Ninth LACCEI Latin American and Caribbean Conference (LACCEI'2011), Engineering for a Smart Planet, Innovation, Information Technology and Computational Tools for Sustainable Development, August 3-5, 2011, Medellín, Colombia.

Propuesta Metodológica Para la Creación de Modelos Neuronales de Caja Gris Utilizando Matlab

Francisco Cruz
Universidad Andrés Bello, Santiago, Chile, fcruz.cl@gmail.com

Gonzalo Acuña
Universidad de Santiago de Chile, Santiago, Chile, gonzalo.acuna@usach.cl

Guillermo Badillo
Universidad Andrés Bello, Santiago, Chile, gbadillo@unab.cl

RESUMEN

Los modelos de caja gris combinan ecuaciones diferenciales, que actúan como caja blanca y redes neuronales, utilizadas como caja negra para complementar el modelo fenomenológico. Estos modelos han sido utilizados en diversos trabajos comprobándose su eficacia. El presente trabajo tiene por objetivo proponer una metodología para la creación, entrenamiento y simulación de modelos neuronales de caja gris en Matlab a través del uso del toolbox de redes neuronales artificiales. Un completo ejemplo desde la representación hasta los resultados es expuesto en este trabajo.

Palabras claves: modelos de caja gris, parámetros variantes en el tiempo, redes neuronales, Matlab

ABSTRACT

Gray-box neural models mix differential equations, which act as white boxes, and neural networks, used as black boxes, to complete the phenomenological model. These models have been used in different researches proving their efficacy. The aim of this work is to propose a methodology for creation, training and simulation of Gray-box neural models with Matlab, using artificial neural networks toolbox.

Keywords: gray-box model, time-varying parameters, neural networks, Matlab

1. INTRODUCCIÓN

El presente trabajo tiene por objetivo mostrar el uso del toolbox de redes neuronales de Matlab, aplicación sobre la cual se soporta este trabajo, con el fin de diseñar y simular modelos neuronales de caja gris.

El toolbox de redes neuronales que Matlab incorpora, posee apoyo para el manejo de variadas estructuras de redes neuronales, por ejemplo, perceptrón simple, adaline, perceptrón multicapa, redes neuronales de base radial, redes de Hopfield, mapas autoorganizados, entre otras. Todas las redes neuronales antes mencionadas se soportan sobre una única estructura de red neuronal, la red neuronal personalizada (custom networks), la cual permite crear objetos de redes neuronales, los cuales en fases posteriores pueden ser entrenados y simulados.

Cada vez que se utiliza un comando para crear cualquier red neuronal antes mencionada, el comando que se ejecuta por debajo es el mismo, el comando network. Este comando permite crear redes neuronales personalizadas y cada llamada a una función para crear una red neuronal cualquiera, corresponde a un caso específico de utilizar este comando.

Para mostrar el uso de este comando para la creación de modelos neuronales de caja gris, se mencionará en forma breve estos modelos y luego se describirá un caso, el cual consiste el la creación un modelo de caja gris para la estimación de parámetros variantes en el tiempo de un proceso de CSTR.

2. MODELOS NEURONALES DE CAJA GRIS

Los modelos neuronales de caja gris, inicialmente fueron denominados como redes neuronales híbridas, hace ya casi dos décadas (Psichogios y Ungar, 1992). Son utilizados donde existe conocimiento a priori incompleto, o bien, se conocen algunas leyes del fenómeno, sin embargo, algunos parámetros se determinan a partir de datos observados.

Han sido utilizados en diversos trabajos obteniendo buenos resultados en comparación con modelos neuronales simples, debido al aprovechamiento del conocimiento a priori existente (Chen et. al, 2000), (Hensen et. al, 2000), (James et. al, 2000), (Acuña et. al, 2006), (Cruz et. al, 2007), (Cruz y Acuña, 2010).

Para el trabajo expuesto se utiliza un modelo neuronal de caja gris en serie (Thompson y Kramer, 1994), debido a que presenta mejor rendimiento comparado con el modelo en paralelo (Van Can et. al, 1996). El modelo de caja gris en serie consiste en una red neuronal intermedia que estima el valor de los parámetros desconocidos y luego son introducidos al modelo fenomenológico, como se muestra en la Figura 1a.

El modelo es entrenado y posteriormente simulado utilizando aprendizaje indirecto (Acuña et. al, 1999), el cual consiste en calcular el error a la salida del modelo fenomenológico y desde ahí retropropagar hacia el interior del modelo hasta la red neuronal, como se aprecia en la Figura 1b.

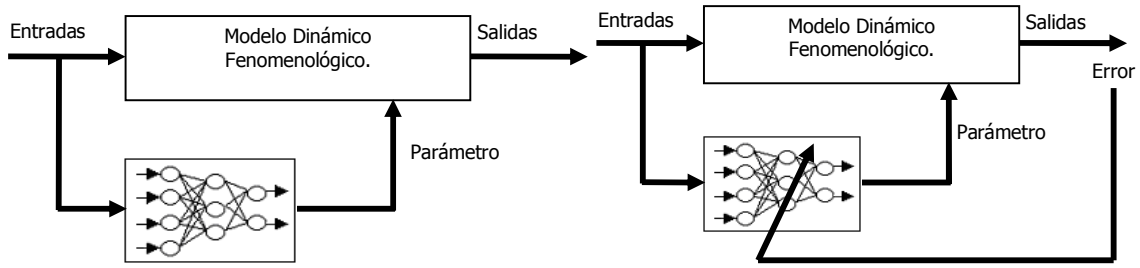

Figura 1a: Modelo neuronal de caja gris en serie. Figura 1b: Modelo neuronal de caja gris con entrenamiento indirecto

3. PROCESO CSTR

El proceso a modelar corresponde a la simulación de la reacción exotérmica de primer orden en un reactor continuo de tanque agitado (Continuous Stirred Tank Reactor, CSTR) (Hernández y Arkun, 1992). La entrada al sistema corresponde a la temperatura de la camisa de enfriamiento y la salida del sistema corresponde al grado de avance de la reacción.

Como los modelos neuronales de caja gris constan de dos partes, una parte fenomenológica que es representada por ecuaciones diferenciales y otra parte compuesta por un modelo empírico que es representado por una red neuronal, por lo tanto, el modelo fenomenológico se muestra en las siguientes ecuaciones (1), (2), (3).

$$x'_1 = -x_1 + 0,072 \cdot (1 - x_1) \cdot \rho \tag{1}$$

$$x'_2 = -x_2 + 8 \cdot 0,072 \cdot (1 - x_1) \cdot \rho + 0,3 \cdot (\mu - x_2) \tag{2}$$

$$y = x_1 \tag{3}$$

Donde ρ corresponde al parámetro con dificultad de obtener, el cual será estimado por la red neuronal. Sin embargo, para efectos experimentales y generación de datos, se utilizará la siguiente expresión mostrada en la ecuación (4).

$$\rho = e^{\left(\frac{x_2}{1+x_2/20}\right)} \tag{4}$$

Luego, el parámetro se asumirá completamente desconocido y se estimará utilizando la red neuronal entrenada en forma indirecta. A continuación, integrando las ecuaciones (1), (2) y (3) en el intervalo de tiempo t y t+1, y desarrollando las ecuaciones, para la construcción del modelo neuronal de caja gris se obtienen las ecuaciones discretas, las cuales se muestran en (5), (6) y (7).

$$x_{1(t+1)} = (1-\Delta t)x_{1(t)} + 0,072\rho\Delta t - 0,072\rho\Delta t x_{1(t)} \tag{5}$$

$$x_{2(t+1)} = (1-1.3\Delta t)x_{2(t)} + 8\cdot 0,072\rho\Delta t - 8\cdot 0,072\rho\Delta t x_{1(t)} + 0,3u\Delta t \tag{6}$$

$$y = x_1 \tag{7}$$

4. REPRESENTACIÓN DEL MODELO EN MATLAB

La solución propuesta es un modelo neuronal de caja gris, en donde, la parte fenomenológica de éste puede ser descrita en conjunto con la parte empírica del mismo, para lo cual es diseñada una red neuronal que contenga ambas partes del modelo de caja gris (Figura 2). Esta red neuronal híbrida posee la capacidad de fijar pesos en la fase de entrenamiento, para así actuar como un modelo de caja gris. Los pesos en la Figura 2 que poseen un valor señalado corresponden a la parte fenomenológica del modelo, de esta manera, estos pesos no serán modificados durante el entrenamiento. Los pesos en los que no se indica valor corresponden a la parte neuronal del modelo, además se puede apreciar que éstos presentan conexión de bias. Estos pesos, inicialmente, fueron asignados con valores pseudoaleatorios obtenidos con el método de inicialización de Nguyen-Widrow (Nguyen y Widrow, 1990).

En la Figura 2 se puede distinguir claramente que la parte neuronal del modelo es estimada por un perceptrón multicapa, que se encuentra inserto en el modelo neuronal de caja gris, el cual posee una arquitectura de 1 x 4 x 1. Su entrada corresponde sólo a X_2 debido a que como se observó en el modelo matemático el parámetro difícil de obtener tienen dependencia sólo con esta variable de estado.

En consecuencia, el perceptrón multicapa estima el valor del parámetro difícil de obtener, el cual se mezcla a su vez con la parte fenomenológica del modelo, para así obtener la salida de éste.

Para la parte neuronal del modelo se utiliza tangente hiperbólica como función de transferencia en la capa intermedia y función identidad en la capa de salida, mientras que para la parte fenomenológica se utiliza la función identidad como función de transferencia. La función de activación que se utiliza es la sumatoria de las ponderaciones de las entradas por los pesos, exceptuando la neurona inmediatamente después de la salida de la parte neuronal, donde se utiliza una productoria para la ponderación de las entradas por los pesos, esto debido a la forma de las ecuaciones fenomenológicas.

Para crear esta red personalizada se comienza con una red vacía, la cual es obtenida con el comando network. El objeto de red neuronal creado contiene múltiples propiedades que se pueden configurar para especificar la estructura de nuestra red. Más documentación, así como también ejemplos, pueden se encontrados en (Demuth y Beale, 1998).

El código necesario para crear el modelo neuronal de caja gris para la determinación de parámetros variantes en el tiempo de un proceso CSTR se muestra a continuación, con su respectiva explicación. A través del código que se muestra se puede crear el modelo de caja gris para su posterior entrenamiento y simulación, de la misma forma que cualquier estructura de red neuronal en Matlab. Para esto los datos de las dos variables de estado deben estar cargados en las variables x1 y x2, y la entrada en la variable u.

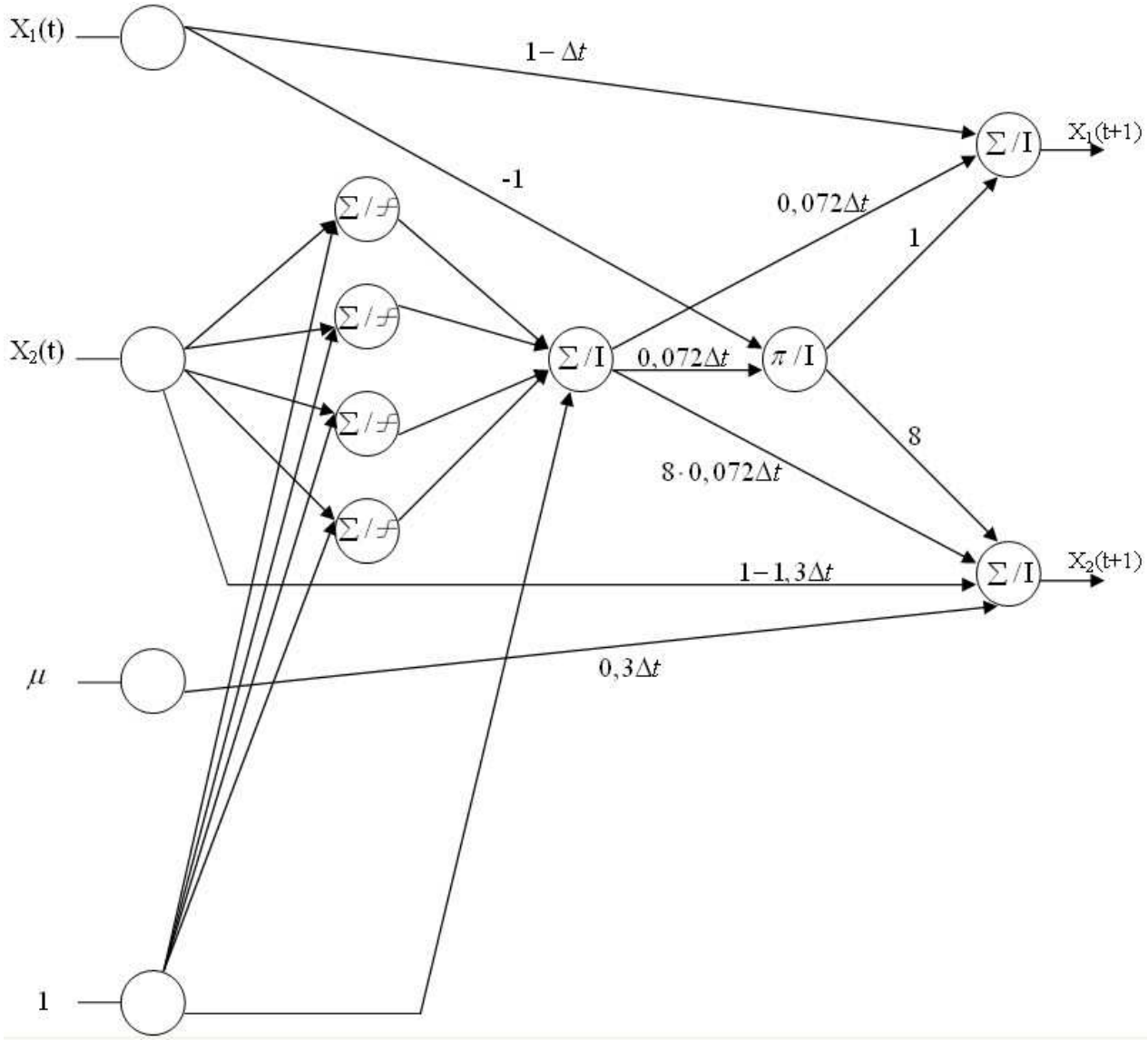

Figura 2: Modelo neuronal de caja gris para simulación de proceso CSTR con pesos fijos.

```
1  cajaGrisCstr = network;
```

La línea 1 del código es la encargada de inicializar y crear la estructura de red neuronal personalizada sobre la que se soportará el modelo de caja gris. Esta estructura posee múltiples propiedades la cuales serán determinadas a continuación, dependiendo de la estructura que tenga cada red.

```
2  cajaGrisCstr.numInputs = 3;
3  cajaGrisCstr.numLayers = 4;
```

En la línea 2 y 3 se especifican las primeras propiedades que debemos indicar para preparar el modelo, éstas corresponden al número de entradas y número de capas, respectivamente. En el caso de las entradas no se debe considerar el bias, ya que éste posee una estructura específica separada, la cual se verá a continuación. Mientras, en el caso de las capas, como se observa en la Figura 2, el modelo posee claramente 4 capas. Debemos recordar

que las neuronas de entrada no se consideran una capa, ya que solamente proporcionan las conexiones desde éstas a la capa intermedia, y en sí no son neuronas con un comportamiento como el resto.

```
4  cajaGrisCstr.biasConnect = [1;1;0;0];
```

La línea 4 del código muestra la propiedad con la cual indicamos que capa posee conexión de bias y cuales no. Para esto, es necesario que ya se encuentre definido la cantidad de capas que tiene el modelo, ya que esta estructura es un vector de largo igual a la cantidad de capas, es decir, en este caso 4. Como se puede observar en el código, las capas que poseen conexión de bias se señalan con un número 1, mientras las que no la poseen con un número 0, por lo que en el caso de este modelo la capa 1 y 2 poseen conexión de bias, lo cual es acorde a la Figura 2. Por otra parte, hay que notar que las capas que poseen conexiones de bias corresponden a la parte neuronal del modelo, que en definitiva es un perceptrón multicapa con 4 neuronas en su capa oculta y una neurona en su capa de salida.

```
5  cajaGrisCstr.inputConnect = [0 1 0;0 0 0;1 0 0;1 1 1];
```

En la línea 5 se declara la propiedad que indica las entradas que se conectan a cada capa, por lo que esta propiedad es una matriz de dimensión número de capas por número de entradas, en este caso 4x3, en donde en la posición i,j se especifica con un número 1 si la capa i posee pesos proveniente desde la entrada j, en cuyo caso el valor de los pesos será descrito en una estructura que se verá más adelante en el código. En caso de no existir peso desde la entrada j hacia la capa i se asigna 0 a la posición.

```
6  cajaGrisCstr.layerConnect = [0 0 0 0;1 0 0 0;0 1 0 0;0 1 1 0];
```

La línea 6 indica que capa se conecta con otra capa, por lo que esta propiedad es una matriz cuadrada de dimensión número de capas por número de capas, en el caso del modelo es 4x4. La posición i,j se especifica con un 1 si la capa i tiene pesos proveniente de la capa j, en cuyo caso los valores de los pesos se especifican en una estructura que también será descrita más adelante en conjunto con los pesos desde las entradas. Al igual que en caso anterior de no existir conexiones entre las capas, se debe asignar un valor igual a 0.

```
7  cajaGrisCstr.outputConnect = [0 0 0 1];
8  cajaGrisCstr.targetConnect = [0 0 0 1];
```

Las líneas 7 y 8 especifican las capas que poseen valores de salida y valores como objetivos (targets). En el caso de los valores de salida se asigna un valor igual a 1 a las capas que se desee observar el comportamiento posteriormente a través de simulaciones o gráficos. Para el caso de los valores objetivos, se asigna un valor igual a 1 a las capas que poseen valores medidos para la etapa de aprendizaje. En el modelo solamente la capa 4 es especificada como salida y con valores objetivos, ya que es la única capa de salida y es la única para la cual se poseen medidas.

```
9  cajaGrisCstr.initFcn = 'initlay';
```

La línea 9 señala la función de inicialización de los pesos. En el modelo, la función que se especifica muestra que cada capa será inicializada con su propia función de pesos.

```
10  cajaGrisCstr.inputs{1}.range = [min(x1) max(x1)];
11  cajaGrisCstr.inputs{2}.range = [min(x2) max(x2)];
12  cajaGrisCstr.inputs{3}.range = [min(u) max(u)];
```

Entre las líneas 10 y 12 se indican los valores de variación para cada una de las entradas, en este caso 3, en una estructura de vector de dimensión 2, indicando el mínimo y el máximo de los valores de cada entrada.

```
13  cajaGrisCstr.layers{1}.size = 4;
14  cajaGrisCstr.layers{1}.transferFcn = 'tansig';
15  cajaGrisCstr.layers{1}.netInputFcn = 'netsum';
16  cajaGrisCstr.layers{1}.initFcn = 'initnw';
```

En las líneas 13 a 16 se especifica la estructura que poseerán las neuronas de la primera capa. Como se ve en la línea 13 esta capa posee 4 neuronas (ver Figura 2) que corresponden a las neuronas ocultas del perceptrón multicapa en la parte neuronal del modelo. La función de transferencia utilizada en esta capa, como se observa en la línea 14, corresponde a tangente sigmoidal y la función de activación, se señala en la línea 15, es la sumatoria, es decir, la suma de las entradas por los pesos. Finalmente, en la línea 16 se ve la función de inicialización de los pesos para la primera capa, en el modelo la función corresponde a la inicialización pseudo aleatoria de Nguyen-Widrow (Nguyen y Widrow, 1990).

```
17  cajaGrisCstr.layers{2}.size = 1;
18  cajaGrisCstr.layers{2}.transferFcn = 'purelin';
19  cajaGrisCstr.layers{2}.netInputFcn = 'netsum';
20  cajaGrisCstr.layers{2}.initFcn = 'initnw';
```

Las líneas desde la 17 a la 20, de la misma forma que el caso anterior, especifican qué tipo de neuronas se utilizarán en la capa 2. En este caso la capa posee 1 neurona, función de transferencia identidad y función de activación sumatoria de pesos por las entradas. Al igual que antes, la función de inicialización corresponde al método de Nguyen-Widrow.

```
21  cajaGrisCstr.layers{3}.size = 1;
22  cajaGrisCstr.layers{3}.transferFcn = 'purelin';
23  cajaGrisCstr.layers{3}.netInputFcn = 'netprod';
```

En las líneas entre 21 y 23 se especifica la estructura de las neuronas de la capa 3. En este caso el tamaño de la capa es de 1 neurona, la función de transferencia es identidad y la función de activación, a diferencia de los casos anteriores, corresponde a la productoria (π) o multiplicación de los pesos por las entradas. En este caso no es necesario cambiar la función de inicialización debido a que los valores de los pesos se especificarán arbitrariamente y no serán modificados durante el entrenamiento, ya que corresponden a la parte fenomenológica del modelo y poseen pesos fijos.

```
24  cajaGrisCstr.layers{4}.size = 2;
25  cajaGrisCstr.layers{4}.transferFcn = 'purelin';
26  cajaGrisCstr.layers{4}.netInputFcn = 'netsum';
```

En las líneas 24 a 26 se detalla la estructura de las neuronas de la capa 4. Esta capa posee 2 neuronas, utiliza función de transferencia identidad y función de activación sumatoria de los pesos por las entradas.

```
27  cajaGrisCstr.performFcn = 'mse';
28  cajaGrisCstr.trainFcn = 'trainlmGris';
```

Finalmente, la línea 27 indica qué función de desempeño (performance) será utilizada, en este caso se utiliza la función de error cuadrático medio y la línea 28 indica que función de entrenamiento se utilizará en la fase de aprendizaje. En este caso la función de entrenamiento corresponde al algoritmo modificado de Levenberg-Marquardt que permite mantener pesos fijos en la fase de entrenamiento.

Con lo anterior, la red se encuentra completamente creada, pero aún no está lista para su entrenamiento y simulación, para esto se tendrá que señalar que pesos se mantendrán fijos en la fase de aprendizaje, para así completar el modelo neuronal de caja gris. Este proceso se tendrá que realizar previamente al entrenamiento de la red. Luego de inicializado los pesos fijos, el entrenamiento y la simulación del modelo se realizan de la misma forma que con cualquier red neuronal en Matlab. La forma de definir que pesos serán fijos y los valores que éstos poseerán se muestran en el siguiente código:

```
vectorDePesosFijos = ['-1            ';
                      '1 - deltat    ';
                      '0             ';
                      '0             ';
                      '1 - 1.3*deltat';
                      '0             ';
                      '0.3*deltat    ';
                      '0.072*deltat  ';
                      '0.072*deltat  ';
                      '8*0.072*deltat';
                      '1             ';
                      '8             '];
ubicacionesPesosFijos = ['IW{3,1}(1)';
                         'IW{4,1}(1)';
                         'IW{4,1}(2)';
                         'IW{4,2}(1)';
                         'IW{4,2}(2)';
                         'IW{4,3}(1)';
                         'IW{4,3}(2)';
                         'LW{3,2}(1)';
                         'LW{4,2}(1)';
                         'LW{4,2}(2)';
                         'LW{4,3}(1)';
                         'LW{4,3}(2)'];
```

Como se observa en el código se deberán declarar dos vectores de igual tamaño, en donde en uno de ellos se especifica los valores que tendrán los pesos fijos, y en otro vector, en posiciones equivalentes, la ubicación que tiene el peso fijo en el modelo de caja gris. Estos valores de los pesos son los que se observan en el modelo planteado en la Figura 2. Se debe diferenciar entre los pesos que van desde una entrada a una capa y los pesos que van desde una capa a otra capa, ya que como se describió anteriormente existen estructuras distintas para indicar esto. Los pesos que provienen desde las entradas se almacenan en una estructura llamada "IW" y los pesos entre las capas se almacenan en otra estructura llamada "LW".

De esta forma, el primer peso indicado en el código con un valor igual a -1 es el peso que va desde la entrada 1 a la capa 3, como se indica en la primera posición del segundo vector que contiene las ubicaciones dentro del modelo. El segundo peso indicado en el primer vector con valor (1 – deltat) corresponde al peso que va desde la entrada 1 a la primera neurona de la capa 4. En este caso el peso desde la entrada 1 a la segunda neurona de la capa 4 se fija en un valor igual a 0, señalado en el tercer valor del vector, debido a que no presenta conexión en el diagrama de la Figura 2. Así mismo se continúan definiendo los pesos desde las entradas a las capas.

En el caso de los pesos entre las capas, se puede observar el penúltimo valor del Vector de Pesos Fijos que es 1, su correspondiente en el segundo vector indica que el peso va desde la capa 3 a la primera neurona de la capa 4. Finalmente, la última posición del vector tiene un valor 8, en su correspondiente del segundo vector se indica que es el peso que va desde la capa 3 hasta la segunda neurona de la capa 4. De esta misma forma se definen todos los pesos para todas las conexiones existentes entre las capas.

5. RESULTADOS Y VALIDACIÓN

El la red se utiliza como función de performance el error cuadrático medio, que es calculado en la salida del modelo fenomenológico y desde ahí retropropagado hacia los pesos de la red que son modificables. En la fase de entrenamiento la entrada utilizada es una señal binaria pseudoaleatoria, mientras que en la etapa de simulación la entrada utilizada es una señal sinusoidal.

El algoritmo de entrenamiento corresponde al método de Levenberg-Marquardt que es un algoritmo de segundo orden que presenta una ligera modificación sobre el método tradicional de Newton, debido a que posee la capacidad de modificar sólo los pesos que le sean señalados, dejando así un grupo de pesos fijos en la fase de entrenamiento que, como ya se mencionó, representan la parte fenomenológica del modelo.

El modelo fue entrenado durante 1000 épocas en varias ocasiones, seleccionando el mejor. Es importante notar que el número de épocas de entrenamiento puede aumentar en este tipo de modelos, ya que al poseer pesos fijos es una restricción que se añade al algoritmo de optimización.

Posteriormente, en la fase de simulación se calculan índices error para comprobar la calidad de los resultados, éstos son IA (índice de adecuación), RMS (raíz cuadrada media) y RSD (desviación estándar residual), los valores considerados aceptables para estos índices son IA>0.9, RMS <0.1 y RSD<0.1. Las ecuaciones de los índices de calidad se muestran en la ecuación (8).

$$IA = 1 - \frac{\sum_{i=1}^{n} (o_i - p_i)^2}{\sum_{i=1}^{n} \left(|o_i'| + |p_i'| \right)^2} \quad RMS = \sqrt{\frac{\sum_{i=1}^{n} (o_i - p_i)^2}{\sum_{i=1}^{n} o_i^2}} \quad RSD = \sqrt{\frac{\sum_{i=1}^{n} (o_i - p_i)^2}{N}} \tag{8}$$

Para la simulación se realizaron distintas pruebas, acá se expondrán pruebas para datos con un 5% de ruido. La simulación se realizó para los 200 datos disponibles. El valor utilizado para la simulación de Δt fue de 0.005, mientras los valores iniciales para las variables de estado son $X_1 = 0.1$ y $X_2 = 0.8$.

En la Figura 3 se muestra la respuesta de la red a la simulación OSA (one step ahead), donde los datos son refrescados en cada paso. La línea continua muestra la salida real del sistema y la línea segmentada muestra la salida estimada por el modelo. Ambas salidas concuerdan por lo que el modelo responde satisfactoriamente.

En la Figura 4 se aprecia la simulación MPO (model predictive output), donde a partir del vector de estado inicial todos los valores son retroalimentados hacia la entrada. En la figura se muestra en línea continua la salida real del sistema. En línea segmentada se aprecia la salida simulada. En este caso la salida estimada no presenta ruido, actuando como filtro. Esto se debe a que las simulaciones MPO sólo consideran el vector inicial y luego a partir de éste iteran el modelo para realizar todas las simulaciones.

Los índices de calidad utilizados para validar la salida estimada con respecto a la salida real del sistema con ruido, se muestran en la Tabla 1. Todos los índices de calidad resultantes para estas simulaciones están dentro de los rangos aceptables, por lo que el modelo neuronal de caja gris ajusta adecuadamente la salida deseada para el caso de un 5% de ruido en los datos.

Tabla 1: Índices de calidad de la salida estimada, para OSA y MPO, simulado con 5% de ruido.

	OSA	MPO
IA	0.9975	0.9948
RMS	0.0080	0.0118
RSD	0.0012	0.0017

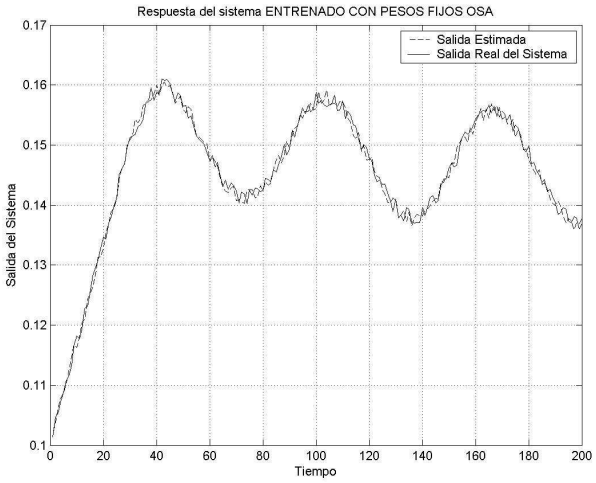

Figura 3: Respuesta del modelo entrenado con pesos fijos, y simulado con 5% de ruido OSA.

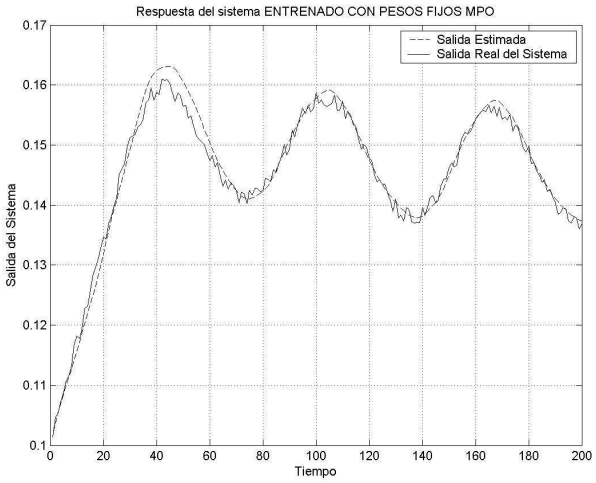

Figura 4: Respuesta del modelo entrenado con pesos fijos, y simulado con 5% de ruido MPO.

6. CONCLUSIONES

Los modelos neuronales de caja gris poseen reales ventajas con respecto a los de caja negra, ya que permiten aprovechar el conocimiento a priori que se posea de un sistema. Además apoyan a los modelos de caja blanca estimando parámetros difíciles de obtener o bien que no pueden ser obtenidos directamente en línea.

La metodología propuesta ha sido probada en un ejemplo mostrando los resultados esperados e introduciendo una forma didáctica para manipular este tipo de modelos.

La propuesta de creación, entrenamiento y simulación para modelos de caja gris utilizó entrenamiento indirectocon retropropagación y Levenberg-Marquardt, la cual muestra buenos resultados en la fase de aprendizaje y posterior simulación sólo con las variables medibles a manera de sensor virtual.

Los índices de calidad obtenidos posteriormente al entrenamiento cumplen con las expectativas, lo que se refleja en la posterior validación gráfica de la salida estimada por el modelo neuronal de caja gris en comparación con la salida real del sistema.

AGRADECIMIENTOS

Los autores desean agradecer el financiamiento parcial de Proyecto Fondecyt 1090316

REFERENCIAS

Acuña, G., Cubillos, F., Thibault, J., Latrille, E., *"Comparison of methods for training grey-box neural network models"*, Computers and Chemicals Engineering Supplement, 561-564 (1999)

Acuña, G., Cruz, F., Moreno, V., *"Identifiability of Time varying Parameters in a Grey-Box Neural Model: Application to a Biotechnological Process"*, Biannual Foodsim Conference, University of Naples Federico II, Naples, Italy (2006).

Chen, L., Bernard, O., Bastin, G., Angelov, P., *"Hybrid modelling of biotechnological processes using neural networks"*. Control Engineering Practice 8, pp. 821-827 (2000)

Cruz, F., Acuña, G., Cubillos, F., Moreno, V., Bassi, D., *"Indirect Training of Grey-Box Models: Application to a Bioprocess"*, Fouth International Symposium on Neural Networks, Mandarin Garden Hotel, Nanjing, China (2007).

Cruz, F., Acuña, G., *"Indirect Training with Error Backpropagation in Gray-Box Neural Model: Application to a Chemical Process"*, Proceeding of the SCCC XXIX International Conference, IEEE Press, Universidad Católica del Norte, Antofagasta (2010).

Demuth, H., Beale, M., *"Neural Network Toolbox For Use with Matlab®"*. Mathworks, Fifth printing, Version 3 (1998).

Hensen, R., Angelis, G., van de Molengraft, M., de Jager, A., Kok, J., *"Grey-box modeling of friction: An experimental case-study"*. European Journal of Control 6, 258-267 (2000)

Hernández, E.; Arkun, Y., *"Study of the Control-Relevant Properties of Backpropagation Neural Network Models of Nonlinear Dynamical Systems"*, Computers Chemical Engineering, Vol. 16, n° 4, 227-240 (1992)

James, S., Legge, R., Budman, H., *"Comparative study of black-box and hybrid estimation methods in fed-batch fermentation"*. Journal of Process Control 12, 113-121 (2000)

Nguyen, D., Widrow, B., *"Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights"*. Proceedings IEEE IJCNN, 21-26, San Diego (1990)

Psichogios, D., Ungar, L., *"A Hybrid Neural Network-First Principles Approach to Process Modeling"*, Vol. 38, Nº10, 1499-1511 (1992)

Thompson, M.; Kramer, M., *"Modeling Chemical Processes Using Prior Knowledge and Neural Networks"*, Vol. 40, Nº8, 1328-1340 (1994)

Van Can, H., Hellinga, C., Luyben, K., Heijnen, J., *"Strategy for Dynamic Process Modeling Based on Neural Network in Macroscopic Balance"*, AIChE 42, 3403-3418 (1996)

Autorización y Renuncia

Los autores autorizan a LACCEI para publicar el escrito en las memorias de la conferencia. LACCEI o los editores no son responsables ni por el contenido ni por las implicaciones de lo que esta expresado en el escrito.