Control de un Robot mediante Computación Neuronal

Lucas Gamero, Carlos Mércuri, Leonardo Rufiner y Daniel Zapata

Facultad de Ingeniería-Bioingeniería
Universidad Nacional de Entre Ríos

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

Control de un Robot mediante Computación Neuronal

Lucas Gamero, Carlos Mércuri, Leonardo Rufiner y Daniel Zapata

Facultad de Ingeniería-Bioingeniería, Universidad Nacional de Entre Ríos

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

RESUMEN

Los métodos utilizados para el diseño de estrategias de control basados en las técnicas convencionales, involucran la construcción de modelos matemáticos que describen la dinámica del sistema a controlar, y la aplicación de métodos analíticos a los mismos para derivar las leyes de control. Cuando se pretende controlar plantas no lineales, normalmente se realiza una linealización de las características del proceso en las proximidades de la zona de operación y se procede al diseño del controlador utilizando la teoría de sistemas lineales. Este enfoque puede imponer severas limitaciones a la implementación de tales sistemas, restándole generalidad y garantizando su validez sólo para no linealidades específicas. Por otra parte, las técnicas convencionales generalmente fallan cuando es difícil de obtener un modelo suficientemente representativo del sistema real. Esta situación se presenta cuando existe incertidumbre, el sistema es muy complejo, o viola las hipótesis en que se sustentan las técnicas de síntesis de estrategias de control.

Un controlador neuronal lleva a cabo un control adaptativo, aprendiendo la dinámica de la planta. Este control toma la forma de una red no lineal multicapa cuyos parámetros adaptables son los pesos de conexión entre las neuronas. De alguna manera, la red aprende a controlar la planta interactuando con ella, sin necesidad de un conocimiento *a priori* de las características de la misma.

En el presente trabajo describimos nuestra experiencia en el abordaje del problema del control de un brazo robot utilizando redes neuronales anteroalimentadas. El manipulador modelado consta de tres juntas rotacionales con movimiento en el plano cartesiano. En el modelo se considera, además, fricción viscosa y de Coulomb y una perturbación en los torques. Para el control clásico, se utiliza un controlador proporcional derivativo (PD). En el caso del control neuronal, se aplican técnicas de estimación de parámetros mediante redes neuronales (RN) y se implementa el controlador mediante una RN.

Con este propósito realizamos la simulación en computadora de todos los modelos que utilizamos durante el desarrollo y la evaluación del sistema: modelo neuronal de la planta-robot, controlador neuronal y simulador animado del brazo controlado. El entrenamiento de los bloques neuronales y su optimización fue realizada utilizando la asistencia de sistemas desarrollados a tal fin por el grupo.

I INTRODUCCIÓN

Los métodos utilizados para el diseño basados en las técnicas de control convencionales, involucran la construcción de modelos matemáticos que describen la dinámica del sistema a controlar, y la aplicación de técnicas analíticas a los mismos para derivar las leyes de control. Estas técnicas convencionales generalmente fallan cuando es difícil de obtener un modelo representativo del sistema real. Esta situación se presenta cuando existe incertidumbre, el sistema real es muy complejo, o viola las hipótesis en que se sustentan las técnicas de síntesis de estrategias de control.

En los últimos años ha ido incrementándose el interés por los sistemas basados en redes neuronales aplicados a la resolución de problemas de control. Tal incremento se debe principalmente a la posibilidad de aplicarlos con éxito allí donde las estrategias convencionales no han demostrado ser eficientes. Una distinción entre la forma en que las redes neuronales procesan la información y el modo en que lo hacen los sistemas de control convencionales es la capacidad de procesamiento colectivo que les proporciona a aquellas la posibilidad de responder rápidamente a entradas sensoriales complejas. Por otro lado, los algoritmos de control sofisticados están severamente limitados por el tiempo necesario para ejecutarlos. Una tercera distinción, y quizás la más importante, es que el control realizado por un sistema neuronal se adquiere a través de un aprendizaje, mientras que en los controladores clásicos es necesario especificar, *a priori*, un algoritmo. Por lo tanto, para implementar un controlador algorítmico eficaz se debe poseer un conocimiento suficiente de la planta a controlar, lo que es difícil de lograr en la práctica.

Un controlador neuronal realiza una forma de control adaptativo en la cual el controlador es una red no lineal de neuronas dispuestas en varias capas, y los parámetros adaptables son los pesos de las interconexiones entre las neuronas. Este tipo de controladores posee las siguientes características fundamentales: capacidad de procesamiento colectivo, adaptación e inmunidad al ruido, y da lugar a un control más eficiente cuando se presentan incertezas, complejidad y no linealidades.

Cuando no se conocen los parámetros de la planta a controlar es posible utilizar diversas técnicas de estimación de los mismos. Las redes neuronales también pueden ser aplicadas a esta tarea, y son especialmente adecuadas cuando se pretende usar otra red neuronal como controlador. En el presente trabajo nos proponemos modelar el control de un robot basándonos en técnicas de redes neuronales y compararlas con las técnicas clásicas aplicadas al mismo robot.

El artículo esta organizado de la siguiente manera: en la sección II se introduce el tema de redes neuronales, en particular las utilizadas en nuestro trabajo; en la sección III se describen las distintas arquitecturas de control mediante redes

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

neuronales. En la sección IV se aborda el tema de la identificación de los parámetros de la planta, utilizando redes neuronales y en la sección V se presenta el modelo del robot utilizado. En la sección VI se muestran las simulaciones realizadas y sus resultados. Por último, en la sección VII se presenta una breve conclusión.

II. REDES NEURONALES ANTEROALIMENTADAS

En una red anteroalimentada (feedforward) la salida de cualquier neurona nunca puede dar lugar a una retroalimentación consigo misma, en forma directa ni indirecta a través de otras neuronas. Esta forma de interconexión determina que la salida presente no influya sobre las futuras y las neuronas de las capas más bajas no reciban influencia de lo que ocurre en las más altas. En este tipo de redes, el cómputo de la respuesta se completa en un solo paso. Cuando un pattern se presenta a los terminales de entrada de la red, las neuronas de la primera capa computan los valores de salida y los pasan a la capa siguiente. Cada capa recibe entonces valores provenientes de la capa previa, calcula los valores de salida y los pasa a la siguiente. El proceso termina cuando se calculan los valores de salida de la última capa. Esta regla sólo puede ser violada durante la fase de entrenamiento, cuando la salida de una neurona se utiliza para ajustar los pesos de conexión entre las neuronas de capas contiguas, afectando así las futuras salidas de esa neurona. Un ejemplo de este tipo de estructura es el perceptrón multicapa, que será el tipo de red utilizado en este trabajo.

En una red neuronal anteroalimentada no hay interconexiones entre las neuronas de una misma capa. Sin embargo, cada neurona de una capa proporciona una entrada a cada una de las neuronas de la capa siguiente, como se observa en la Figura 1.

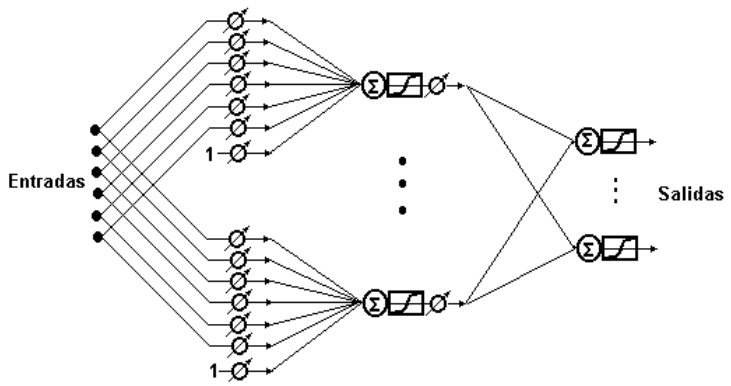


Figura 1: Esquema de una red Neuronal Anteroalimentada

La cantidad de neuronas en la capa de entrada es igual al número de características de cada pattern a ser procesado, y cada una de las neuronas recibe una de éstas. Las salidas de las neuronas de la última capa constituyen la salida de la red. La cantidad

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

de neuronas en las capas ocultas, normalmente queda a criterio del diseñador de la red. Demasiadas neuronas tenderán a que la red opere simplemente reconociendo los ejemplos del entrenamiento, dando lugar a que la red tenga dificultades para reconocer nuevos tipos de patrones de entrada realizando generalizaciones. La utilización de pocas neuronas, por otra parte, no le permitirán a la red hacer la cantidad suficiente de representaciones internas para mapear el espacio de entrada en el espacio de salida [Lip87].

A. Perceptrones multicapa.

Los perceptrones de capa múltiple son redes anteroalimentadas con una o más capas de neuronas entre las neuronas de entrada y salida. Estas capas adicionales contienen neuronas o unidades ocultas que no están directamente conectadas a las neuronas de entrada y salida.

Los perceptrones multicapa poseen una función no lineal a la salida de cada neurona y utilizan varias capas adaptables. No solamente los pesos de una capa son adaptables, sino que lo son los de todas las capas que forman la red. Se puede demostrar que un perceptrón multicapa es capaz de aproximar tan ajustadamente como se desee cualquier mapeo entre un vector dado de dimensión M y cualquier otro de dimensión N [Gal90, Shi90].

Las neuronas utilizadas en los perceptrones multicapa son del tipo Mc Culloch-Pitts, con una función sigmoide (continua, monótona creciente, continuamente diferenciable y con asíntotas horizontales para valores finitos) para la cual, los valores asintóticos son generalmente 1 y 0. Un ejemplo de una función sigmoide es

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

donde θ es un umbral y x es la suma de las entradas pesadas para cada neurona. En este trabajo, las funciones de activación para las neuronas de la capa de salida son lineales, debido a que, en la práctica, no es posible acotar en forma arbitraria las salidas de la planta y del controlador .

Un perceptrón multicapa adecuado para el reconocimiento de patrones complejos requiere normalmente tres capas, como mínimo: una capa de entrada, una capa oculta y una capa de salida. Puede demostrarse que no se requieren más que tres capas en redes tipo perceptrón anteroalimentado para generar regiones de decisión arbitrariamente complejas, aunque por razones de tipo práctico puede ser útil trabajar con más capas ocultas [Gal90]. En nuestro trabajo utilizamos dos capas ocultas en la estructura de las redes.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

B. Entrenamiento Supervisado

Hay dos operaciones distintas que tienen lugar durante la fase de entrenamiento: el cómputo hacia adelante (feedforward computation) y la actualización de los pesos basándose en el error de salida de la red. Los perceptrones utilizan entrenamiento supervisado, ya que se le deben presentar a la red los patterns de entrada y sus correspondientes salidas para ser asociados.

Cómputo hacia adelante.

El proceso de cómputo hacia adelante consiste en la presentación de un pattern de entrada a las neuronas de la primera capa, las cuales transfieren el valor a la primera capa oculta. Cada una de las neuronas de esta capa oculta calcula una suma pesada de sus entradas, pasa la suma a través de su función de activación y presenta el valor obtenido a la siguiente capa. Cada neurona en la capa de salida también calcula una suma pesada de sus entradas y pasa la suma a través de su función de activación, dando lugar a uno de los valores de salida de la red. Normalmente, cada neurona de la red usa la misma función de activación y el mismo valor umbral, aunque esto no es una condición necesaria. En nuestro trabajo hemos experimentado utilizando diferentes funciones de activación y umbrales para cada una de las capas, a fin de mejorar el rendimiento de la red [HuH90, HuH91].

Algoritmo de Retropropagación

El algoritmo de retropropagación es una generalización del algoritmo de cuadrados medios mínimos (LMS). Este es una técnica de búsqueda por gradiente para minimizar una función costo igual a la diferencia cuadrática media entre la salida deseada y la salida real de la red.

La red es entrenada seleccionando inicialmente pequeños pesos aleatorios y umbrales internos, y luego presentando los datos de entrenamiento en forma sucesiva. Una vez que han sido computados los valores de salida, son comparados con las salidas deseadas a fin de generar los valores de error con los que se ajustan los pesos de la red. Los pesos son ajustados después de cada intento utilizando información externa que especifica la clase correcta hasta que los pesos converjan y la función costo sea reducida a un valor aceptable [Mel89].

Un componente esencial del algoritmo es el método iterativo que propaga los términos de error requeridos para adaptar los pesos de los neuronas que vuelven de la capa de salida a los de las capas más bajas. El error para la j-ésima neurona de la capa de salida es

$$E_j = (d_j - y_j)\, f'(red_j) \qquad (2)$$

donde dj es la salida deseada, yj es la salida computada, fj' es la derivada de la función de activación y redj es la suma de las entradas pesadas, a la neurona. La ecuación de actualización para la salida de la j-ésima neurona es

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

$$\mathbf{W}_j^{nuevo} = \mathbf{W}_j^{viejo} + \frac{\beta E_j \mathbf{X}}{\|\mathbf{X}\|^2} \tag{3}$$

donde **W**j es el vector de pesos para la j-esima neurona de la capa de salida, **X** es el vector de entrada a la capa de salida, y ß es el coeficiente de aprendizaje. Para la neurona i-esima de una capa oculta, la ecuación de actualización de pesos es

$$\mathbf{W}_i^{nuevo} = \mathbf{W}_i^{viejo} + \frac{\beta\left[f'(red_i)\sum_j (w_{ij}E_j)\right]\mathbf{Y}}{\|\mathbf{Y}\|^2} \tag{4}$$

donde $\mathbf{W}_i$ es el vector de pesos en la i-esima neurona de la capa oculta, **Y** es el vector de entrada, wij es el peso de conexión entre la neurona i-ésima de la primera capa oculta a la neurona j-ésima de la capa siguiente, Ej es el error de la salida para la j-ésima neurona de la capa de salida y la sumatoria se realiza sobre todas las neuronas de la capa de salida.

Cuando se usan redes neuronales aplicadas a problemas de control, normalmente las neuronas de la capa de salida no se afectan por una función sigmoide ya que ésta posee asíntotas horizontales que la acotan en magnitud y las señales de control no deben ser condicionadas *a priori*. Lo más adecuado es aplicar una función lineal, la cual permite recorrer sin restricciones el espacio de estados. Para la función sigmoide que se aplica a las neuronas de las capas ocultas se verifica que

$$f'(x) = f(x)\,1 - f(x) \tag{5}$$

y para el caso en el que las salidas están entre 0 y 1 la derivada tendrá un máximo para x=1/2, o sea cuando una neurona dada no ha sido aún forzada a tomar valor 0 o 1.

En algunas oportunidades, el algoritmo de backpropagation puede converger a un mínimo local en lugar de hacerlo al mínimo absoluto. A fin de evitar un mínimo local y mejorar así el rendimiento del algoritmo se utiliza un término de momento de tal modo que la ecuación (3), para la capa de salida toma la forma

$$\mathbf{W}_j^{nuevo} = \mathbf{W}_j^{viejo} + \frac{\beta E_j \mathbf{X}}{\|\mathbf{X}\|^2} + \alpha(\mathbf{W}_j^{nuevo} - \mathbf{W}_j^{viejo})_{previo} \tag{6}$$

El segundo término del segundo miembro de esta ecuación, es el término de momento. Este término suma una parte del cambio más reciente en los pesos cuando se están calculando los nuevos valores de las conexiones. La constante α determina la influencia que tienen dichos cambios en la corrección actual. El objetivo es

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

proporcionar a la neurona cierta "inercia", permitiéndole superar los mínimos locales.

III. ARQUITECTURAS NEURONALES PARA CONTROL

Pueden considerarse tres métodos diferentes de aprendizaje: el método indirecto, el general y el especializado [Psa88, Nar90].

A. Aprendizaje indirecto

En un controlador anteroalimentado implementado como una red neuronal, su salida **u** acciona la planta; la salida deseada de la planta es **d** y su salida real es **y**. El controlador neuronal debería actuar como la inversa de la planta, produciendo una señal **u**, a partir de la respuesta deseada **d** que opere sobre la planta para que sea **y**≈ **d**. El objetivo del aprendizaje es seleccionar los valores de los pesos de la red a fin de producir un mapeo correcto **d**→**u**, al menos sobre el rango de **d** en el cual opera la planta.

Suponiendo que el controlador anteroalimentado ha sido entrenado exitosamente, de tal modo que **y**≈**d,** entonces la red utilizada como controlador anteroalimentado reproducirá aproximadamente la entrada de la planta desde **y** (**t**≈**u**). De este modo, la red puede ser entrenada para minimizar el error $\boldsymbol{\varepsilon_1}$**=u-t** utilizando la arquitectura mostrada en la figura 2, ya que si el error **ε**=**d-y** tiende a 0, también lo hará $\boldsymbol{\varepsilon_1}$. Una ventaja de esta disposición es que la red puede entrenarse en sólo las regiones de interés, ya que se comienza con la respuesta deseada **d** y todas las demás señales son generadas a partir de ésta. Además, es ventajoso adaptar los pesos para minimizar el error directamente en la salida de la red. Desafortunadamente, este método no es un procedimiento de entrenamiento válido, dado que la minimización de $\boldsymbol{\varepsilon_1}$ no garantiza la minimización de **ε**. A pesar de ello, puede ser utilizado en forma conjunta con otras estrategias para minimizar el error **ε**.

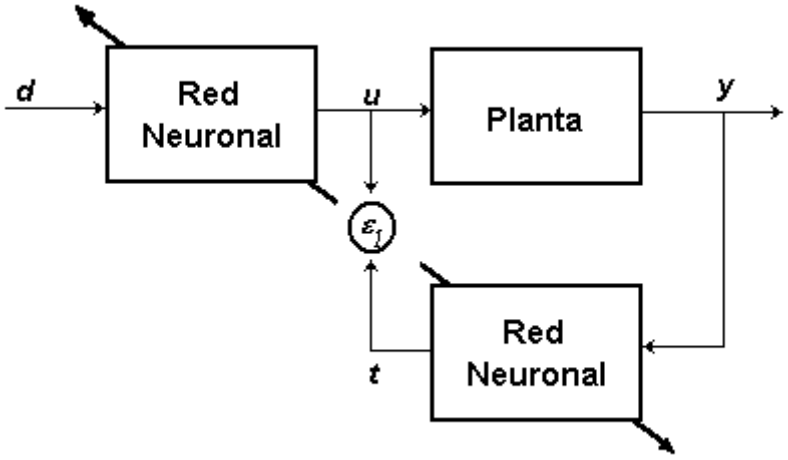


Figura 2: Arquitectura de aprendizaje indirecto

B. Aprendizaje General

En la figura 3 se presenta esquemáticamente un método para entrenar controladores neuronales que minimicen el error ε^2. Se selecciona la entrada **u** y es proporcionada a la planta para obtener la salida **y**. La red se entrena para reproducir a **u** como salida a partir de la entrada **y**. La red así entrenada tendría la capacidad de tomar una entrada deseada **d** y proporcionar un valor apropiado de **u**. Esto funcionará si los valores de entrada **d** son suficientemente cercanos a los valores de **y** que la red utilizó durante el entrenamiento. De esta manera, la red debe generalizar para responder a entradas para las cuales no ha sido entrenada. En esta configuración, la red no puede ser entrenada selectivamente para responder correctamente en regiones de interés ya que normalmente no conocemos las entradas **u** que proporcionen la salida deseada **d**. De esta manera, el método general no puede ser eficiente ya que la red debe aprender las respuestas para un dominio operacional del sistema mayor que el realmente necesario. Una posible solución a este problema es combinar el método general con el procedimiento especializado, que se describe a continuación.

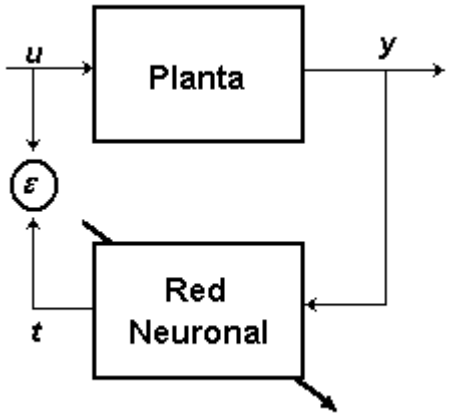


Figura 3: Arquitectura de aprendizaje general

C. Aprendizaje Especializado

En la figura 4 se muestra una arquitectura para el entrenamiento de un controlador neuronal que opere adecuadamente sólo en regiones de especialización. El entrenamiento se lleva a cabo utilizando la respuesta deseada **d** como entrada a la red. La red se entrena entonces para hallar la entrada a la planta **u** que lleve la salida del sistema y al valor deseado **d**. Esto se realiza usando el error entre la respuesta real y la deseada para ajustar los pesos de las redes mediante un método de gradiente descendente; durante cada iteración los pesos son ajustados para decrementar máximamente el error. Este procedimiento requiere conocer el jacobiano de la planta.

A diferencia del método general, que debe ser entrenado off-line, el procedimiento especializado puede ser entrenado on-line mientras el sistema está realizando su trabajo. El procedimiento especializado puede presentar problemas de estabilidad

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

durante el entrenamiento on-line, si la velocidad de aprendizaje no se adecua a las constantes de tiempo del sistema.

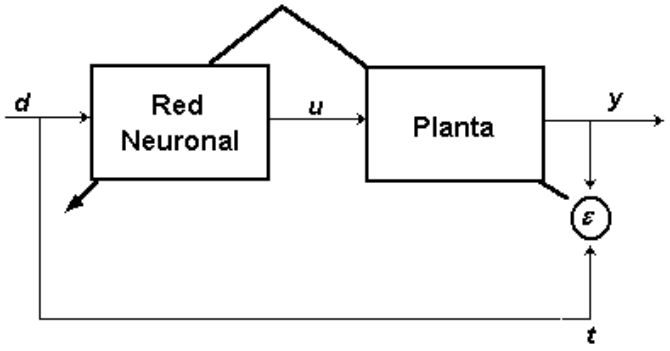


Figura 4: Arquitectura de aprendizaje especializado

D. Entrenamiento de las redes

El método de entrenamiento que se elija puede afectar el funcionamiento final del sistema, su capacidad de adaptación y el tiempo de aprendizaje.

En las arquitecturas de aprendizaje indirecto y general se aplica el algoritmo de retropropagación en forma directa, de acuerdo a las expresiones vistas en la sección II. Sin embargo, no es posible aplicar en forma directa este procedimiento a las arquitecturas para el aprendizaje especializado, debido a que el error se toma a la salida de la planta. Como se ve en la figura 4, puede pensarse a la planta como una capa adicional de la red, pero que no puede ser modificada. El error total **ε**=**d**-**y** es retropropagado a través de la planta usando las derivadas parciales de la planta en su punto de operación.

$$\delta_a^n = f^{n'}(p_a^n)\sum_i \delta_i^p \frac{\partial \mathrm{P}_i}{\partial u_a} \tag{7}$$

$$\delta_a^p = d_a - y_a$$

donde $P_i(u)$ representa el i-ésimo elemento de la salida de la planta para una entrada **u**. De este modo puede aplicarse el algoritmo de retropropagación, y la minimización del error sigue siendo del tipo gradiente descendente.

Si se desconoce la función de transferencia de la planta, es posible aproximar sus derivadas parciales según:

$$\frac{\partial P_i}{\partial u_j} \approx \frac{P_i(\mathbf{u} + \Delta u_j i_j) - P_i(u_j)}{\Delta u_j} \tag{8}$$

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

Otra alternativa es utilizar una red entrenada para que simule el comportamiento de la planta. Este último enfoque es el utilizado en el presente trabajo.

E. Aprendizaje especializado y generalizado

Un alternativa para combinar estos métodos, es realizar primero un entrenamiento general para aprender en forma aproximada el comportamiento de la planta y luego utilizar el especializado para realizar un ajuste fino de la red en el rango de operación en régimen del sistema. De esta manera, los pesos obtenidos por el método general son luego utilizados como condiciones iniciales del especializado, logrando una mayor velocidad de convergencia. Otra ventaja de utilizar primero el aprendizaje general, es que la red puede adaptarse más fácilmente frente a virtuales cambios del punto de operación, o ante el agregado de nuevos puntos de operación.

La diferencia entre los aprendizajes general y especializado reside en que en cada caso se minimizan distintas funciones de error; de esta manera estos métodos siguen diferentes caminos para alcanzar el mínimo.

Si se adopta una estrategia de alternancia entre estos dos métodos, es posible escapar de un mínimo local producido por un método entrenando la red con el otro. Específicamente, realizando un aprendizaje general previo al especializado se pueden proporcionar mejores condiciones iniciales para el procedimiento especializado, ya que el error de partida es pequeño.

IV IDENTIFICACIÓN DE PARÁMETROS

Cuando se trabaja con sistemas basados en representaciones discretas del tiempo, es posible utilizar para su representación las ecuaciones en diferencias correspondientes a las ecuaciones diferenciales que describen al sistema en una representación de tiempo continuo. Estas ecuaciones en diferencias toman una forma general

$$\begin{aligned} x(k+1) &= \Phi\ x(k), u(k) \\ y\ (k)\ &= \Psi\ x(k) \end{aligned} \tag{9}$$

donde u(.), x(.) y y(.) son secuencias temporales discretas. Si el sistema descrito por las ecuaciones anteriores es lineal y temporalmente invariante, las ecuaciones que gobiernan su comportamiento pueden expresarse como

$$\begin{aligned} x(k+1) &= Ax(k) + Bu \\ y(k) &= Cx(k) \end{aligned} \tag{10}$$

donde **A**, **B** y **C** son matrices de n×n, n×p y m×n, respectivamente. El sistema queda entonces parametrizado por la terna {**C**,**A**,**B**}. La teoría de sistemas lineales

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

temporalmente invariantes donde se conocen **A**, **B** y **C** está muy desarrollada, y conceptos tales como controlabilidad, estabilidad y observabilidad han sido estudiados en extenso para ser aplicados a ese tipo de sistemas. Del mismo modo, son también conocidos varios métodos para determinar la entrada de control u(.) optimizando un determinado criterio de performance. La forma de abordar estos problemas con este tipo de enfoque, se reduce a hallar la solución de un sistema de n ecuaciones lineales con n incógnitas. En contraste con esto, los problemas que involucran ecuaciones no lineales donde se conocen las funciones Φ y Ψ, dan lugar a ecuaciones algebraicas no lineales para las cuales no existe un soporte matemático adecuado para su resolución.

Cuando las funciones Φ y Ψ o las matrices **A**, **B** y **C** son desconocidas, nos vemos enfrentados al problema de la identificación de un sistema desconocido.

La entrada y la salida de una planta con dinámica causal, de tiempo discreto son u(.) y y(.) respectivamente. La planta se asume como estable, con una parametrización conocida, pero con valores desconocidos de los parámetros. El objetivo es construir un modelo de identificación adecuado, de tal modo que cuando sea sometido a una entrada u(k), produzca una salida $\hat{y}_p$(k) que se aproxime a la salida de la planta, y_p(k) sometida a la misma entrada.

Como nuestro interés es presentar una metodología general que refleje situaciones reales, en las que se desconoce el modelo de la planta, durante el desarrollo del trabajo suponemos desconocerlo . Sólo se utiliza dicho modelo para su simulación digital (debido a que no se cuenta en este caso con el robot real).

A. El problema de control

La representación standard de una planta de tiempo discreto y dimensión finita es mostrada en la figura 5. El vector u(k) representa las entradas a la planta en el tiempo k y el vector z(k) representa el estado de la planta en el tiempo k. La función A(z(k),u(k)) mapea las entradas y el estado actual en el estado siguiente. Cuando la planta es lineal, la ecuación se convierte en:

$$z(k+1) = \mathbf{A}(z(k), u(k)) = \mathbf{F}z(k) + \mathbf{G}u(k) \qquad (11)$$

donde **F** y **G** son matrices. La función **A**(z(k),u(k)) sería no lineal si la planta fuese no lineal.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

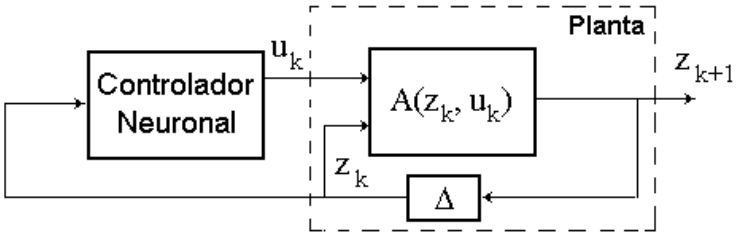


Figura 5: Diagrama de la planta con un controlador neuronal

Un problema común en control es proveer el vector correcto de entradas de manera de conducir una planta no lineal desde un estado inicial a un estado subsecuente deseado z(d). La aproximación típica utilizada para resolver este problema implica linealizar la planta alrededor de un número de puntos de operación, mediante la construcción de modelos lineales de espacio de estados de la planta en estos puntos de operación, y luego construyendo el controlador. Para plantas no lineales, esta aproximación es en general computacionalmente intensiva y requiere un esfuerzo de diseño considerable.

El objetivo de este trabajo es entrenar un controlador, en este caso una red neuronal, para producir la señal correcta u(k) para conducir la planta a un estado deseado z(d), dado el estado actual z(k) de la planta (Fig. 5). Cada valor de u(k) en el tiempo juega su papel en la determinación del estado de la planta. Conociendo el estado deseado, sin embargo, no puede extraerse fácilmente información acerca de los valores de u(k) que se requieren para alcanzarlo.

Se dispone de un número de enfoques diferentes para el entrenamiento del controlador. Entre ellos podemos mencionar control inverso y control óptimo. La arquitectura y el algoritmo de entrenamiento usan una red neuronal en control óptimo, entrenando al controlador para maximizar una función de performance (ídem minimizar una función de costo).

B. Entrenamiento del simulador neuronal de la planta

Antes de proceder al entrenamiento del controlador, se entrena otra red neuronal de tal modo que se comporte como la planta real ante el mismo conjunto de entradas. Específicamente, la red se entrena para simular el modelo dinámico del robot.

En la teoría de control, el entrenamiento del simulador persigue el mismo objetivo que la identificación de una planta . La diferencia radica en que la identificación, en este caso, se efectúa automáticamente por medio de una red neuronal que es capaz de modelar plantas no lineales.

En el presente trabajo asumimos que los estados de la planta son directamente observables sin ruido. Se procede entonces a construir una red que posee tantos

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

nodos de salida como variables de estado y tantas entradas como (variables de estado + entradas a la planta).

El número de capas en la red y la cantidad de neuronas en las capas ocultas, se determina empíricamente, ya que estos valores dependen del grado de no linealidad y del orden de la planta.

En la figura 6 se muestra un diagrama de bloques del método usado para la identificación del modelo del robot. El proceso de entrenamiento comienza con la planta en un estado inicial y luego se le proporcionan entradas generadas aleatoriamente. En el tiempo *k*, se le presenta a la red un par de entradas, que corresponden al estado actual de la planta z_k y a su salida real u_k.

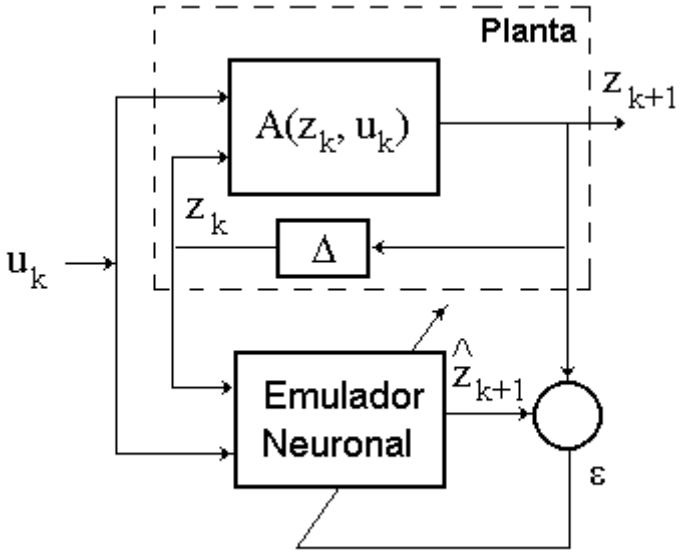


Figura 6: Entrenamiento de la RN para estimar el comportamiento de la planta

La red neuronal se entrena mediante el algoritmo de retropropagación a fin de predecir el próximo estado de la planta a partir del valor z_{k+1} utilizado como valor deseado de salida durante el proceso de entrenamiento. Este proceso se parece en algo al que efectuaría un diseñador humano para identificar la planta, sólo que en este caso es llevado a cabo automáticamente por una red neuronal.

Dado que el emulador posee ahora la misma dinámica que la planta, es posible utilizarlo con el propósito de entrenar al controlador neuronal. El controlador debe aprender a llevar a la planta, desde un estado inicial z_0 hasta un estado deseado z_d en k pasos de tiempo. El objetivo del proceso de aprendizaje es hallar un conjunto de pesos de conexión para el controlador, tales que minimicen la función error **J**, donde **J** está promediada sobre el conjunto de estados iniciales z_0

$$\mathbf{J} = E\left(\left\|d(x) - Y(x)\right\|^2\right) \qquad (12)$$

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

El proceso de entrenamiento comienza con el simulador de la planta fijado en un valor inicial aleatorio z_0. Debido a que el controlador neuronal está aún sin entrenar, dará una señal de control u_0 errónea al simulador. Este se desplazará al próximo estado z_1, y el proceso continuará durante K pasos temporales hasta llegar al estado z_k. En este punto es posible modificar los pesos en la red controladora de tal manera que el error cuadrático $(z_d - z_k)2$ sea cada vez menor.

Para entrenar la red controladora es necesario conocer el error de la salida de la misma u_k para cada paso k. Lamentablemente, sólo se dispone del error en el estado final de la planta $(z_d - z_k)$. Sin embargo es posible retropropagar este error a través de la red simuladora de la planta utilizando las ecuaciones (3) y (4) a fin de obtener un error equivalente para el controlador en la K-ésima etapa. Este error puede ser utilizado entonces para entrenar al controlador mediante la ecuación (4), lo cual hace que -en cierto sentido- el simulador traslade el error del estado final de la planta al error en la salida del controlador. La planta real no puede ser utilizada para esta tarea ya que no es posible propagar ningún error a través de ella. Esta es la razón por la que se necesita un simulador neuronal de la planta.

El entrenamiento continúa mediante el algoritmo de retropropagación hasta que el ajuste de los pesos de la red controladora determine salidas cuyo error sea menor que un valor prefijado. Debido a que el algoritmo de entrenamiento es esencialmente un método de gradiente descendente, la aparición de mínimos locales en la función error puede conducir a soluciones subóptimas.

Este es el caso más general, donde no se conoce un modelo apropiado de la planta. En el caso de conocerse la estructura de este modelo, es posible aplicar algún algoritmo tradicional de identificación de parámetros; luego se puede utilizar el jacobiano del modelo estimado para invertir la planta y obtener las entradas u_k para entrenar el controlador neuronal o resolver el jacobiano en forma numérica.

V MODELO DEL ROBOT UTILIZADO

En esta sección se describen las características del robot, los parámetros de junta y la ecuación del modelo dinámico utilizado. Las ecuaciones de los modelos cinemático y cinemático inverso se describen en el apéndice B y los términos del modelo dinámico en el apéndice C.

A. Manipulador

El modelo del robot utilizado se muestra en la figura 7. Consiste en un manipulador de tres juntas rotacionales, con un pie de altura L, dos brazos L_1 y L_2 y un órgano terminal L_3. El espacio de trabajo está limitado a un disco en el plano cartesiano. No se impone límite mecánico al movimiento de las juntas. Se consideran masas puntuales concentradas en las juntas m_1, m_2, m_3.

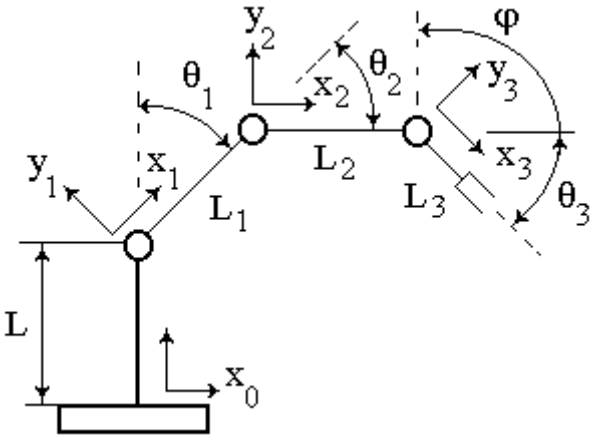


Figura 7: Robot utilizado en las simulaciones.

TABLA I

PARÁMETROS DE LAS JUNTAS DEL MANIPULADOR UTILIZADO.

-	**0**	**1**	**2**	**3**
α	0	0	0	0
a	L	L1	L2	L3
d	0	0	0	0
θ	θ1	θ2	θ3	0

B. Modelo Dinámico

Las ecuaciones del modelo dinámico se obtienen aplicando la ecuación de Lagrange

$$F_i(t) = \frac{d}{dt}\left(\frac{\partial K}{\partial \dot{q}_i}\right) - \frac{\partial K}{\partial q_i} + \frac{\partial U}{\partial q_i} \tag{13}$$

donde $F_i(t)$ es la fuerza o torque de la i-ésima junta, q_i es la coordenada de la i-ésima junta, K es la energía cinética y U es la energía potencial.

La ecuación general del modelo dinámico es

$$\tau = \mathbf{M}(\theta)\ddot{\theta} + \mathbf{B}(\theta)\dot{\theta}\dot{\theta} + \mathbf{C}(\theta)\dot{\theta}^2 + \mathbf{g}(\theta) + \mathbf{F}(\theta,\dot{\theta}) \tag{14}$$

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

donde **M**, **B** y **C** son matrices 3×3 que representan los términos de inercia, Coriolis y centrífugos respectivamente, **g** y **F** son vectores 3×1 de los términos de gravedad y fricción respectivamente. Aplicando el Lagrangiano a cada una de las juntas, se obtienen los términos del modelo dinámico, que se encuentran en el apéndice C.

VI SIMULACIÓN Y RESULTADOS OBTENIDOS

La simulación se llevó a cabo mediante programas de computación en lenguaje Turbo Pascal. Los módulos principales desarrollados son los siguientes:

- Modelo dinámico del robot a lazo abierto, donde se generan los valores entrada-salida utilizados para entrenar al simulador neuronal.

- Modelo dinámico del robot a lazo cerrado, donde se aplican las técnicas del control clásico. En este caso se implementa un controlador PD.

- Módulo de simulación de la planta: este programa genera un archivo de los datos de entrenamiento con los valores de I/O de la planta real en base a su simulación numérica.

- Módulo de entrenamiento: entrena una red neuronal tipo perceptrón con los datos de salida/entrada leídos del archivo generados mediante la simulación de la planta. Este programa genera un informe con los resultados del entrenamiento y otro con la estructura, pesos y umbrales de la red entrenada. Aquí se implementa el algoritmo de backpropagation y durante el entrenamiento se ajustan los pesos y umbrales.

- Módulo de simulación de la red entrenada: utiliza los pesos generados por el módulo de entrenamiento con los datos generados por el módulo de simulación de la planta. Se simula la red con estos pesos para entradas tipo delta, ruido blanco, sinusoidal, cuadrada, etc.

- Modelo dinámico del robot a lazo cerrado, donde se aplican las técnicas de control neuronal.

A. Control Clásico

En esta sección se analiza el enfoque clásico usado en este trabajo para el control del robot. La trayectoria deseada en el espacio cartesiano, se genera a partir de los puntos inicial y final; se impone que las variables en el espacio de las juntas, sigan una trayectoria suave. Para ello se utiliza un polinomio cúbico.

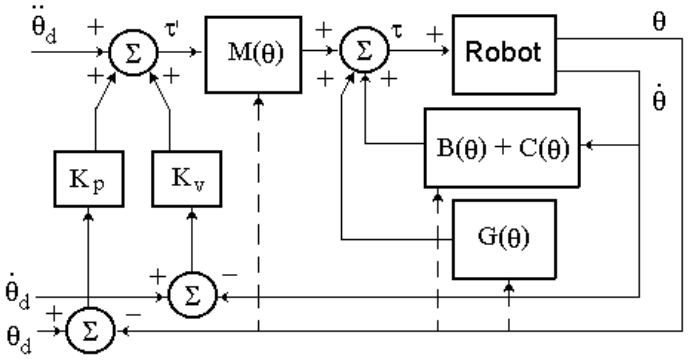


Figura 8: Diagrama de bloques del sistema de control

En la figura 8 puede verse un diagrama de bloques del sistema de control utilizado. Se implementa un controlador PD con constantes $\mathbf{K}_p$ y $\mathbf{K}_V$ de posición y velocidad respectivamente.

La ecuación del modelo dinámico es

$$\tau = \mathbf{M}(\theta)\ddot{\theta} + \mathbf{B}(\theta)\dot{\theta}\dot{\theta} + \mathbf{C}(\theta)\dot{\theta}^{2} + \mathbf{g}(\theta) + \mathbf{F}(\theta,\dot{\theta}) \qquad (15)$$

y la ley de control es

$$\tau' = \ddot{\theta}_d + K_v \dot{E} + K_p E \qquad (16)$$

donde el error está dado por

$$E = \theta_d - \theta \qquad (17)$$

Ejemplo 1

En las figuras 9.a y 9.b, se observan los errores de posición y velocidad de las tres juntas. La dinámica del robot se aproxima a un sistema sobreamortiguado y los errores tienden rápidamente a cero.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

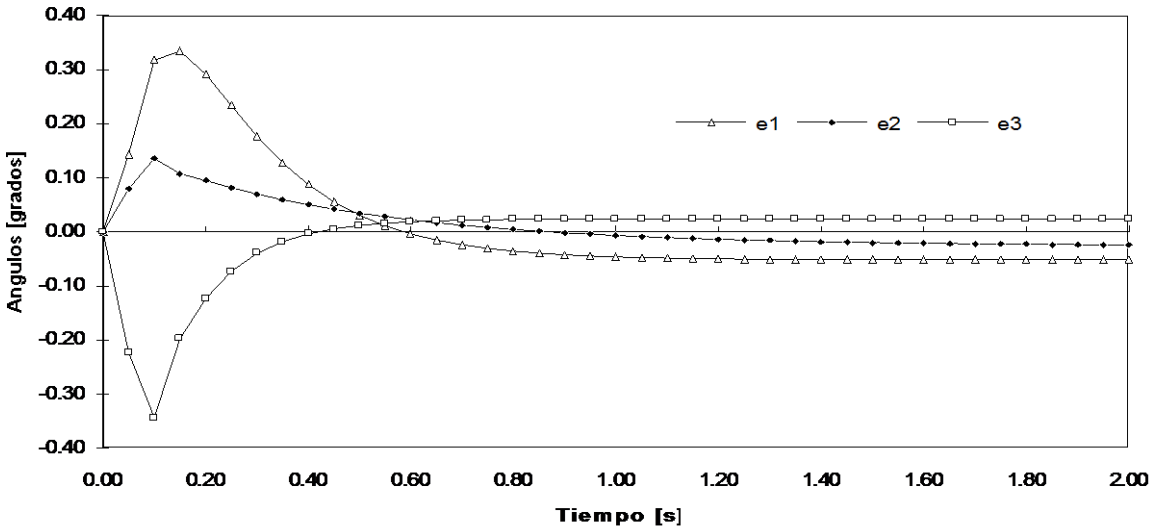


Fig.9a) Errores de posición

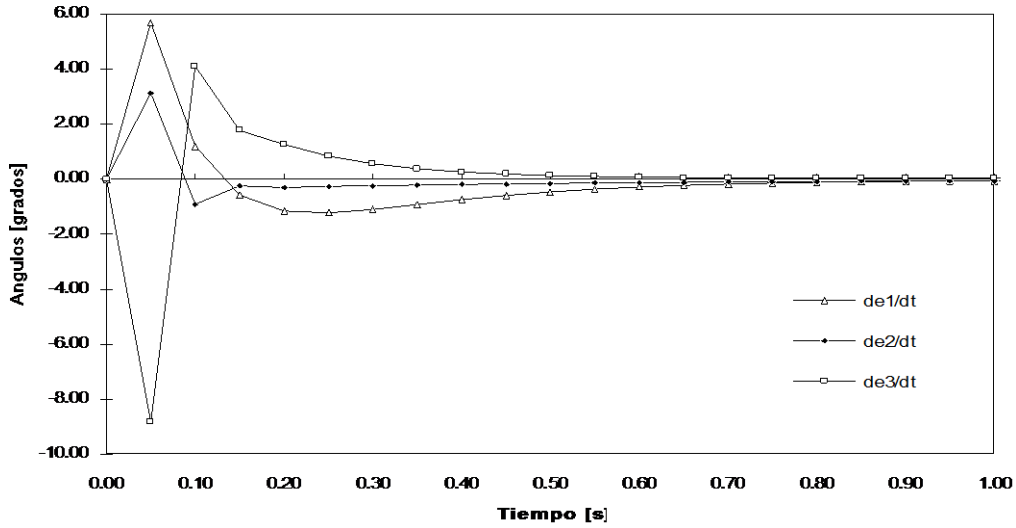


Fig.9 b) Errores de velocidad del ejemplo 1.

Ejemplo 2:

En las figuras 10.a y 10.b se observa el efecto de disminuir el valor de K_V de la junta 1, donde hay un claro desacople entre las juntas, y la dinámica de la junta 1 es subamortiguada.

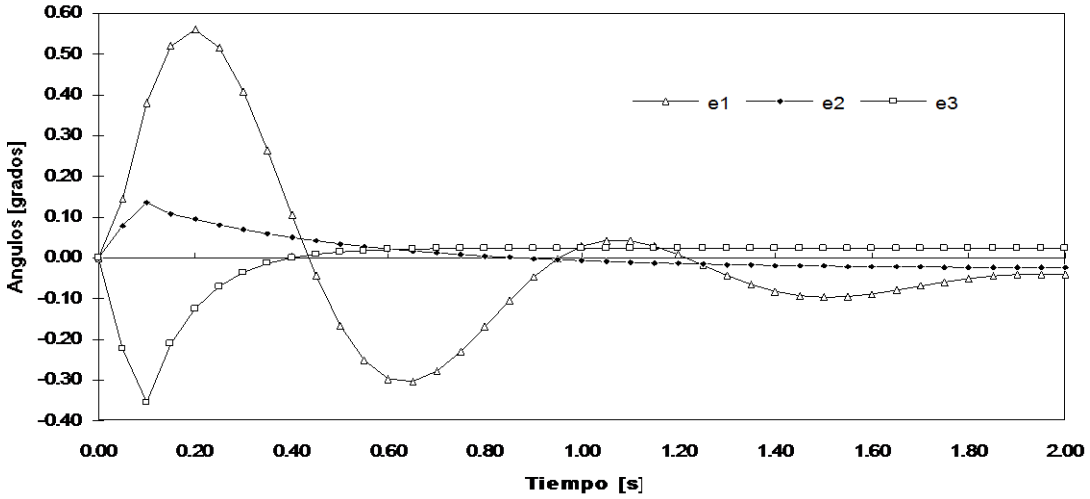


Fig. 10 a) Errores de posición del ejemplo 2.

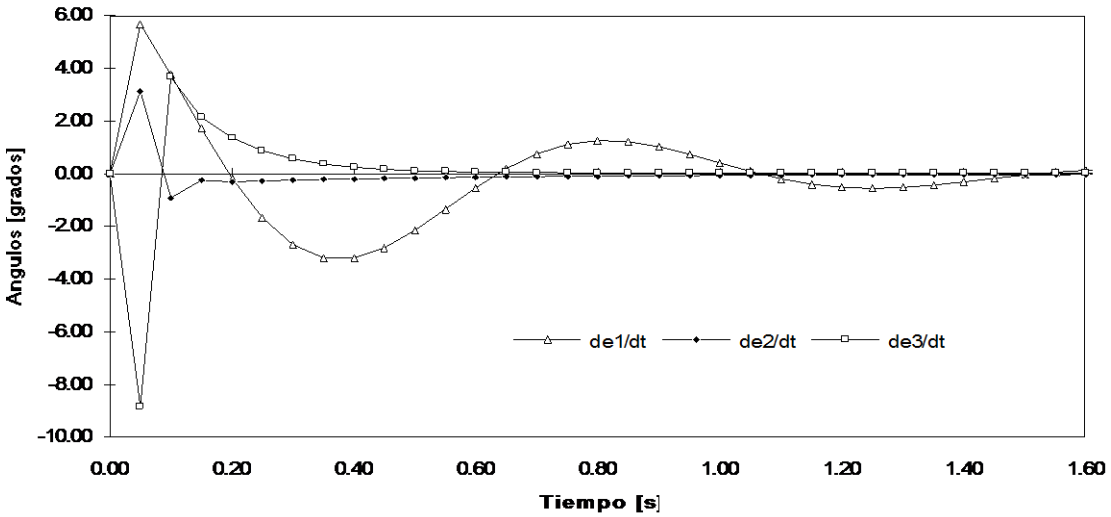


Fig. 10 b) Errores de velocidad del ejemplo 2.

Ejemplo 3: Aquí , se incorpora una perturbación en el torque de la junta 1 (K_{r1}<>0) modelada como ruido blanco. Los demás parámetros son iguales a los de la tabla II.

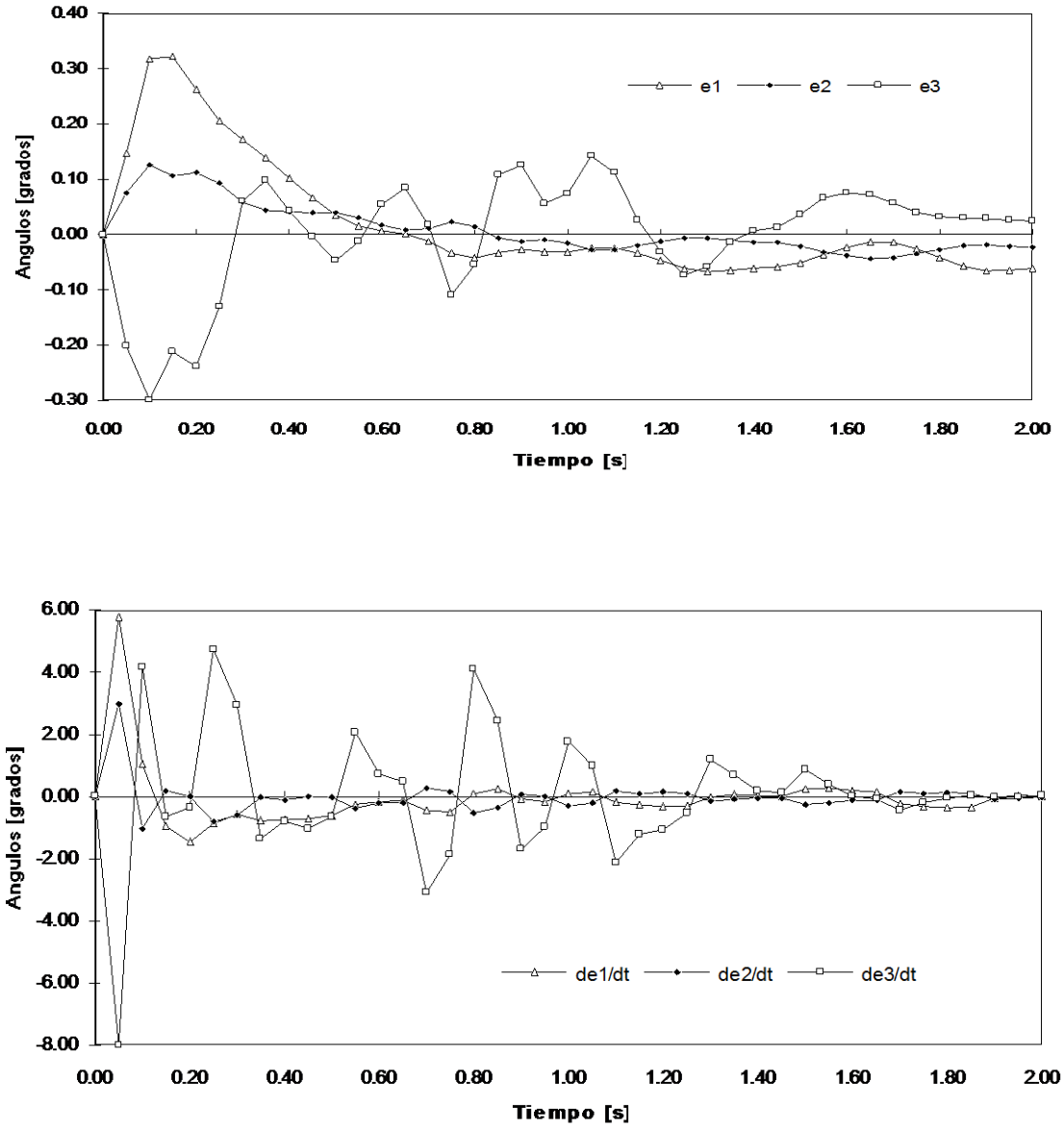


Fig.11: a) Errores de posición y b) Errores de velocidad del ejemplo 3.

B. Control Neuronal

Como primera aproximación, se comenzó realizando todos los programas de manera de entrenar una RN para que se comporte con la dinámica del brazo robot en un rango de trabajo reducido. De esta manera se redujo la envergadura del problema para luego poder pasar a trabajar en el espacio de trabajo completo. Por otra parte, las experiencias obtenidas en el entrenamiento de la red con plantas de dinámica más simple, permitieron sacar rápidamente conclusiones generales debido a que la red aprendía con mayor velocidad.

El criterio con el cual se midió el porcentaje de aciertos fue considerar si la salida de la red neuronal caía dentro de un porcentaje de tolerancia de la salida que debería dar la planta. Dicha tolerancia es en todos los casos del 5%.

La otra medida de performance consideró el error cuadrático medio sobre todas las salidas y los valores correctos. Debe tenerse en cuenta que para la medición de estos indicadores se utilizan los mismos datos que para el entrenamiento, cosa que puede no ser cierta en las simulaciones que siguen.

En las simulaciones, el error cuadrático medio referido al promedio se tomó de manera de relativizar la medida del error de la magnitud de la señal, ya que de otra manera para valores muy pequeños de entrada el error se hace también pequeño, a pesar de que la dinámica de la red no es análoga a la de la planta. Este criterio de error podría también aplicarse al ajuste de los pesos. Los resultados de estos entrenamientos pueden observarse a continuación en los gráficos de salida de la red ya entrenada, comparada con las salidas de la planta. Se presentan además, los resultados numéricos de performance del entrenamiento.

Resultados de la simulación de la red-planta

Durante la etapa de simulación fue probada una serie de estructuras de varios nodos por capa. Dado que la cantidad mínima de pesos necesaria para resolver la dinámica de una planta como ésta era tan sólo de nueve pesos, se optó por una red con nueve entradas, tres neuronas en la primera capa, tres en la segunda y tres en la capa de salida. Las entradas corresponden a los tres torques -uno por cada junta-, y los seis restantes corresponden a los ángulos de las juntas retardados dos veces (z^{-1} y z^{-2}). Las salidas corresponden a los ángulos de las juntas sin retardar. A continuación puede verse un informe de entrenamiento emitido por el sistema desarrollado a tal efecto por el grupo.

Simulación de un brazo robot mediante un PM
Datos de la red y resultados numéricos:

Perceptrón de 3 capas. (algoritmo BP)
9 entradas, 3 neuronas en la 1º capa oculta.
3 neuronas en la 2º capa oculta.
3 neuronas en la salidas que corresponden a $\Theta_1\Theta_2\Theta_3$
Etas=0.05
Alpha=0.01
Datos distintos de entrenamiento: 599 patrones mostrados
Entrenamientos realizados: 55929/1000000
Aciertos/Salida:
Θ_1: 98.00 %
Θ_2: 96.00%
Θ_3: 93.00%
Error Cuadrático Medio/Salida:
Θ_1: 0.023
Θ_2: 0.051
Θ_3: 0.077

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

Esta red demostró alcanzar un mejor comportamiento que otras configuraciones más complejas, con un menor número de entrenamientos y mejor performance y adaptación a los cambios en los niveles de entrada. Durante las pruebas realizadas se notó cierta dificultad para asimilar cambios bruscos en la escala de las entradas presentadas a la red. Esto se manifestó por un total "olvido" de lo aprendido hasta ese momento para tener que "reaprender" la dinámica nuevamente. Para subsanar este problema, en el caso de rangos de trabajo más amplios, se debe trabajar con entradas tales que permitan barrer el espacio de trabajo del robot; cualquier entrada por encima de estos valores provocará seguramente algún efecto de "saturación". En nuestras simulaciones se notó que en general la red aprendía más rápidamente si primero se la entrenaba con ruido y luego con el resto de las entradas en forma alternativa.

En la Fig.12 se observa la respuesta de la red-planta y de la planta para la junta 1 frente a ruido blanco , en la Fig. 13 se presenta la respuesta a un tren de Deltas Dirac para la misma junta y en la Fig. 14 se muestran los resultados para una entrada senoidal.

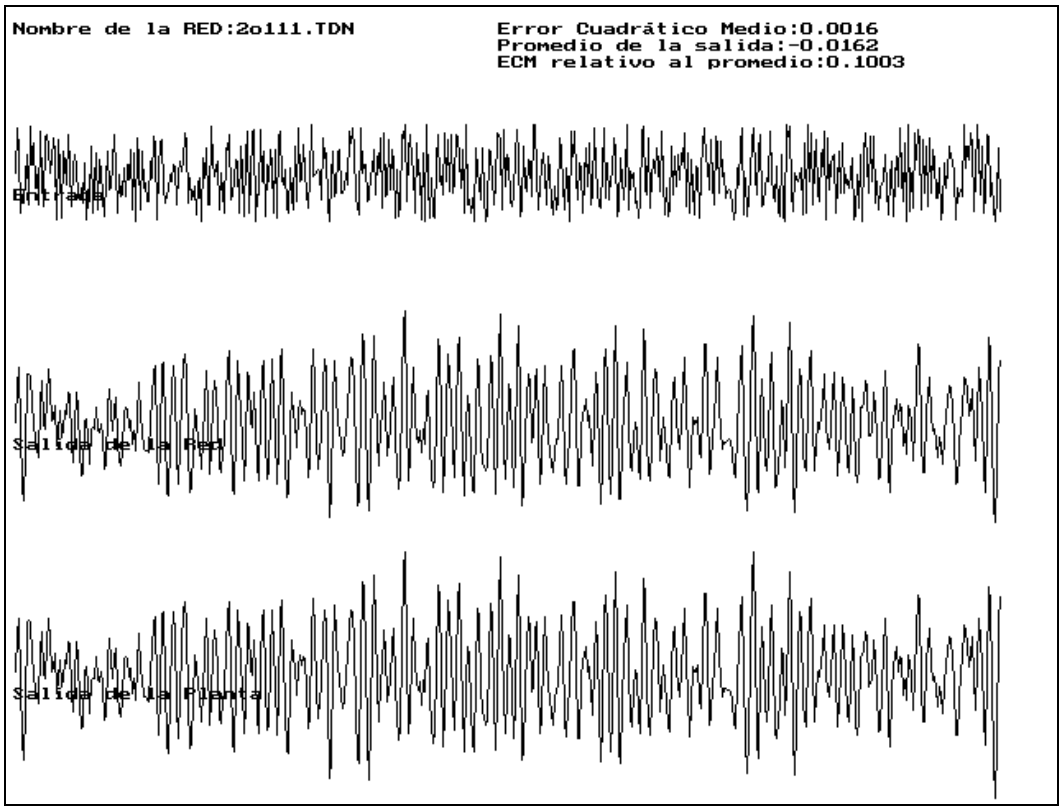


Fig. 12: Respuestas frente a ruido blanco.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

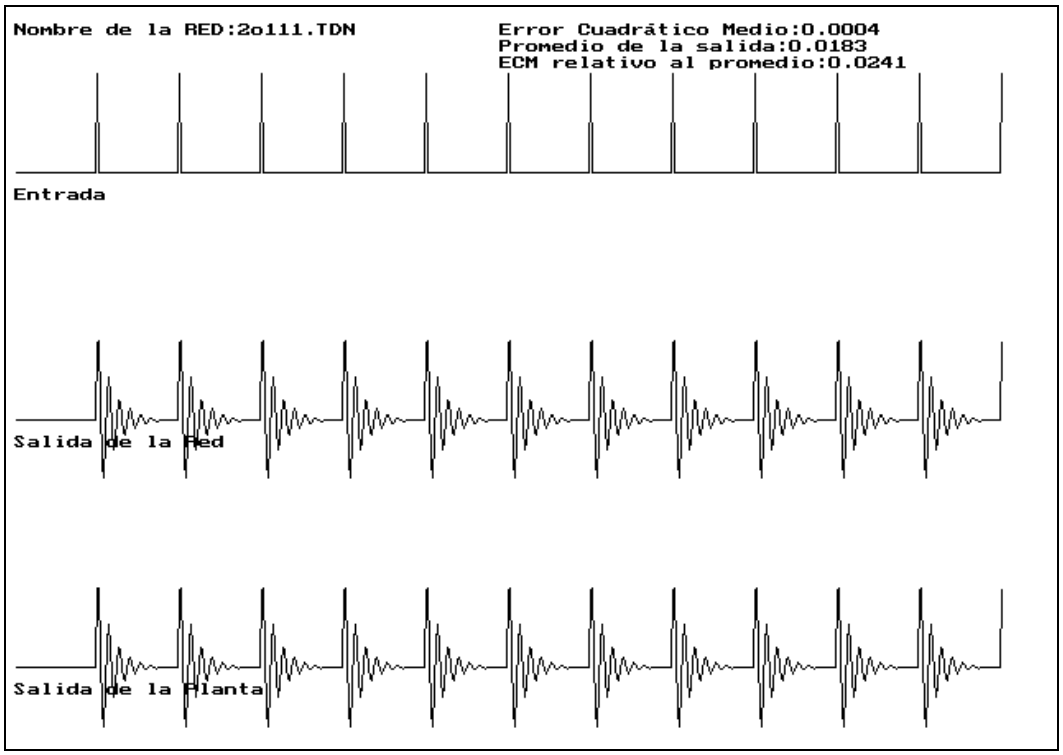


Fig. 13: Respuestas frente a un tren de impulsos Dirac.

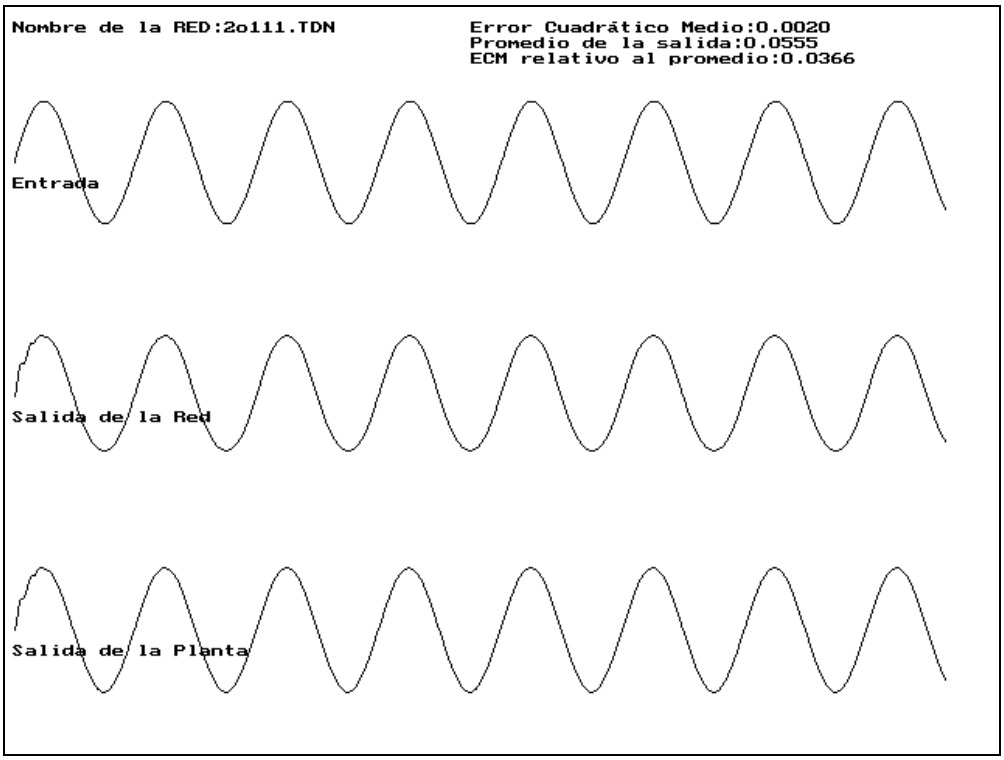


Fig. 14: Respuestas frente a entrada senoidal.

Resultados del entrenamiento del controlador

Para entrenar al controlador se probaron dos alternativas distintas. La primera consistió en entrenar a aquél habiendo entrenado previamente la red-planta, la que congela sus parámetros internos durante el entrenamiento del controlador. Este se

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

entrena a "lazo abierto" según se observa en la figura 15. Este enfoque no dio buenos resultados ya que el controlador tardaba mucho en aprender la dinámica inversa de la planta debido a la información posiblemente errónea que le llegaba de sus salidas en estados (y tiempos) anteriores. En otras palabras, como el controlador al principio tenía una configuración de pesos y umbrales totalmente aleatoria -la cual debía ser optimizada mediante el algoritmo de B.P.- las salidas que proporcionaba eran incorrectas y, por lo tanto, al usarse en el próximo instante como entradas, "confundían " a la red.

La segunda estrategia utilizada consistió en una configuración de "lazo cerrado", donde la red obtenía sus entradas (distintas que la consigna) de las salidas de la planta que simulaba al robot (con sus pesos y umbrales fijos). De esta manera se evitó la inestabilidad del caso anterior. El esquema se observa en la figura 16.

Por último se conectó el controlador entrenado al modelo real del robot según se muestra en la figura 17.

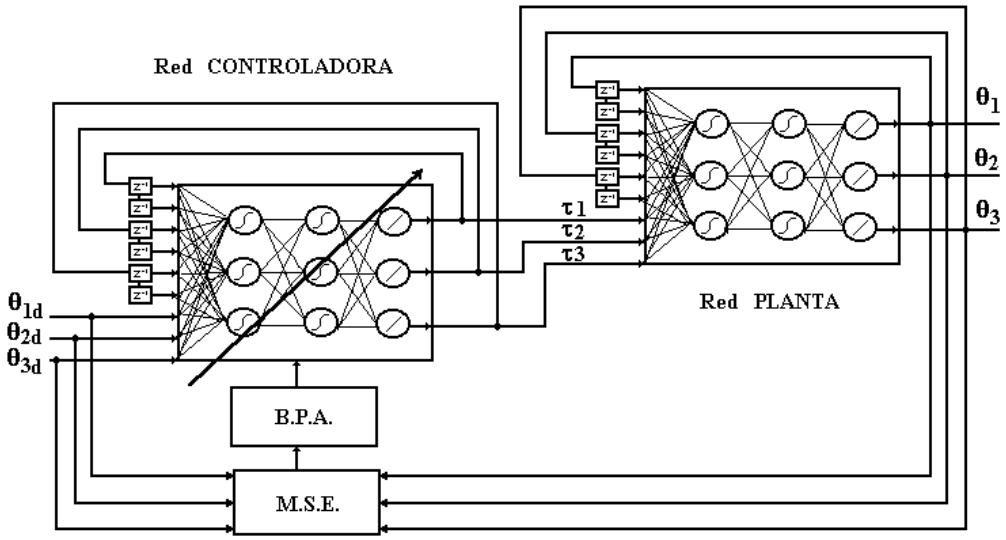


Fig. 15: Entrenamiento del controlador neuronal a lazo abierto.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

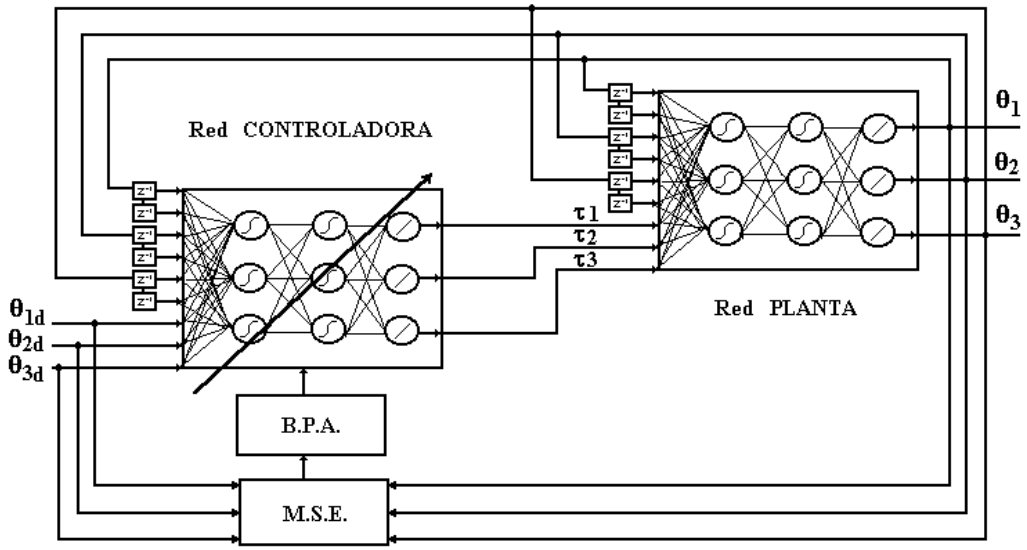


Fig. 16: Entrenamiento del controlador neuronal a lazo cerrado.

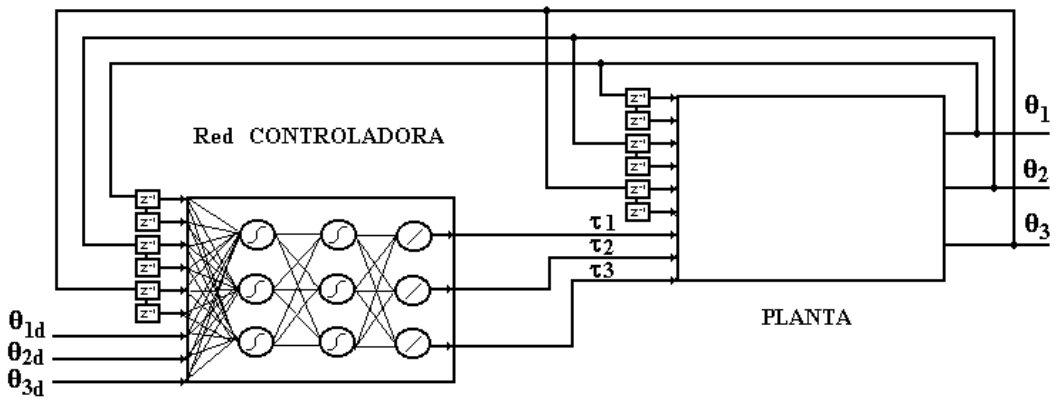
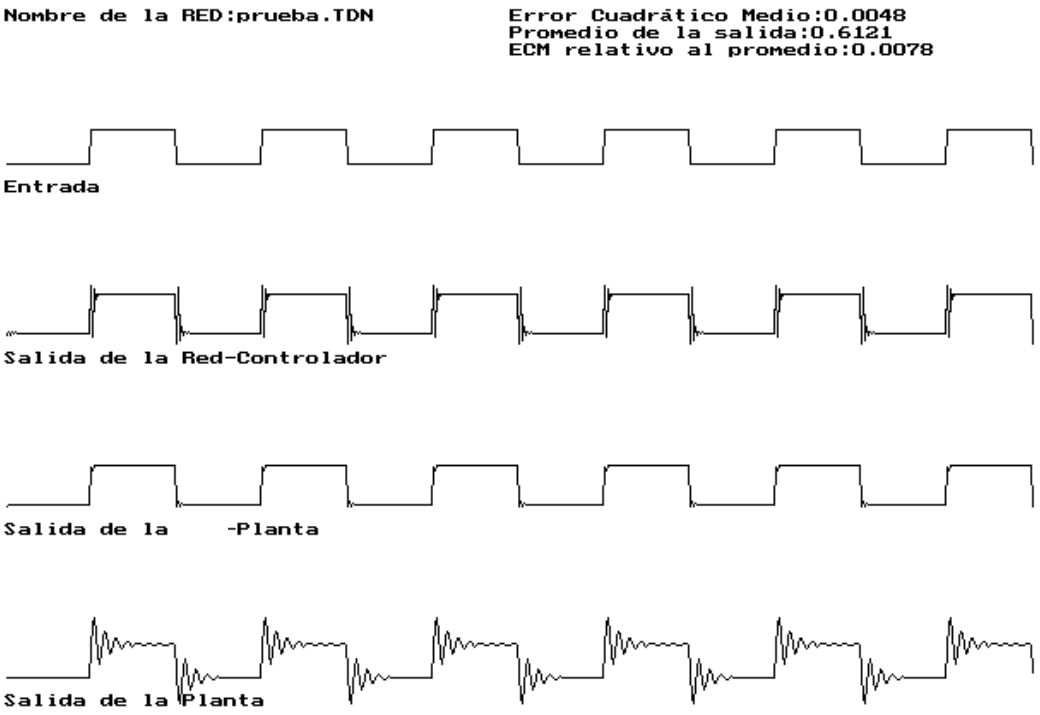


Fig. 17: Esquema del controlador operando sobre la planta real.

A continuación se muestran los comportamientos de la junta 1 , con el controlador y sin él, frente a una consigna del tipo tren de pulsos rectangulares. Como puede apreciarse el controlador actúa eliminando casi por completo los efectos sobre la señal provocados por la planta.



VII. CONCLUSIÓN

En el presente trabajo se pudieron comprobar en forma cualitativa las posibilidades de las redes neuronales como controladores e identificadores de sistemas complejos. A pesar de que el controlador se restringió a un rango de trabajo reducido, los resultados pueden ser extrapolables al espacio de trabajo completo. Se pudo observar que desde el punto de vista de la flexibilidad en el diseño, la adaptabilidad a sistemas dinámicos no lineales y la tolerancia a ruido o cambios del punto de operación de la planta, las redes neuronales representan una mejor alternativa al enfoque clásico. Sin embargo, se requiere un estudio aún más exhaustivo de la performance de estos sistemas en comparación con los convencionales.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

APÉNDICE A

La Regla Delta como un Método de Gradiente Descendente

Para la siguiente deducción considérese un conjunto de pares correctos entrada-salida con los que se entrena una neurona elemental. Para cada pattern que se le presente, la neurona computa un producto interior para estimar el valor de salida. Este valor es restado al valor de salida deseado para generar un error a partir del cual van a ajustarse los pesos de entrada a la neurona para tender a reducirlo. La regla para ajustar los pesos puede escribirse:

$$\Delta w_j = \beta(d(p) - y(p)) I_j(p) = \beta\delta(p) I_j(p) \tag{18}$$

donde *d(p)* es la salida deseada para el p-ésimo pattern, y(p) es la salida estimada para el p-ésimo patrón de entrada, Ij(p) es la j-ésima componente del pattern de entrada p-ésimo, ß es el factor de aprendizaje y Δw_j es el ajuste para el j-ésimo peso.

Demostraremos que la regla Delta minimiza el cuadrado de las diferencias entre la salida deseada y la salida obtenida. A fin de probar que la regla Delta es un método de gradiente descendente, debemos demostrar que la derivada de la función error respecto a cada peso es negativamente proporcional al cambio de peso en la regla Delta.

Sea

$$E(p) = \frac{1}{2}\ d(p) - y(p)\ ^2 \tag{19}$$

la medida del error para el j-ésimo pattern y definamos el error global como E=ΣE(p). Aplicando la regla de la cadena, la derivada de E(p) respecto del j-ésimo peso es

$$\frac{\partial E(p)}{\partial w_j} = \frac{\partial E(p)}{\partial y(p)} \frac{\partial y(p)}{\partial w_j} \tag{20}$$

El primer factor del segundo miembro es el cambio en el error respecto del valor de salida estimado, y el segundo factor es el cambio en la salida estimada respecto del j-ésimo peso.

A partir de la ecuación (19) puede escribirse

$$\frac{\partial E(p)}{\partial y(p)} = -\ d(p) - y(p)\ = -\delta(p) \tag{21}$$

Y dado que

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

$$y(p) = \sum_j w_j I_j(p) \tag{22}$$

el segundo factor de la ecuación (20) es

$$\frac{\partial y(p)}{\partial w_j} = I_j(p) \tag{23}$$

Sustituyendo las ecuaciones (21) y (22) en la ecuación (20) obtenemos

$$-\frac{\partial E(p)}{\partial w_j} = \delta(p) I_j(p) \tag{24}$$

Comparando esta última expresión con la ecuación (18) puede observarse que la regla Delta, como se pretendía demostrar, ajusta los pesos proporcionalmente a la derivada negativa del error cuadrático, respecto de cada peso. Combinando la ecuación (24) con el hecho de que

$$\frac{\partial E}{\partial w_j} = \sum_p \frac{\partial E(p)}{\partial w_j} \tag{25}$$

demuestra que el cambio en w_{ij} luego de un ciclo completo de presentaciones de patterns es proporcional a esta derivada, por lo que la regla Delta constituye un método de gradiente descendente sobre el error global E .

Demostración de la regla delta generalizada

Sea un conjunto de pares correctos entrada-salida con los que se entrena la red. Para cada pattern de entrada presentado se computar un vector estimado de salida en base a la configuración actual de pesos y funciones de activación en las sucesivas capas. Cada una de las neuronas de la red responderá según

$$red_j(p) = \sum_i w_{ij} y_i(p) \tag{26}$$

donde redj(p) es la información que computa la j-ésima neurona de la capa actual para el pattern p-ésimo, wij es el peso de conexión entre la i-ésima neurona de la capa anterior y la j-ésima neurona de la capa actual, y yi(p) es la salida de la i-ésima neurona de la capa anterior para el pattern p-ésimo. Si la neurona i es una unidad de entrada, entonces $y_i(p)=I_i(p)$, donde $I_i(p)$ es el i-ésimo componente del pattern de entrada p-ésimo. La salida de la neurona será

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

$$y_j(p) = f_j(red_j(p)) \tag{27}$$

donde f es una función sigmoide. Según lo visto anteriormente, la regla delta generalizada será un método de gradiente descendente si

$$\Delta w_{ij} \alpha - \frac{\partial E(p)}{\partial w_{ij}} \tag{28}$$

Usando la regla de la cadena, la derivada de E(p) con respecto a los pesos es

$$\frac{\partial E(p)}{\partial w_{ij}} = \frac{\partial E(p)}{\partial red_j(p)} \frac{\partial red_j(p)}{\partial w_{ij}} \tag{29}$$

donde el último término puede escribirse

$$\frac{\partial red(p)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} y_k(p) = y_i(p) \tag{30}$$

si definimos $\delta_j(p) = -\frac{\partial E(p)}{\partial red_j(p)}$, la ecuación (27) puede ser reescrita como

$$-\frac{\partial E(p)}{\partial w_{ij}} = \delta_j(p) y_j(p) \tag{31}$$

la cual corresponde a la expresión simbólica de la regla delta. Sin embargo, tal expresión no nos proporciona información sobre cómo calcular $\delta_j(p)$. Para calcularla aplicamos la regla de la cadena para obtener

$$\delta_j(p) = -\frac{\partial E(p)}{\partial red_j(p)} = -\frac{\partial E(p)}{\partial y_j(p)} \frac{\partial y_j(p)}{\partial red_j(p)} \tag{32}$$

Utilizando la ecuación (27), la segunda parte de la ecuación (32) es

$$\frac{\partial y_j(p)}{\partial red_j} = f'(red_j(p)) \tag{33}$$

Considérense dos casos para el primer término. Primero que se trata de una neurona de salida, por lo tanto

$$\frac{\partial E(p)}{\partial y_j(p)} = -\, d(p) - y_j(p) \tag{34}$$

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

sustituyendo (33) y (34) en la ecuación (32) obtenemos

$$\delta_j(p) = (d_j(p) - y_j(p))\, f'(red_j(p)) \tag{35}$$

para cualquier neurona de salida. Segundo, considérese que no se trata de una neurona de salida, entonces utilizando la regla de la cadena

$$\begin{aligned} \sum_k \frac{\partial E(p)}{\partial red_k(p)} \frac{\partial red_k(p)}{\partial y_j(p)} &= \sum_k \frac{\partial E(p)}{\partial red_k(p)} \frac{\partial}{\partial y_j(p)} \sum_i w_{ij} y_i(p) \\ &= \sum_k \frac{\partial E(p)}{\partial red_j(p)} w_{ij} \\ &= \sum_k \delta_k(p) w_{jk} \end{aligned} \tag{36}$$

donde wjk es el peso de conexión entre la j-ésima neurona de la capa actual con la k-ésima neurona de la capa siguiente y la sumatoria se toma sobre todas las neuronas de la capa siguiente. Sustituyendo (33) y (36) en la ecuación (32) obtenemos

$$\delta_j(p) = f'(red_j(p)) \sum_k \delta_k(p) w_{jk} \tag{37}$$

para cualquier neurona que no pertenezca a la capa de salida. Si las ecuaciones anteriores son aplicadas para $\delta_j(p)$, la regla Delta generalizada se aproxima a un método de gradiente descendente.

APÉNDICE B

Modelo Cinemático

Para la obtención del modelo cinemática utilizamos la técnica de representación homogénea de Denavit-Hartenberg, asignando los sistemas de referencia como se observan en la Figura 7, a partir de los cuales se obtienen los parámetros de junta que se muestran en la Tabla I.

La matriz de transformación del órgano terminal respecto a la base esta dada por la expresión

$$ {}^{0}T_{4} = {}^{0}T_{1}\,{}^{1}T_{2}\,{}^{2}T_{3}\,{}^{3}T_{4} \tag{38} $$

$$ {}^{0}T_{4} = \begin{bmatrix} c_{123} & -s_{123} & 0 & L + L_1c_1 + L_2c_{12} + L_3c_{123} \\ s_{123} & c_{123} & 0 & L + L_1s_1 + L_2s_{12} + L_3s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Donde

$$ c_{ij} = \cos(\theta_i + \theta_j) $$
$$ s_{ij} = sin(\theta_i + \theta_j) $$

La matriz de la ecuación 30, permite conocer la posición del extremo del órgano terminal, respecto de la base para cualquier configuración, por medio del vector traslación

$$ {}^{0}P_{OT} = \begin{bmatrix} P_{x0} & P_{y0} & P_{z0} \end{bmatrix}^{t} \tag{39} $$

Donde las componentes del vector son

$$ P_{x0} = L + L_1c_1 + L_2c_{12} + L_3c_{123} $$
$$ P_{y0} = L + L_1s_1 + L_2s_{12} + L_3s_{123} $$
$$ P_{z0} = 0 $$

Modelo Cinemático Inverso

A partir de la figura 5 y de las ecuaciones 38 y 39, se definen las siguientes variables

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

$$
\begin{aligned}
x &= L + L_1 s_1 + L_2 s_{12} + L_3 s_{123} \\
y &= L_1 c_1 + L_2 c_{12} + L_3 c_{123} \\
\phi &= \theta_1 + \theta_2 + \theta_3
\end{aligned}
\tag{40}
$$

Trabajando algebraicamente con las ecuaciones anteriores, se obtiene finalmente

(41)

$$
\theta_2 = \cos^{-1}\left[\frac{(y - L_3 sin(\phi))^2 + (x - L - L_3\cos(\phi))^2 - {L_1}^2 - {L_2}^2}{2 L_1 L_2}\right]
$$

$$
\theta_1 = \tan^{-1}\left[\frac{(y - L_3 sin(\phi))}{(x - L - L_3\cos(\phi))}\right] - \tan^{-1}\left[\frac{L_2 sin(\theta_2)}{L_1 + L_2\cos(\theta_2)}\right]
$$

$$
\theta_3 = \phi - \theta_1 - \theta_2
$$

APÉNDICE C

Términos del modelo dinámico

1. Matriz de Inercia

(42)

$$M_{11} = m_1 L_1^{\,2} + m_2(L_1^{\,2} + L_2^{\,2} + 2L_1L_2c_2) + m_3(L_1^{\,2} + L_2^{\,2} + L_3^{\,2} + 2L_1L_2c_2 + 2L_1L_3c_{23} + 2L_2L_3c_3)$$
$$M_{12} = m_2(L_2^{\,2} + L_1L_2c_2) + m_3(L_2^{\,2} + L_3^{\,2} + L_1L_2c_2 + L_1L_3c_{23} + 2L_2L_3c_3)$$
$$M_{13} = m_3(L_3^{\,2} + L_1L_3c_{23} + L_2L_3c_3)$$
$$M_{21} = M_{12}$$
$$M_{22} = m_2L_2^{\,2} + m_3(L_3^{\,2} + L_2^{\,2} + 2L_2L_3c_2)$$
$$M_{23} = m_2L_2^{\,2} + m_3(L_3^{\,2} + L_2^{\,2} + 2L_2L_3c_2)$$
$$M_{31} = M_{13}$$
$$M_{32} = M_{23}$$
$$M_{33} = m_3L_3^{\,2}$$

2.Matriz de Coriolis

(43)

$$B_{11} = -2m_2L_1L_2s_2 - 2m_3(L_1L_2s_2 + L_1L_3s_{23})$$
$$B_{12} = -2m_3(L_1L_3s_{23} + L_2L_3s_3)$$
$$B_{13} = B_{12}$$
$$B_{21} = 0$$
$$B_{22} = -2m_3L_2L_3s_3)$$
$$B_{23} = B_{22}$$
$$B_{31} = -B_{23}$$
$$B_{32} = 0$$
$$B_{33} = 0$$

3. Matriz de términos de fuerza centrifuga

(44)

$$C_{11} = 0$$
$$C_{12} = -m_2L_1L_2s_2 - m_3(L_1L_3s_{23} + L_2L_1s_2)$$
$$C_{13} = -m_3(L_1L_3s_{23} + L_2L_3s_3)$$
$$C_{21} = m_2L_1L_2s_2 + m_3(L_1L_2s_2 + L_1L_3s_{23})$$
$$C_{22} = 0$$
$$C_{23} = -m_3L_3L_2s_3$$
$$C_{31} = m_3(L_1L_3s_{23} + L_2L_3s_3)$$
$$C_{32} = m_3L_3L_2s_3$$
$$C_{33} = 0$$

4. Vector gravedad

(45)

$$g_1 = -(m_1 + m_2 + m_3) L_1 s_1 - (m_2 + m_3) L_2 s_{12} - m_3 L_3 s_{123}$$
$$g_2 = -(m_2 + m_3) L_2 s_{12} - m_3 L_3 s_{123}$$
$$g_3 = -m_3 L_3 s_{123}$$

5. Vector fricción

Se modela la fricción en cada una de las juntas considerando fricción viscosa y de Coulomb. El vector **F** está dado por

$$f_1 = k_{h1}\,\dot{\theta}_1 + k_{c1} signo(\dot{\theta}_1)$$
$$f_2 = k_{h2}\,\dot{\theta}_2 + k_{c2} signo(\dot{\theta}_2) \qquad (46)$$
$$f_3 = k_{h3}\,\dot{\theta}_3 + k_{c3} signo(\dot{\theta}_3)$$

donde los vectores K_h y K_c son los coeficientes de las fricciones viscosa y de Coulomb respectivamente.

REFERENCIAS

-[Gal90] S. Gallant. Perceptron-Based Learning Algorithms. Trans. on Neural Networks. Vol. 1, No. 2, June 1990.

-[HuH90] S. Huang, Y. Huang. Learning Algorithms for Perceptrons Using Back-Propagation with Selective Updates. IEEE Control Systems Mag. Apr. 1990.

-[HuH91] S. Huang, Y. Huang. Bounds on the Number of Hidden Neurons in Multilayer Perceptrons. Trans. on Neural Networks. Vol. 2, No. 1, Jan. 1991.

-[Lip87] R. Lippmann, An Introduction to Computing with Neural Nets. IEEE ASSP Mag., Vol 4. (1987).

-[Mel89] P. Melsa, Neural Networks: A conceptual Overview, Octubre 1989.

-[Nar90] K. S. Narendra, Identification and Control of Dinamical System using Neural Networks, IEEE Trans. Neural Networks, Vol. 1 Nº1, Marzo 1990.

-[Psa88] D. Psaltis, A Multilayer Neural Network Controller, IEEE Control System Magazine, Abril 1988.

-[Shi90] J. Shink, Performance Surfaces of a Single-Layer Perceptron. Trans. on Neural Networks. Vol. 1, No. 3, Sep. 1990.

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
H. L. Rufiner, D. Zapata, A. Sigura & L. G. Gamero; "Control de un Robot Mediante Computación Neuronal"
Anales del X Congreso Nacional de Informática y Telecomunicaciones, Vol. 1, pp. 201-208, may, 1993.

Sumario