Patrones de Sistemas de Primero y Segundo Orden, en un Ambiente de Instrumentación Virtual

Luis Pastor Sánchez Fernández
Centro de Investigación en Computación del IPN
Av. Juan de Dios Bátiz esquina con M. Othón de Mendizábal
Unidad Profesional Adolfo López Mateos. México, D.F. 07738, México
lsanchez@cic.ipn.mx

RESUMEN

Se establecen los patrones para reconocer, mediante una red neuronal (RN) backpropagation, modelos dinámicos de sistemas de primero y segundo orden. La RN de tres capas, tiene 30 neuronas de entrada, 11 en la capa oculta y 4 neuronas de salida. Se utiliza un almacenador circular para guardar los n últimos valores adquiridos de cada variable.
Antes de ejecutar la RN, los datos almacenados son acondicionados y filtrados digitalmente. Posteriormente se realiza una conversión de la frecuencia de muestreo para obtener 30 puntos. La salida de la RN indicará cual es el modelo más apropiado. El software es desarrollado utilizando el LabVIEW [1] y DLL el DELPHI y C.
Palabras clave: virtual, neuronales, patrones, predicción, supervisión.

I. INTRODUCCIÓN

Las RN son herramientas matemáticas muy poderosas en la identificación de sistemas. La red backpropagation es utilizada de forma muy exitosa para reconocer patrones diversos y en el trabajo que se presenta se utiliza para reconocer patrones de señales de sistemas dinámicos de 1er. orden y de 2do. orden [1], con los cuales se puede representar, con buena aproximación, la dinámica de una considerable cantidad de procesos tecnológicos. La metodología utilizada consiste en estimar los parámetros de los modelos mediante un algoritmo de optimización [3][4][5]. Antes de realizar la estimación de dichos parámetros se determina, mediante una red neuronal backpropagation previamente entrenada, cual será el modelo más apropiado, lo cual disminuye el tiempo total de procesamiento.

II. DIAGRAMA DE BLOQUES

En la fig. 1 se presenta el diagrama de flujo de la primera sección del ciclo de un algoritmo de alarma predictiva que utiliza una RN para determinar el modelo más apropiado. Se utiliza un almacenador circular de dimensión configurable. Este ciclo inicia con el almacenamiento permanente de los N últimos datos de la variable del proceso tecnológico o dispositivo que es supervisado. Mediante un algoritmo de por tendencia lineal se determina el instante en el cual se debe iniciar el proceso de reconocimiento del patrón de la señal, correspondiente a los puntos guardados en el almacenador circular.

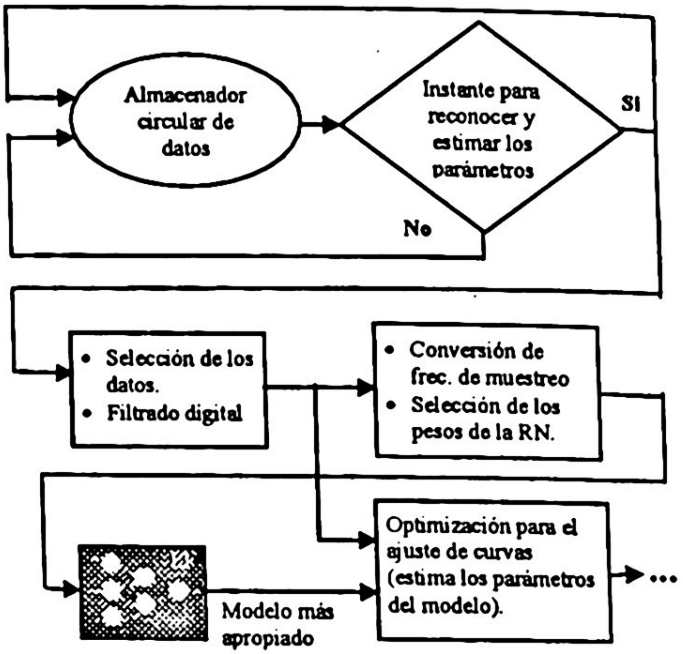


Fig. 1. Diagrama de flujo de la primera sección del ciclo de un algoritmo de alarma predictiva.

Si es el instante de reconocer el modelo, se seleccionan los datos como se ilustra de forma simplificada en la fig. 2 desechándose los datos más viejos hasta el punto donde se produce un cambio en la tendencia de la pendiente, como se señala con los puntos 1, 2, 3 y 4.
Posteriormente se realiza un filtrado de media móvil con un filtro que tiene la siguiente expresión [6]:

$$Y(k) = \frac{1}{2M+1} \sum_{i=-M}^{+M} X(k-i) \qquad (1)$$

Se está utilizando M = 2.
Seguidamente se realiza una conversión de la frecuencia de muestreo, mediante un factor no entero

J. Díaz de León, G. González, J. Figueroa (Eds.): Avances en Ciencias de la Computación, pp. 286-290, 2003.

(ejemplo en la fig. 3) combinando la interpolación y el diezmado [7] lo cual permite obtener 30 puntos que es el número de neuronas de entrada de la RN. Se selecciona el conjunto de los pesos de la RN, de acuerdo al signo de la pendiente de la curva formada por los 30 puntos pues se entrenó la misma para patrones con pendiente positiva y negativa. La RN dará como salida el modelo más apropiado para el cual se ajustarán el total de los puntos seleccionados.

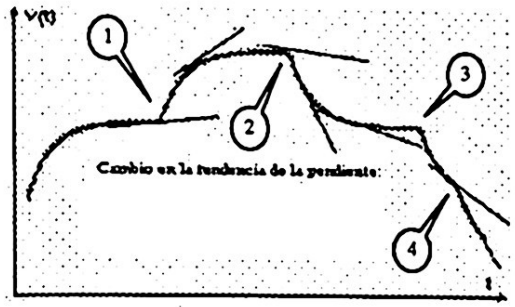


Fig. 2. Ilustración del procedimiento para tomar los puntos que se utilizarán en la identificación.

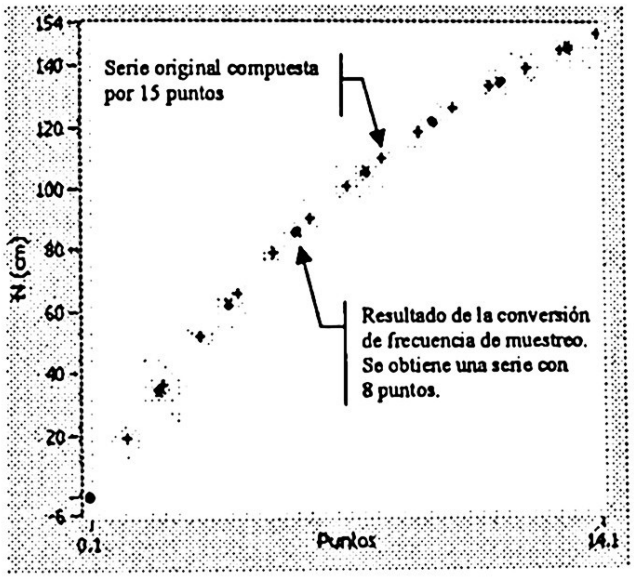


Fig. 3. Ejemplo de conversión de la frecuencia de muestreo, utilizando un factor no entero.

2.1 Conversión de la frecuencia de muestreo.

Para obtener 30 puntos correspondientes a las 30 neuronas de entrada, se realiza la conversión de la frecuencia de muestreo. Se obtiene un factor de conversión como sigue:

$$\text{FactorCFM} = \frac{\text{NPSF}}{\text{NPSO}}$$

Donde:
NPSF: número de puntos de la serie final (30 puntos).
NPSO: número de puntos de la serie original obtenidos del almacenador circular.
Por ejemplo:

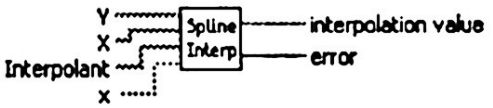
$$\text{NPSF=30 y NPSO=45}$$

$$\text{FactorCFM} = \frac{2}{3}$$

Que significa que el factor de interpolación es 2, lo que equivale a insertar una muestra entre cada dos puntos. El factor de diezmado es 3, lo que significa tomar un punto y eliminar 2, hasta el final de la serie. Se realiza un ajuste previo de la cantidad de puntos de la serie original (NPSO) para poder obtener un FactorCFM adecuado, lo cual se logra eliminando puntos finales, hasta que NPSO sea múltiplo de algunos de los siguientes números: 5, 6, 10, 15 ó 30.

2.1.1 Interpolación mediante "splines" cúbicos.

Para realizar esta interpolación se usó la biblioteca de funciones matemáticas que tiene el LabVIEW [1], las cuales también están disponibles en el MATLAB. De acuerdo a la notación que aparece en el LabVIEW, se usa el siguiente esquema:



Spline Interpolation.vi

Donde:
Y: arreglo los valores que serán interpolados.
X: arreglo de los valores correspondiente al eje x, en este caso, será tiempo, el cual se irá incrementando en el período de muestreo con que fueron adquiridos los datos del arreglo Y.
x: (x minúscula) valor del eje x para el cual se desea tener un punto interpolado. Este valor se irá incrementando en el nuevo período de muestreo que resultará de la interpolación, será más pequeño que el período original de muestreo y estará dentro del rango de los valores del arreglo X.
Interpolant: es la segunda derivada de la función de interpolación de "spline" cúbica, la cual es calculada por otro instrumento virtual (VI), denominado ***spline interpolant.vi.***
error: retorna un valor que indica si la ejecución de la función fue exitosa o no.

El valor de salida de la interpolación z (interpolation value), en el intervalo $[x_i, x_i + 1]$ está dado por:

$$z = Ay_i + By_i + 1 + Cy_i^{\cdot} + Dy_i^{\cdot} + 1$$

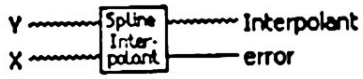
Donde:

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i},\ B = 1 - A$$

$$C = \frac{1}{6}(A^3 - A)(x_{i+1} - x_i)^2$$

$$D = \frac{1}{6}(B^3 - B)(x_{i+1} - x_i)^2$$

El valor interpolant, puede ser obtenido con otra función (instrumento virtual: Spline Interpolant.vi) que tienen como entradas los registros X, Y.



Spline Interpolant.vi

La función de interpolación g(x) pasa a través de todos los puntos: (x_i, y_i).

$y_i = g(x_i)$; donde i = 0,1,.....n-1.

El instrumento virtual: ***spline Interpolant.vi***, obtiene la función de interpolación g(x) interpolando en cada intervalo $[X_i, X_{i+1}]$ con una función polinómica cúbica $P_i(x)$ que reúne las condiciones siguientes:

1. $p_i(x_i) = y_i$

2. $p_i(x_{i+1}) = y_{i+1}$

3. g(x) tiene continua la primera y segunda derivada en el intervalo $[X_0, X_{n-1}]$ y se cumple que:

   a) $p_i'(x_i) = p_{i+1}'(x_i)$

   b) $p_i''(x_i) = p_{i+1}''(x_i)$

Para i = 0, 1,...,n-2.

De la última condición, se derivan las ecuaciones siguientes:

$$\frac{x_i - x_{i-1}}{6} g''(x_{i-1}) + \frac{x_{i+1} - x_{i-1}}{3} g''(x_i) + \frac{x_{i+1} - x_i}{6} g''(x_{i+1}) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

Donde i = 1, 2,...,n-2 y se obtienen n-2 ecuaciones con n $g''(x_i)$ desconocidas, para i = 0, 1,...,n-1. Este instrumento virtual (***spline Interpolant.vi***) calcula $g''(x_0)$, $g''(x_{n-1})$ usando la fórmula siguiente:

$$g'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{3A^2 - 1}{6}(x_{i+1} - x_i) g''(x_i) + \frac{3B^2 - 1}{6}(x_{i+1} - x_i) g''(x_{i+1})$$

Donde:

$$A = \frac{x_{i+1} - x}{x_{i+1} - x_i}$$

$$B = 1 - A = \frac{x - x_i}{x_{i+1} - x_i}$$

Este VI usa $g''(x_0)$, $g''(x_{n-1})$ para resolver todas las $g''(x_i)$, para i = 1,..., n-2; $g'(x_i)$ es la salida Interpolant, la cual se utiliza como entrada en el VI ***spline interpolation.vi.***

2.1.2 Diezmado.

Se programó un sencillo procedimiento para realizar el diezmado, cuya parte principal se presenta a continuación:

En el arreglo **Puntos** (variable declarada globalmente) se encuentra la serie con los valores ya interpolados.

```
Procedure Diezma;
VAR
   I : Integer;
   Indice: Integer;
BEGIN
   Indice:=1+FactorDiezmado;
   For I:=2 TO NPSF DO BEGIN
      Puntos[I]:=Puntos[Indice];
      Indice:=Indice+FactorDiezmado;
    END;
END;
```

III. PATRONES DE ENTRENAMIENTO

Se programó una RN backpropagation en una DLL que es utilizada desde el LabVIEW [1]. La selección de los patrones de entrenamiento, se basó en el comportamiento de las respuestas dinámicas de los sistemas de primero y segundo orden a estímulos paso escalón por ser los más frecuentes. En la fig. 4 se muestran las respuestas de un sistema de 2do. orden críticamente amortiguado, con frecuencias naturales de oscilación w_n igual a 1, 0.5 y 0.25, respectivamente. Para cada curva se muestran 30 puntos, los cuales han sido tomados a frecuencias de muestreo aproximadas de 4, 2 y 1 muestras por segundo, respectivamente. Es por ello, que cada intervalo de tiempo en el eje X, será el período de muestreo de cada curva. Si los puntos de las tres curvas son graficados usando un mismo intervalo de tiempo para el eje X, quedan superpuestas como se ilustra en la fig. 5.

Similar comportamiento se presentará en sistemas de primer orden respecto a la constante de tiempo, y en sistemas de segundo sobreamortiguados y subamortiguados [1], en los cuales sólo habrá diferencia respecto a su coeficiente de amortiguamiento ζ, como se ilustra en la fig. 6.

Se observa que la w_n no tiene influencia en el patrón de la señal cuando cada marca en el eje X es el orden en que se tomaron los puntos, en lugar del tiempo. No es de interés estimar los parámetros del modelo con la RN (constante de tiempo y ganancia para 1er. orden; ganancia, ζ y w_n para sistemas de 2do. orden) y sí reconocer cual es el tipo de modelo que mejor ajusta o representa al registro de datos almacenados. Por tanto para los patrones de entrada de sistemas de 1er. orden y 2do. críticamente amortiguado sólo se varía la amplitud, para 2do. orden sobreamortiguado y subamortiguado sólo se varían la amplitud y el ζ. Las variaciones en amplitud se toman en %, normalizadas, desde el 40% al 90%, como se ilustra en la fig. 7 para un sistema de segundo orden críticamente amortiguado. De forma similar se hizo para los restantes sistemas.

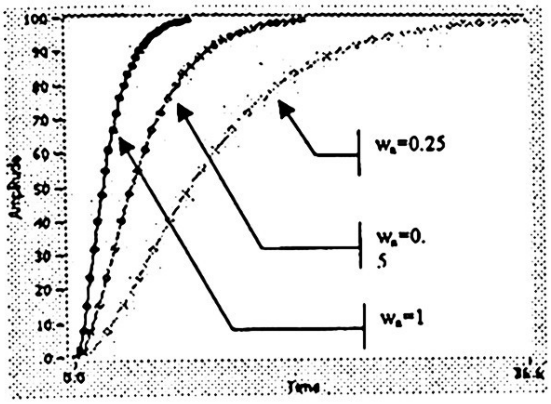


Fig. 4. Sistema críticamente amortiguado.

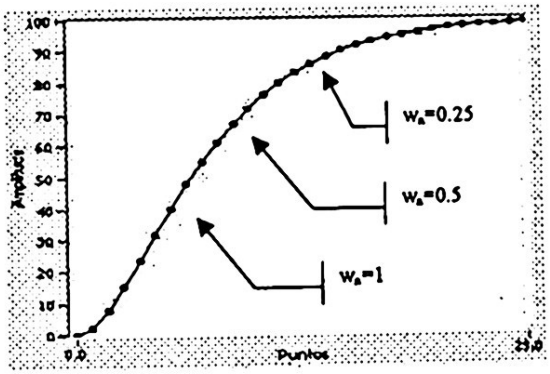


Fig. 5 Las tres curvas de la fig. 4, superpuestas.

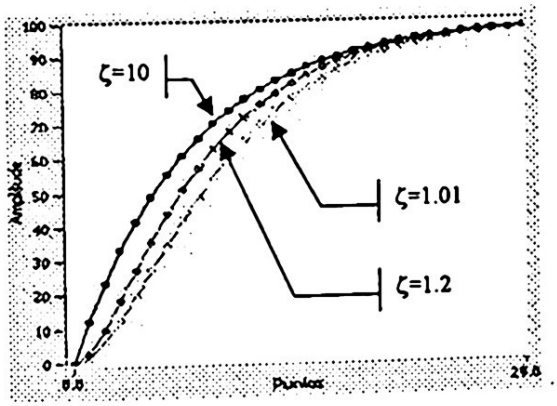


Fig. 6 Respuestas sobreamortiguadas.

Cuando el programa está en operación y corresponde procesar la señal, si el valor máximo de la misma está entre 40 y 90 se deja con su valor; si es inferior a 40, se amplifica para llevarla hasta 40; Si es superior a 90, se atenúa para reducirla a 90. Se observaron mejores resultados cuando se amplifica o atenúa, según el caso, hasta llevarla a 90, siempre que la misma esté fuera del intervalo de 40-90.

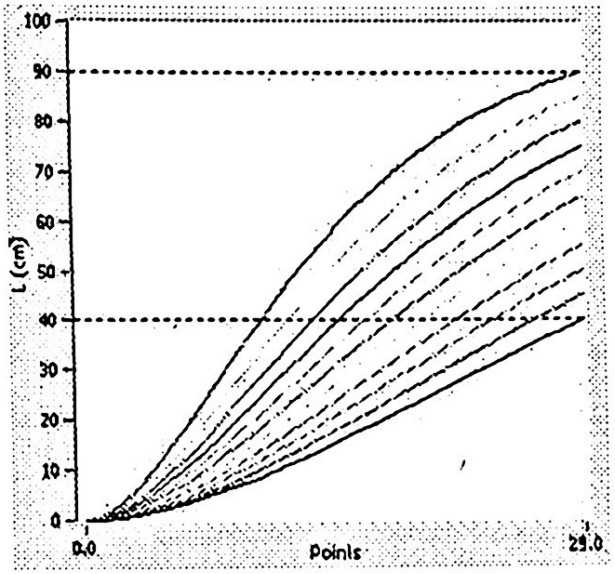


Fig. 7 Patrones con pendientes positivas, para sistemas críticamente amortiguados.

Después de numerosas pruebas, se realizó el entrenamiento con 858 patrones de entrada, distribuidos de la manera siguiente:

- Para sistemas de 2do. orden sobreamortiguado (SSobre):
  Por cada valor de ζ se obtienen 11 patrones correspondientes a las variaciones de la amplitud desde 40 hasta 90, con incremento de 5 (40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90).
  El ζ se varía desde 1.2 hasta 3, con incremento de 0.09, obteniéndose un total de 220 patrones.
  Para ζ mayores que 3, el sistema se confunde con uno de 1er. orden.
- Para sistemas de 2do. orden subamortiguado (SSub):
  De forma similar por cada valor de ζ se obtienen 11 patrones correspondientes a las variaciones de la amplitud.
  El ζ se varía desde 0.1 hasta 0.7, con incremento de 0.0667, obteniéndose un total de 99 patrones.
- Para sistemas de 1er. orden (P) y 2do. orden críticamente amortiguado (SC) se crean 11 patrones, respectivamente, correspondientes a variaciones de la amplitud desde 40 hasta 90.

Para tener un número similar de patrones para cada modelo y lograr un mejor entrenamiento de la RN, los patrones de (P) y de (SC) se repiten 20 veces, respectivamente, para un total de 440 patrones.

Para (SSub.) se repiten dos veces para 198 patrones. EN TOTAL 858 PATRONES. Se obtienen muy buenos resultados en el entrenamiento y en la generalización de la RN. El error de entrenamiento fue del 0.15%.
Una vez seleccionados los patrones, se fueron utilizando diversas topologías hasta obtener la más simple pero que diera una respuesta adecuada. Finalmente se utilizó una red neuronal con 30 entradas, 11 neuronas en la capa oculta y cuatro neuronas de salida.
Se utilizaron más de 1000 patrones de pruebas obteniéndose una respuesta correcta de la red neuronal, con un error del 2% de fallos.

IV. PROGRAMACIÓN EN LABVIEW

Cómo entorno principal de desarrollo se ha utilizado el LabVIEW 6.1 [1], mediante el cual se programan aplicaciones conocidas como instrumentos virtuales. La RN backpropagation se programó usando el Delphi [1], exportada mediante una DLL y utilizada por los instrumentos virtuales desarrollados en LabVIEW.

4.1 Código en pascal de la RN backpropagation.

```
Procedure Backpropagation;
VAR
    k : Integer; {general counter}
    I : Integer; {counter of layer}
    J : Integer; {counter of neurons}
    I : Integer; {counter of neurons of preview layer}
{commentaries}
{out : array of neurons's exits}
{InputVector: input values array of neuronal network }
{Layers: number of exit layer of neuronal network.
For 3 layers, layers=2, because the input layer is
number zero}
{N : array for the number of neurons of each layer}

BEGIN
   { to initialize out for the input layer }
    For k := 1 To N[0] do
       out[0, k] := InputVector[k];
   {Step through the layer from the input to the
output}
   For I := 1 To Layers DO BEGIN
      {Step through the nodes on layer I, where NI is
      the number of nodes in layer I}
       For J := 1 To N[I] DO BEGIN
       {sum over the nodes in the previous layer}

          s[I, J] := 0;
          For I := 1 To N[I - 1] DO
             s[I, J] := s[I, J] + weight[I, J, I] * out[I - 1, I];
          {add bias term}
          s[I, J] := s[I, J] + Bias[I, J];
          {if a sigmoid activation function is used}
          If s[I, J] > 100 Then  {limiting the values}
             s[I, J] := 100
          Else
          If s[I, J] < -100 Then  {limiting the values}
             s[I, J] := -100;
          out[I, J] := 1 / (1 + Exp(-s[I, J])); {sigmoid
         activation function}
       END;
   END;
END;
```

V. CONCLUSIONES

Se obtuvieron resultados satisfactorios en el entrenamiento de la RN, con alto nivel de generalización. Durante la operación reconoció todas las señales utilizadas, incluso las afectadas por ruidos. Ha significado un paso de avance satisfactorio para el desarrollo de algoritmos eficientes de alarma predictiva por tendencia con un consumo mínimo de tiempo de procesamiento. Actualmente el método ha sido aplicado utilizando el LabVIEW de National Instruments y DLL escritas en otros lenguajes de programación como C y DELPHI.

Como trabajos inmediatos y futuros se están utilizando señales afectadas por niveles de ruido más intensos. Adicionalmente, se comienza a trabajar para que la RN, además de reconocer el modelo más apropiado, haga una pre-estimación de los parámetros de dicho modelo, con lo cual el algoritmo de ajuste final, si fuera aún necesario utilizarlo, sería extremadamente eficiente, pues sus condiciones iniciales de operación serían los valores estimados por la RN. Se usarán patrones para estímulos de entrada tipo rampa.

REFERENCIAS

[1] National Instruments, LabVIEW, a graphical programming language to create applications, version 6.1, USA, 2002. www.ni.com/labview.

[2] K. Ogata, Ingeniería de control moderna, D.F. México, Prentice Hall, 1998.

[3] T. Södertröm, & P. Stoica, System identification, Englewood Cliffs, New Jersey, USA, Prentice-Hall, 1989.

[4] T.F. Edgar & D.M. Himmelblau, Optimization of chemical processes, New York, USA, MacGraw-Hill, 1988.

[5] D. M. Himmelblau, Applied nonlinear programming, New York, USA, MacGraw-Hill, 1972.

[6] A.V,. Oppenheim & A. Willsky, Señales y sistemas, D.F., México, Prentice-Hall Hispanoamericana, 1998.

[7] A.V Oppenheim, R.W. Schafer y J.R Buck, Tratamiento de señales en tiempo discreto, 2ª. edición Prentice-Hall, Inc.,Madrid, España, 2000, pp. 873.

[8] Borland Software Corporation, Borland Delphi version 4, 1998, Calif. USA. www.borland.com