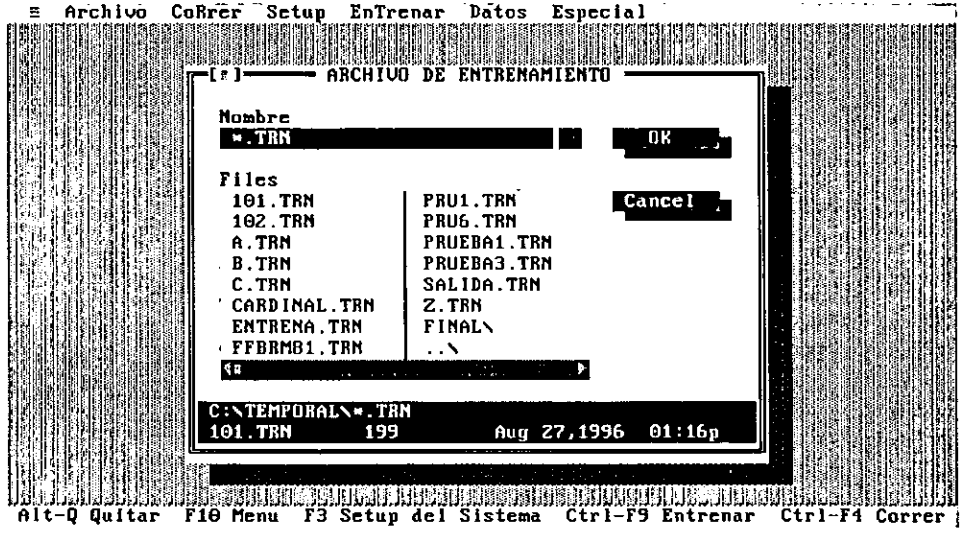
En el menú de Setup se ajustan los parámetros necesarios para generar una red.

Dentro de ésta opción podemos abrir una configuración de parámetros previamente creada y que se ha guardado en un archivo, guardar una configuración en un archivo, imprimir una configuración o entrar a modificar estos parámetros.

Cuando entramos a la opción de Setup de Sistema aparecerá la siguiente pantalla, en la cual podemos modificar los parámetros de la red y del entrenamiento.



Cuando deseamos dar el nombre de entrada o de salida del archivo de pesos o el nombre del archivo de patrones de entrenamiento, aparece la siguiente caja de diálogo, en la cual podemos indicar el nombre del archivo deseado.



En la opción de entrenar podemos generar un archivo de entrenamiento para este sistema en especial, el cual es generado de acuerdo al planteamiento del capítulo 3, en ésta opción podemos definir los valores máximos de la fuerza empleada para generar el entrenamiento.

En este menú también se puede entrenar una red. Cuando entrenamos una red el sistema nos pregunta si deseamos guardar los valores del error de la red y en caso de aceptar no pide el nombre del archivo en el cual se guardarán dichos valores. Así mismo nos pide la fuerza máxima para realizar el entrenamiento.

En el menú de Datos se encuentran los valores del sistema para realizar las simulaciones, los valores que se fijan en esta opción son los de la magnitud de la fuerza correctiva (que es fija), el ángulo de tolerancia y la magnitud de la fuerza aleatoria de inicio.

En la opción de Especial se realiza la simulación del sistema pero teniendo al usuario como controlador, al estar simulando el sistema la fuerza correctiva es proporcionada por el usuario mediante las flechas de navegación (derecha-izquierda).

BIBLIOGRAFÍA

Stephen T. Welstead
Nueral Networss and Fuzzy Logic Applications in C/C++
John Wiley & sons, Inc. 1994

Michael A. Arbid Editor
The Handbook of Brain Theory and Neural Networks
Ed. Advisory Board. 1995 The MIT PRESS
Massachusetts institute of Technology

Bernard Widrow, Fellow, IEEE y Michael A. Lehr
30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation
Proceedings of IEEE, Vol 78, No. 9, September 1990

Charles W. Anderson
Learning to Control an Inverted Pendulum Using Neural Networks
American Control Conference, Atlanta Georgia, June 15-17, 1988.

Charles W. Anderson
Strategy Learning with Multilayer Conectionist Representations
Tech Rept TR87-509.3, GTE Laboratories Inc., Waltham, 1987.

Jian-Kang Wu
Neural Networks and Simulation Methods.
in "The Handbook of Brain theory and Neural Networks"
Ed. Advisory Board. 1995 The MIT PRESS
Massachusetts institute of Technology

Timothy Masters
Practical Neural Networks Recipes in C++
Ed Academic Press Inc. 1993.

Cichocki, A. Unbehauen, R.
Neural Networks for Optimization and Signal Processing
Ed. Jhon Wiley & Sons Ltd. England. 1993

Werbos, Paul John

The Roots of Backpropagation From Ordered Derivateives to Neural Networks and Political Forecasting

A Wiley-Interscience Publication, John Wiley & Sons Inc. N.Y. U.S.A. 1994

Rosenblatt, Frank
On the Convergence of Reinforcement Procedures in Simple Perceptrons
Cornell Aeronautical Laboratory Report VG-1196-G4, Buffalo, NY, Feb 1960

Tou, J. T. and Gonzalez, R. C.
Pattern Recognition Principles

Reed, Russell and Marks II, Robert J.
Neurosmithing: Improving Neural Network Learning
in "The Handbook of Brain theory and Neural Networks"
Ed. Advisory Board. 1995 The MIT PRESS
Massachusetts institute of Technology

Smith, Murray
Nerual Networks For Statical Modeling
Van Nostrand Reinhold , New York 1992