Práctico de Redes Neuronales

Versión resumida, para servir de ayuda al framework nn22 para Octave (o Matlab)

Profesor: Dr. Ing. Daniel Patiño

Alumno: Ing. Alejandro Quiroga Alsina

2014, primer semestre.

Tabla de Contenidos

Parte 1: Conceptos básicos

Muchos de los conceptos expuestos a continuación fueron tomados del libro "**Neural Networks**" de Simon Haykin, o fueron dictados por en el curso de especialización de posgrado "**Redes Neuronales Aplicadas a la Identificación y Control de Sistemas**" dictado por el Dr. Ing. Daniel Patiño en el Instituto de Automática INAUT de la Universidad Nacional de San Juan (UNSJ), Argentina.

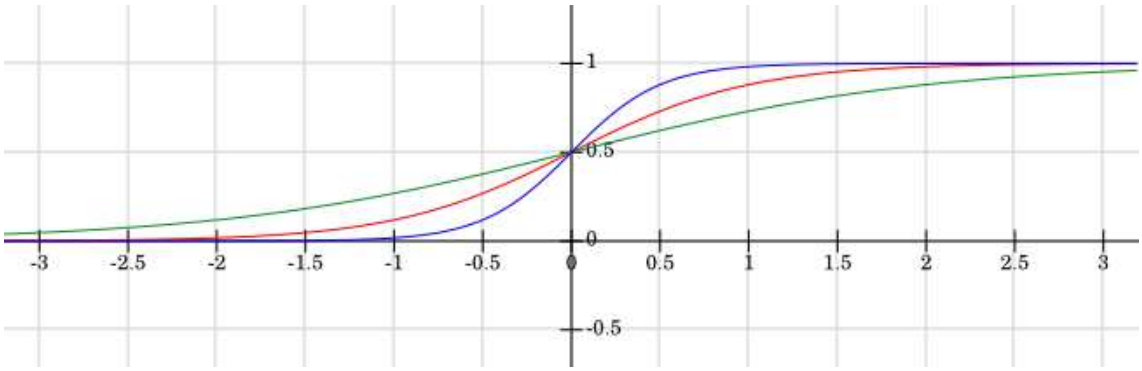
1.1 Función logística

Un ejemplo de la función logística se define como:

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

...cuyos valores límites son 0 y 1.

Gráfica de la función:



para **a =1, 2, 4** (verde, rojo, azul).

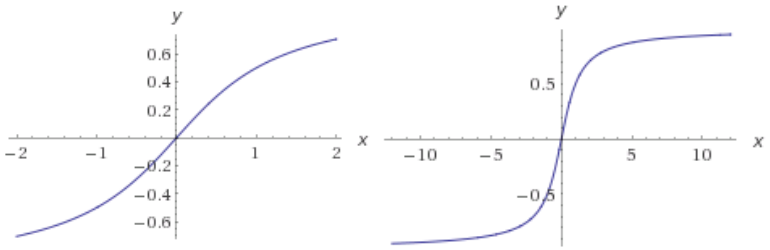
1.2 Función sigmoide

Una función sigmoide singular está definida por:

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

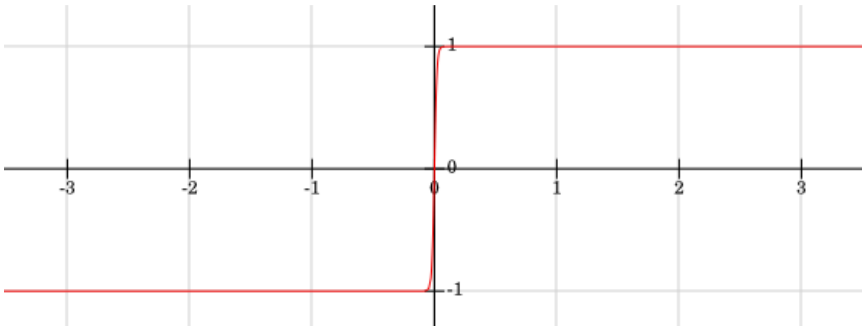
$$= \tanh\left(\frac{ax}{2}\right)$$

...en donde *tanh* significa *tangente hiperbólica*. Los valores límites para esta función son -1 y 1.

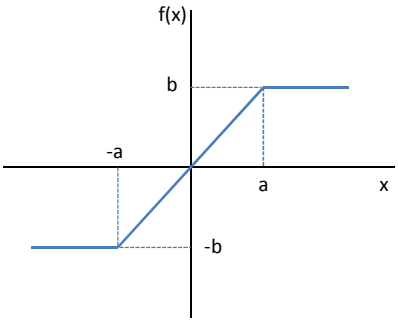


Suponiendo que el parámetro de pendiente ***a*** se hace infinitamente grande, ¿cuál es la forma resultante de ***f(x)***?

La forma de la función se asemejará a un escalón, con pendiente infinita en el origen de abscisas y valores -1 y + 1 en el plano negativo y positivo de abscisas respectivamente:



1.3 Función de activación pseudo-lineal



$$f(x) = \begin{cases} b, & x > a \\ \left(\frac{b}{a}\right)x, & -a \leq x \leq a \\ -b, & x < -a \end{cases}$$

Si se permite que ***a*** se aproxime a cero la función se convierte en un escalón.

1.4 Normalización

Una neurona tiene una función de activación ***f(x)*** definida por la siguiente función logística:

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

...en donde ***x*** es el campo local inducido y el parámetro ***a*** está disponible para ajustar la pendiente.

Sean $x_1, x_2, x_3, ..., x_m$ las señales de entrada aplicadas a los nodos fuente de la neurona y sea ***b*** su polarización. Para una presentación más conveniente, deseamos absorber el parámetro ***a*** en el modelo de campo local inducido escribiendo la función de la siguiente forma:

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

¿Cómo modificaría las entradas $x_1, x_2, x_3, ..., x_m$ para producir la misma salida que lograba anteriormente?

Se debe normalizar las entradas multiplicándolas por un parámetro ***a'*** de forma lineal o eventualmente en forma de potencia de sintonía. De esa forma, las entradas quedarán representadas mediante la siguiente serie:

$a.x_1, a.x_2, a.x_3, ..., a.x_m$; o eventualmente: $(x_1)^a, (x_2)^a, (x_3)^a, ..., (x_m)^a$.

Esto logra que los valores de entrada a la neurona se expandan sobre el eje x en el campo de entrada de la función de activación, simulando que la pendiente de la curva en el origen ha aumentado.

Es posible ver en la siguiente función de activación que...

$$f(x) = \tanh\left(\frac{ax}{2}\right)$$

...el factor ***a*** está multiplicando en forma directa a la variable de campo ***x***; quitarlo de esa función y agregarlo en la entrada es una tarea equivalente.

1.5 Evaluación de salidas de funciones de activación

Una neurona ***j*** recibe entradas de otras cuatro neuronas cuyos niveles de actividad son 10, -20, 4 y -2. Los pesos sinápticos de la neurona ***j*** son, respectivamente, 0.8, 0.2, -1.0 y -0.9. Calcule la salida de la neurona ***j*** para las dos situaciones siguientes:

a) La neurona es lineal.
b) La neurona tiene una función de activación pseudo-lineal con límites en ± 1.
c) La neurona tiene la siguiente función de activación logística:

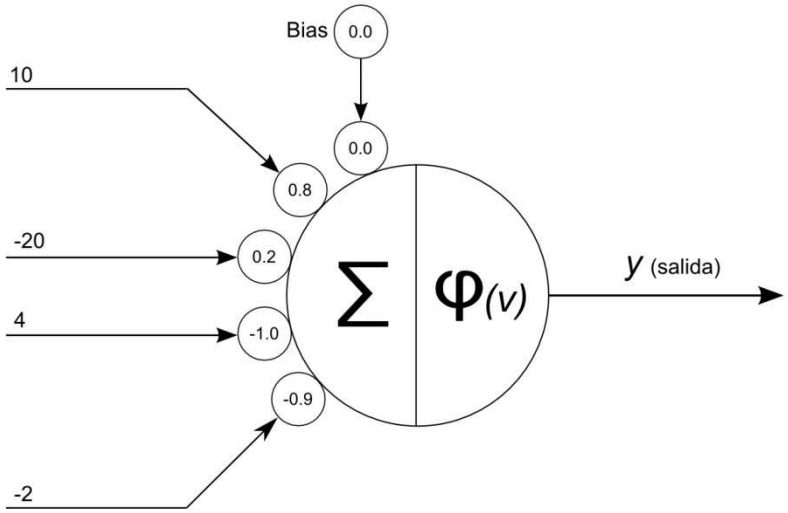
$$\varphi(v) = \frac{1}{1 + e^{-v}}$$

d) La neurona tiene la siguiente función de activación sigmoide:

$$\varphi(v) = \tanh(v)$$

Asuma que la polarización aplicada a la neurona es cero.

Modelo de la neurona con una función lineal por partes (no específica al problema) en la salida:



Sum(x_i) *= 10 * 0.8 - 20 * 0.2 - 4 * 1.0 + 2 * 0.9 = 1.8*

Luego, para el modelo a) si la salida es lineal sin límites, ***φ(v) = 1.8***

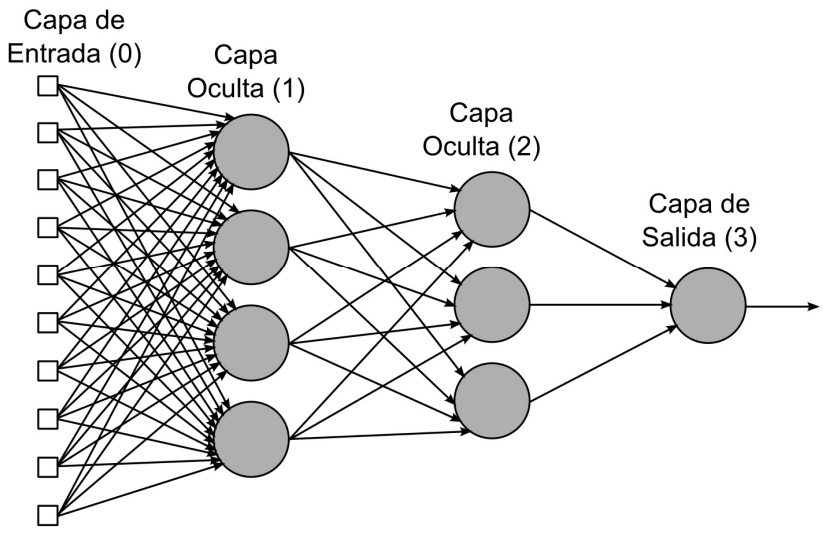
Para el modelo b) con una función pseudo-lineal, la salida es ***φ(v) = 1***

Para el modelo c) con una función logística, la salida es ***φ(v) = 0.8581***

Para el modelo d) con una función sigmoide, la salida es ***φ(v) = 0.9468***

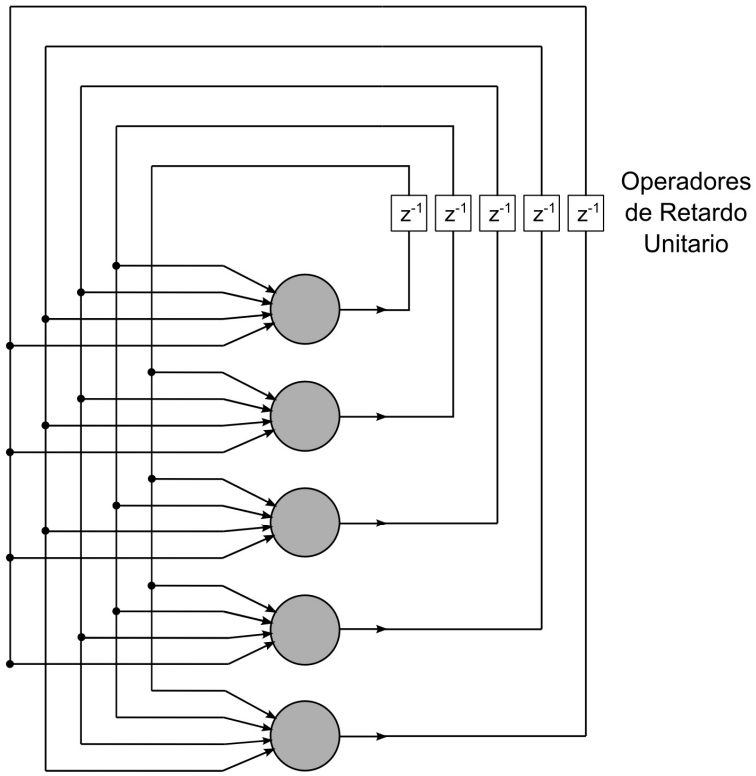
1.6 Estructura de red neuronal básica multicapa

La siguiente red feedforward completamente conectada posee 10 nodos de entrada, dos capas ocultas, una con cuatro neuronas y otra con tres neuronas, y una sola neurona de salida.



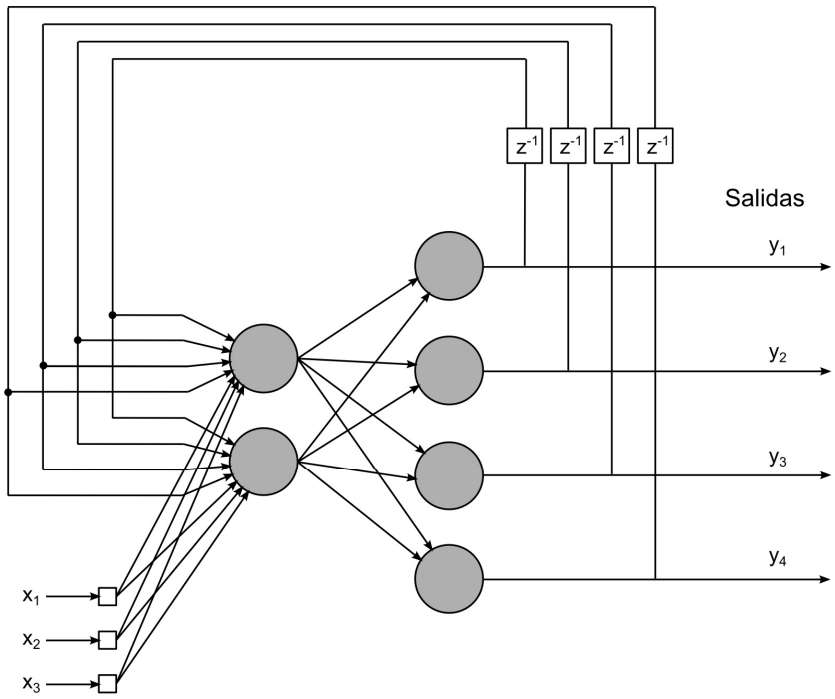
1.7 Estructura de red neuronal recurrente

Una red completamente recurrente con 5 neuronas.



1.8 Estructura de red neuronal recurrente con entradas

Una red recurrente con 3 nodos fuente, 2 neuronas ocultas (esto significa el uso real de 5 neuronas como capa de entrada, o capa cero) y 4 neuronas de salida.



Parte 2: Ejercicios de programación

En esta parte del trabajo práctico se utiliza la aplicación **Octave**, software de licencia libre GNU y de código abierto (FOS), cuyo comportamiento es equivalente a Matlab, exceptuando el uso de los Toolbox que son propietarios de la empresa Mathworks, que no están disponibles para Octave.

En http://es.wikipedia.org/wiki/GNU_Octave es posible encontrar algunas de las ventajas del uso de esta aplicación.

Todos los ejercicios fueron desarrollados en Octave, para lo cual se programó una herramienta de uso genérico que permite la construcción de redes neuronales de estructura Multicapa Feedforward genérica, con retardos de entrada y realimentación de salida también con retardos.

La estructura de las redes está definida en un archivo con extensión .ini que contiene:

a) Cantidad de capas de neuronas, incluyendo entrada y salida.
b) Cantidad de neuronas por capa.
c) Función de activación utilizada por cada capa de neuronas.
d) Posibilidad de realimentación de salida (vector de salida).
e) Número de retardos de la señal de entrada (vector de entrada).
f) Número de retardos en la realimentación de señal de salida (vector de salida).

En cada ejercicio se incluye la definición de estructura de red utilizada para su solución.

Las funciones principales del grupo de herramientas son la función **nn_feedforwardNN** y la función **nn_trainNN**, y ambas pueden ser utilizadas en forma separada del menú principal del ambiente de trabajo, en combinación con otras aplicaciones. Esto permite, por ejemplo, entrenar una red neuronal y luego utilizarla por separado del resto del ambiente, sólo alimentando vectores de entrada y obteniendo como respuesta vectores de salida de la red entrenada.

El resultado del entrenamiento de la red neuronal se almacena en un archivo con extensión .mat que es editable (con un editor de textos común) y simultáneamente compatible con formatos de Matlab.

Ejercicio 1:

Programar la función lógica AND en una red neuronal con una capa.

Respuesta:

Para entrenar esta red se generaron dos conjuntos de datos, uno para entrenamiento y otro para prueba de la red entrenada.

En el primer caso se utilizaron los valores puros binarios de dos entradas y el resultado binario de la salida, en una tabla AND. En el segundo caso se generaron valores aleatorios alrededor de los vectores de entrada y se alimentó con éstos a la red para verificar su salida.

```
% ~~~~~~  Definicion de estructura de NN  ~~~~~~
% ~~~~~~ Red Neuronal -- Curso INAUT 2014 ~~~~~~
%
%     (El comentario de encabezado es opcional)
%
% Columnas son:
%   1. Capa#, (0=entrada, #=oculta, max(#)=salida)
%   2. # de Neuronas en esa capa (sin Bias) = #input + FIR*#input + FB*#output
%   3. Función de Activación (linear, logistic, tanh)
%   4. #retardos de entrada FIR:
%          default=0, <= ((columna_2-#input)-FB*#output)/#input
%      #retardos de realimentación de salida FB FIR:
%          default=0, <= ((columna_2-#input)-FIR*#input)/#output
%   5. Nombre de Red(pares #in/#out) p.ej.: XOR(2/1), SIN(1/1), mcc(2/2), etc.
%
% --------------{fin de encabezado}--------------
0, 2, none, 0, AND(2/1)
1, 2, logistic, ,
2, 1, logistic, 0,
```

Se observa que la red tiene tres capas, la primera con dos neuronas de entrada y sin función de activación (es la capa de entrada), la segunda capa es oculta, con dos neuronas y con función de activación logística y la tercera capa es la de salida, con una neurona y con función de activación logística.

También se puede ver en la cuarta columna que no existen retardos de entrada FIR ni realimentaciones de salida, son los dos ceros luego de la definición de función de activación en las capas de entrada y salida.

Definición de conjuntos de entrenamiento:

En la *Figura a1* se observan cuatro puntos de entrenamiento y en la *Figura a2* cuatro conjuntos de diez puntos de entrenamiento para alimentar cada par de entrada.

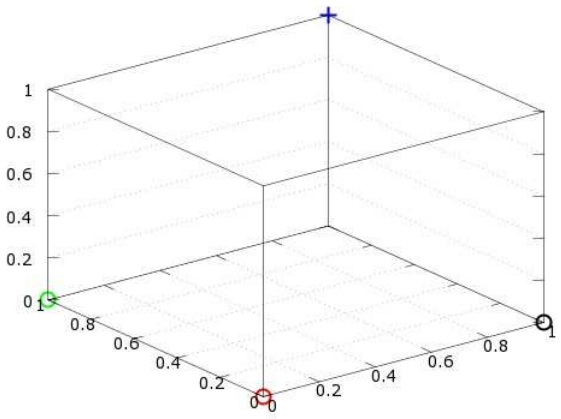


Figura a1

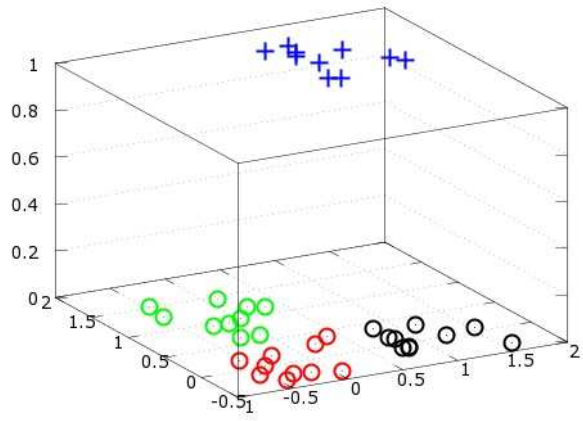


Figura a2

En ambas figuras se puede ver las entradas en los ejes `x` e `y` del plano base y la salida deseada en el eje `z`, como elevación en el espacio. Cuando un conjunto es utilizado para probar la red entrenada sólo el vector de entrada `(x,y)` es utilizado y la salida real de la red `f(x,y)` se compara con el valor de referencia, `z`.

Se grafica el resultado del entrenamiento de 150 iteraciones, con presentación de épocas (conjunto de datos de entrenamiento) en forma ordenada y sucesiva. El entrenamiento se estabiliza en un error fijo menor a 0.24, como se observa en la *Figura a3*. En la *Figura a4* se presenta la respuesta de la red entrenada frente a los datos de entrada utilizados en el entrenamiento.

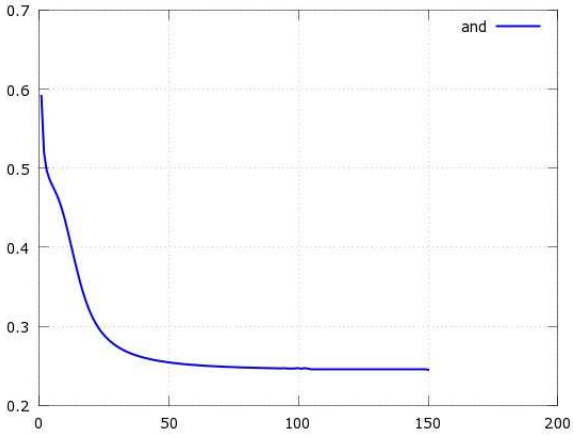


Figura a3

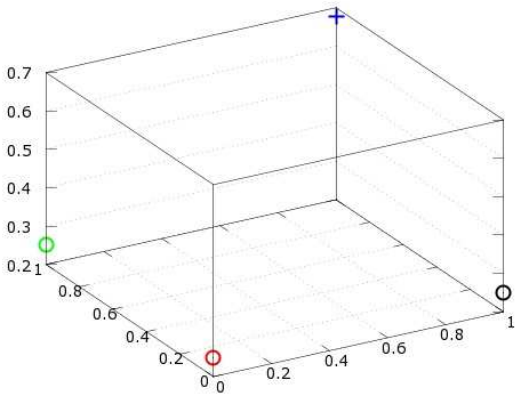


Figura a4

En las siguientes figuras, 5 a 11, se observa la respuesta de la red a los datos de prueba. Los resultados son presentados girando la gráfica en sentido antihorario y lateral, y luego vista superior.

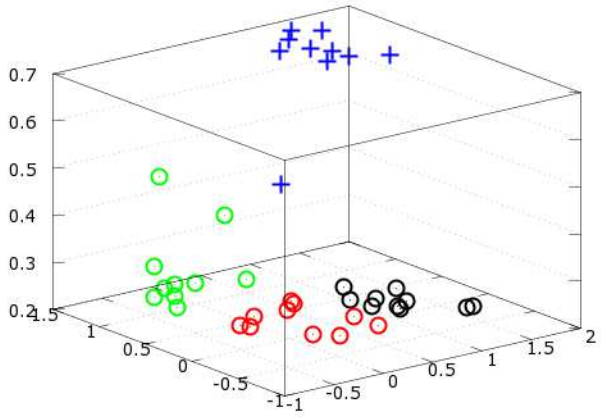


Figura a5

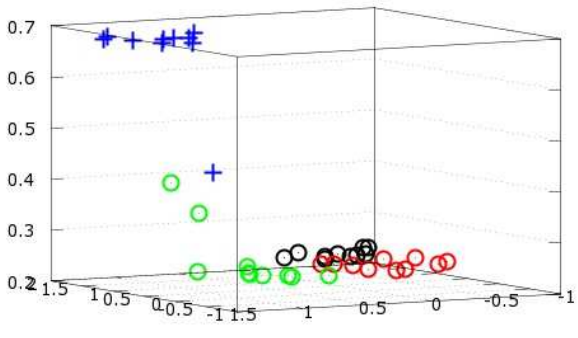


Figura a6

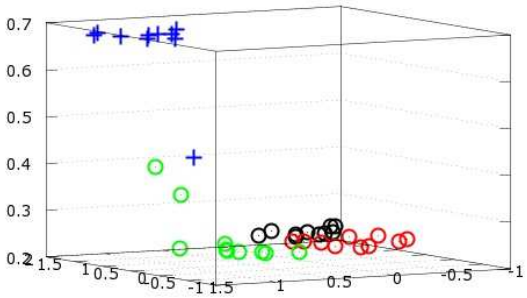


Figura a7

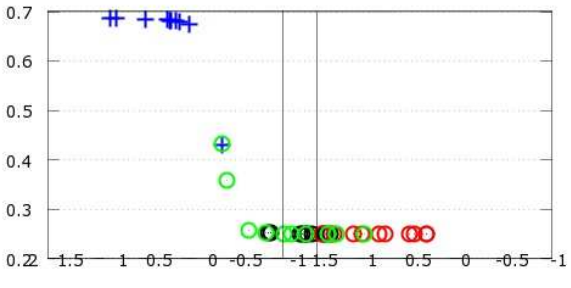


Figura a8

En la *Figura a8* se ve con claridad un falso positivo en el grupo de los resultados que deberían quedar por encima de 0.5, los puntos graficados como signos **+** con color azul.

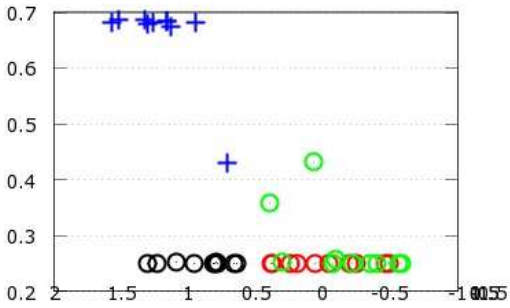


Figura a9

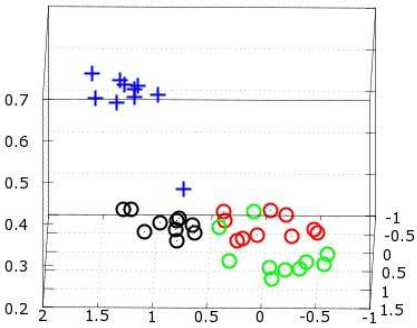


Figura a10

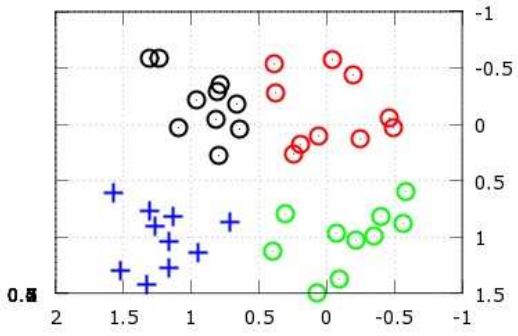


Figura a11

Ejercicio 2:

Programar la función lógica XOR en una red neuronal con una capa.

Respuesta:

Para entrenar esta red se generaron nuevamente dos conjuntos de datos, uno para entrenamiento y otro para prueba de la red entrenada.

En el primer caso se utilizaron los valores puros binarios de dos entradas y el resultado binario de la salida, en una tabla XOR. En el segundo caso se generaron valores aleatorios alrededor de los vectores de entrada y se alimentó con éstos a la red para verificar su salida.

```
% ~~~~~~  Definicion de estructura de NN  ~~~~~~
% ~~~~~~ Red Neuronal -- Curso INAUT 2014 ~~~~~~
%
0, 2, none, 0, XOR(2/1)
1, 2, logistic, ,
2, 1, linear, 0,
```

Se observa que la red tiene tres capas, la primera con dos neuronas de entrada y sin función de activación (es la capa de entrada), la segunda capa es oculta, con dos neuronas y con función de activación logística y la tercera es de salida, con una neurona y con función de activación lineal.

Definición de conjuntos de entrenamiento:

En la *Figura b1* se observan cuatro puntos de entrenamiento y en la *Figura b2* cuatro conjuntos de diez puntos de entrenamiento para alimentar cada par de entrada.

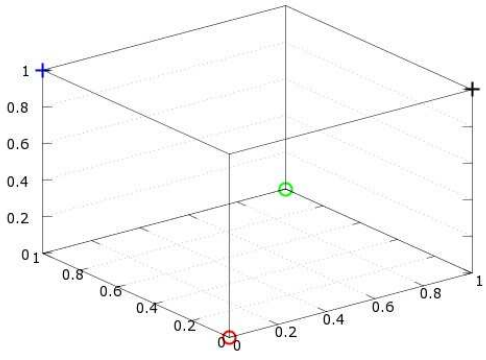


Figura b1

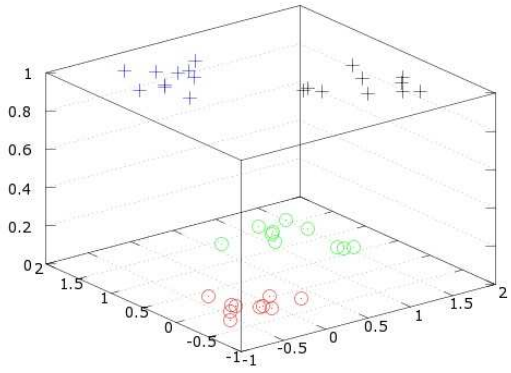


Figura b2

En ambas figuras se puede ver las entradas en los ejes x e y del plano base y la salida deseada en el eje z, como elevación en el espacio. Cuando un conjunto es utilizado para probar la red entrenada sólo se alimenta el vector de entrada (x, y) y la salida real de la red se compara con el valor deseado.

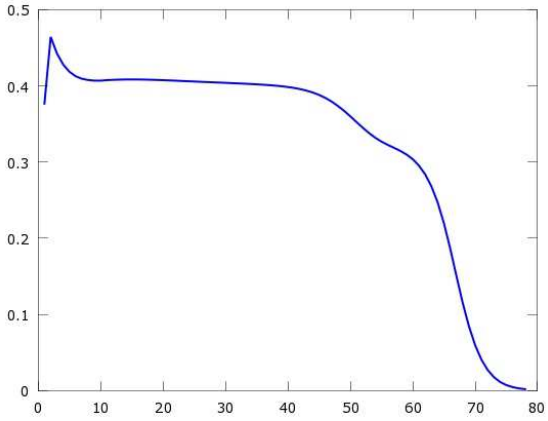


Figura b3

Se muestra el resultado del entrenamiento de 78 iteraciones, con presentación de épocas en forma ordenada y sucesiva. El entrenamiento se estabiliza en un error fijo menor a 0.01.

A continuación se grafican los resultados de la presentación de los datos de prueba a la entrada de la red entrenada. Es posible ver, en la rotación de las imágenes, que existen dos falsos positivos en el grupo de pares (1, 1), representados por los círculos de color verde y cuya salida está por encima del valor medio 0.5.

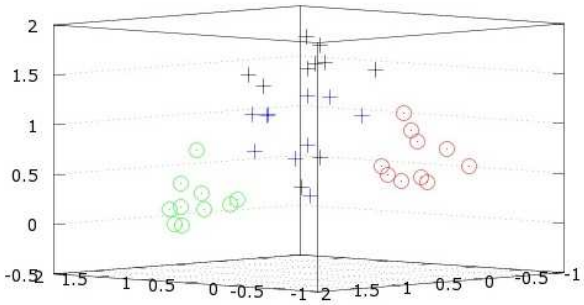


Figura b4

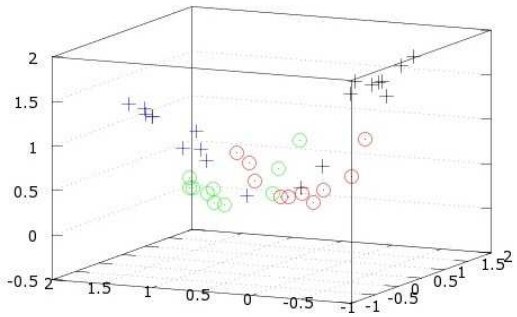


Figura b5

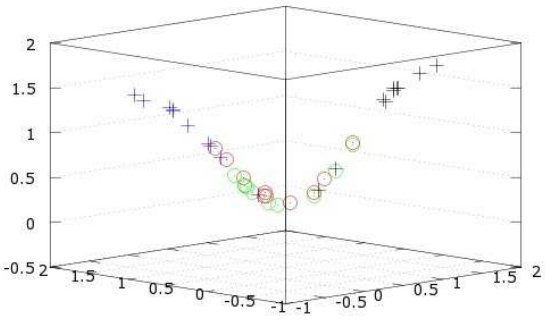


Figura b6

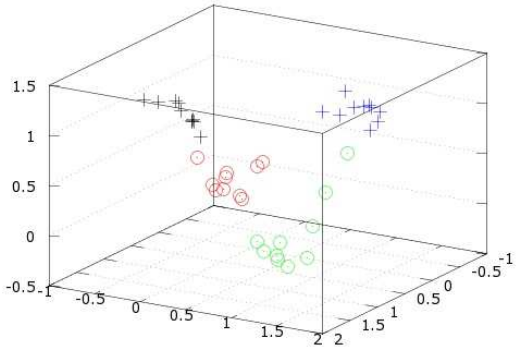


Figura b7

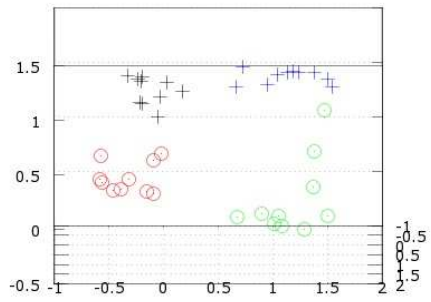


Figura b8

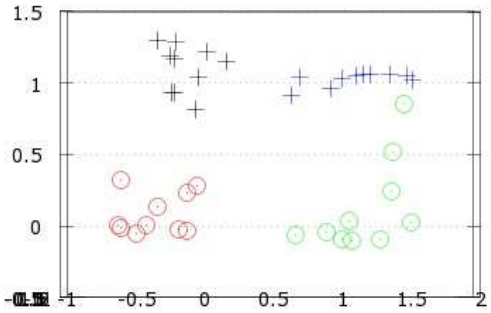


Figura b9

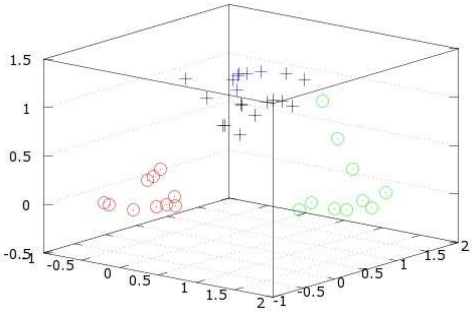


Figura b10

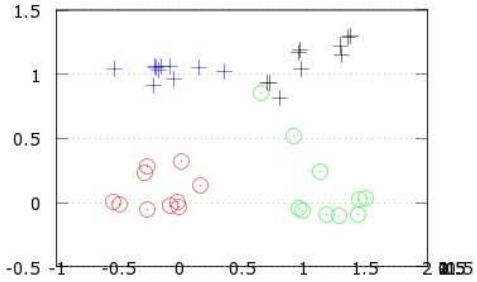


Figura b11

A modo de ejemplo, para esta red XOR, se muestra el contenido de los datos almacenados en el modelo de red que permite ver una "radiografía" de la estructura de datos, almacenada en el archivo `xor_nnData.mat`:

```
debug> nnData
nnData =

    FilesBase = xor

    Files =
        StructIni = xor_Structure.ini
        ioInputTrain = xor_ioInputTrain.txt
        ioInputTest = xor_ioInputTest.txt
        ioIdealTrain = xor_ioIdealTrain.txt
        ioIdealTest = xor_ioIdealTest.txt
        nnData = xor_nnData.mat

    Structure =
        layerNumber =
           0
           1
           2
        layerNeurons =
           2
           2
           1
        layerActivation =
        {
          [1,1] = none
          [2,1] = logistic
          [3,1] = linear
        }
        FIR_FB =
        {
          [1,1] = 0
          [2,1] =
          [3,1] = 0
        }
```

```
        fileNames =
        {
          [1,1] = XOR(2/1)
        }

    layersTotal =  3

    Neurons =
    {
      [1,1] =
         0.00000   1.00000
         0.00000   0.86603
         0.00000   0.86603

      [1,2] =
         0.0000e+000  1.0000e+000
        -1.0488e+001  2.7874e-005
        -2.9520e+000  4.9641e-002

      [1,3] =
         0.00000   1.00000
        -0.86411  -0.86411
    }

    Weights =
    {
      [1,1] = 0

      [1,2] =
        -4.29534  -4.47235  -2.67809
         0.68609  -2.18074  -2.02020

      [1,3] =
        -1.0119  -3.2117   2.9780
    }

    Deltas =
    {
      [1,1] =
         0
         0

      [1,2] =
        0.0000e+000
        3.6159e-007
        -5.6786e-004

      [1,3] =
         0.0000000
        -0.0040392
    }

    Epochs =  609

    PredictionError =

       0.42568
       0.45987
       0.44426
       0.43204
       ...
       ...
       2.5904e-006
       1.5369e-006
       9.1166e-007

debug>
```

En la variable `Neurons`, repetida a continuación, se puede ver que es una estructura con tres capas y tres neuronas en las primeras dos capas y dos neuronas en la última capa.

```
Neurons =
{
  [1,1] =
     1.00000   1.00000
     0.00000   0.86603
     0.00000   0.86603

  [1,2] =
     1.0000e+000  1.0000e+000
    -1.0488e+001  2.7874e-005
    -2.9520e+000  4.9641e-002

  [1,3] =
     1.00000   1.00000
    -0.86411  -0.86411
}
```

En cada capa hay dos valores, la primera columna es la que corresponde a las entradas de cada neurona y la segunda a su salida. Se puede observar que en la primera capa sólo están llenos los valores de salida de las neuronas, puesto que son los valores "alimentados" a la red.

En cada capa también, la primera neurona es la llamada Bias, y su valor de salida es siempre `1.0`.

Analizando cualquiera de las dos neuronas de la segunda capa es posible encontrar su valor de entrada como la sumatoria de los valores de salida de las tres neuronas de la capa anterior por los pesos que corresponden a esa capa:

Por ejemplo, Capa 2, Neurona 2 (primera neurona luego de Bias):

u_{22} = `-10.488 = (1.00000 * -4.29534) + (0.86603 * -4.47235) + (0.86603 * -2.67809)`

En donde: $u_{22} = (b_{11} * w_{21}) + (x_{12} * w_{22}) + (x_{13} * w_{23})$

Luego, la salida de esa misma neurona es y_{22}, fruto de aplicar la función logística a la entrada:

$$y_{22} = \frac{1}{1 + e^{-(-10.488)}} = 0.00002787$$

La salida final de esta red (en este instante) es el segundo valor, segunda capa, de la última neurona:

y_{32} = `-0.86411`

Los valores utilizados para entrenar y probar la red fueron normalizados, por lo que, para este instante, la red muestra que:

...las entradas $x_{12} = x_{13} =$ `0.86603` corresponden a `+1.0`, en tanto que...
...la salida y_{32} = `-0.86411` corresponde aproximadamente a `-1.0`.

Ejercicio 3:

Programar un identificador de sistema de una ecuación diferencial de 1er o 2do orden (FIR).

Respuesta:

Se propone en este ejercicio el entrenamiento de una red que aprenda la siguiente función de cuatro coeficientes:

$$y(n) = \frac{1}{4}x(n) + \frac{1}{4}x(n-1) + \frac{1}{4}x(n-2) + \frac{1}{4}x(n-3)$$

```
a = [1];
b = [1/4 1/4 1/4 1/4];
Y = filter(b, a, X);
```

En donde X es un vector que contiene la señal de ruido a ser filtrado.

Para entrenar esta red también se generaron dos conjuntos de datos: entrenamiento y prueba. En ambos casos son conjuntos de valores aleatorios equivalentes.

```
% ~~~~~~  Definicion de estructura de NN  ~~~~~~
% ~~~~~~ Red Neuronal -- Curso INAUT 2014 ~~~~~~
%
0, 10, none, 9, fir(1/1)
1, 1, tanh, 0,
2, 1, linear, 0,
```

Se observa que la red tiene tres capas, la primera con diez neuronas de entrada y sin función de activación (es la capa de entrada), la segunda capa es oculta, con una neurona con función de activación tangente hiperbólica y la tercera es la capa de salida, con una neurona y con función de activación lineal.

En la capa de entrada, en la columna 4 se puede ver que se agregan nueve retardos a la entrada. La definición de la función, en la columna 5 indica que los pares de entrada y salida constan de un dato de entrada y uno de salida cada par respectivamente, representando la señal de ruido entrante y la señal filtrada a la salida. El total de neuronas en la capa de entrada incluye el valor presente de la señal de entrada y nueve retardos adicionales, dando como resultado el número de neuronas de la capa de entrada, columna 2 de esa capa.

Conjunto de entrenamiento:

En la *Figura c1* se observan cien puntos de entrenamiento, constituidos por pares de entrada y salida. La entrada está graficada en línea continua de color verde claro y la salida del filtro a emular está graficada con círculos en sus puntos, en color rojo.

En la siguiente gráfica, *Figura c2*, la línea continua verde claro es la salida real del filtro que la red debe “aprender”. A su vez, la línea roja con círculos es el resultado entregado por la red. Obsérvese la “demora” en el llenado de los retardos al comienzo y a la salida de la gráfica.

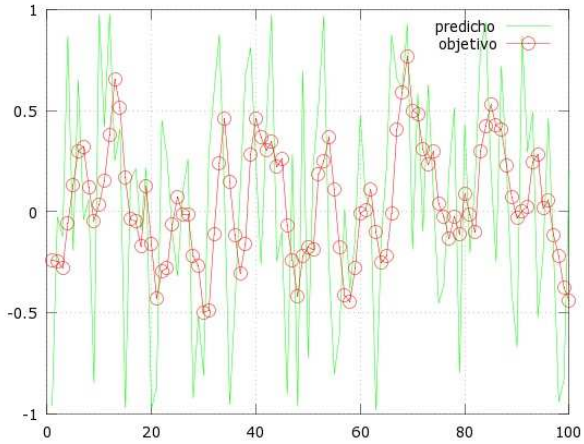


Figura c1

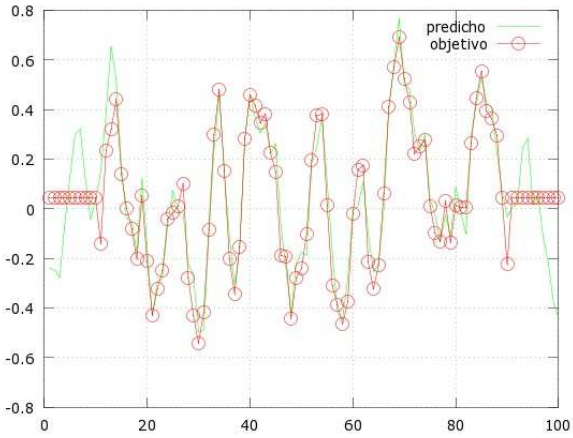


Figura c2

La curva de aprendizaje de la red se muestra a continuación, con un error de aprendizaje de 0.00504 en 400 iteraciones:

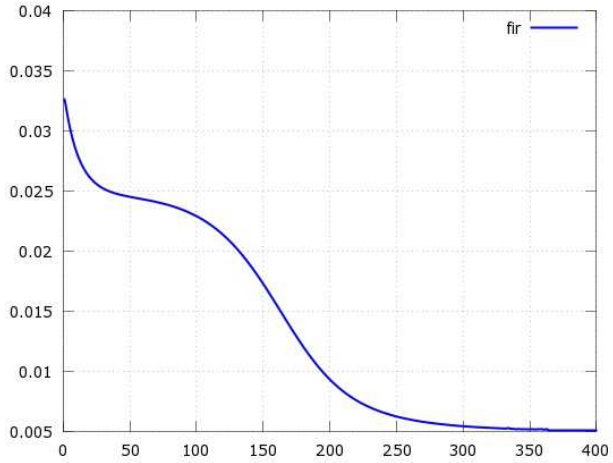


Figura c3

Se presentó un segundo conjunto de datos a modo de prueba a la red, con el siguiente resultado:

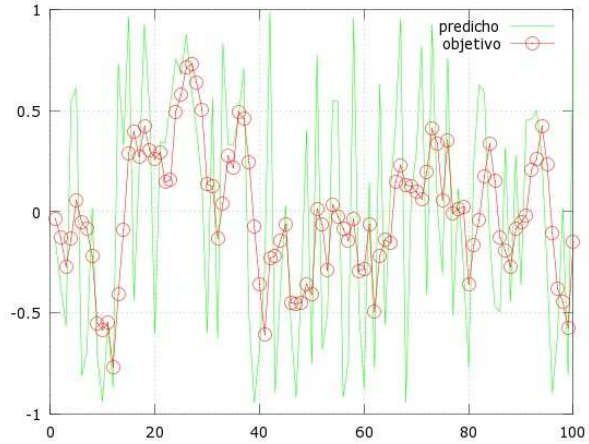


Figura c4

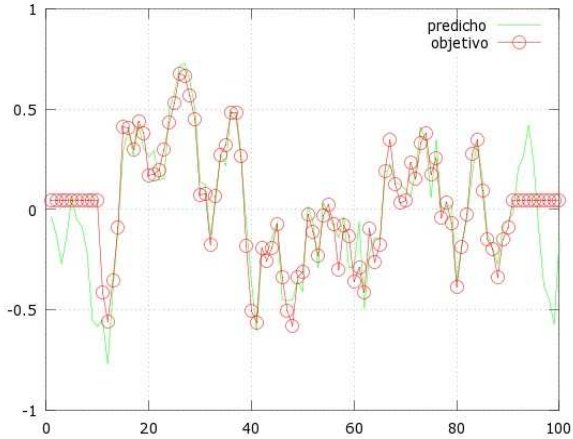


Figura c5

En la *Figura c4* se observan cien nuevos puntos de entrenamiento. La entrada está graficada en línea continua de color verde claro y la salida del filtro a emular está graficada en rojo, con círculos en sus puntos. Esta es la curva que replica la red ya entrenada (*Figura c5*), ante la misma entrada.

En la *Figura c4*, "predicho" equivale a la entrada `X` sin filtrar a la función `Y = filter(b, a, X)`. En la misma figura "objetivo" es la salida del filtro, esto es, `Y`.

En la *Figura c5*, "predicho" es la salida ideal del filtro que está graficada en rojo en la figura anterior. En tanto que "objetivo" es el resultado de haber aplicado la misma entrada de "predicho" de la *Figura c4*, es decir `X`, a la red entrenada. Se puede observar en esta figura cómo el resultado de la red sigue de cerca el valor que se espera de ella.

Queda así demostrado que la red aprendió la función diferencial con la que fue entrenada.

Ejercicio 4:

Programar la aproximación de la función sen(u), con u = [0, 2*pi].

Respuesta:

Se propone en este ejercicio el entrenamiento de una red que aprenda la siguiente función:

$$y(n) = \text{sen}\big(x(n)\big)\ , para\ x(n) \in [0, 2\pi]$$

Se generaron así dos conjuntos de datos, ambos con 100 pares (`genNumPairs=100`) y uno de ellos con un factor de aleatoriedad que permitiese probar la red ante entradas que varían aleatoriamente dentro del mismo segmento, es decir, entradas para las cuales la red no fue entrenada.

En el segundo conjunto de datos el vector `Y2` no tiene diferencia en su uso práctico, ya que la salida de la red es siempre comparada contra una función seno perfecta.

```
rMin = 0;
rMax = 2*pi;

if (genExactOrRandom)
       X1 = [rMin:(rMax/(genNumPairs-1)):rMax]';
       Y1 = sin(X1);
else
       X2 = [rMin:(rMax/((genNumPairs/2)-1))*(1+randn):rMax]';
       Y2 = sin(X2);
end
```

La definición de la red es la siguiente:

```
% ~~~~~~  Definicion de estructura de NN  ~~~~~~
% ~~~~~~ Red Neuronal -- Curso INAUT 2014 ~~~~~~
%
0, 1, none, 0, SENO(1/1)
1, 20, tanh, ,
2, 10, tanh, ,
3, 1, tanh, 0,
```

Se observa que la red tiene cuatro capas, la primera con una neurona y sin función de activación (es la capa de entrada), la segunda y la tercera capas son ocultas, con 20 y 10 neuronas respectivamente y con función de activación tangente hiperbólica, en tanto que la capa de salida posee una neurona y también función de activación de tangente hiperbólica.

Se presenta a continuación el resultado del entrenamiento de la red con el conjunto de pares de entrenamiento para 50 y 100 iteraciones en las *Figuras d1* y *d2* respectivamente:

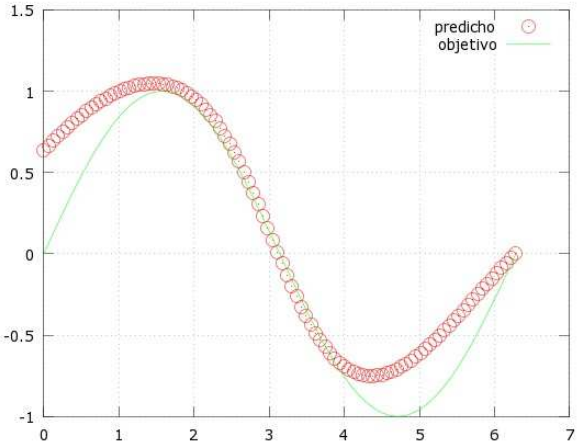


Figura d1

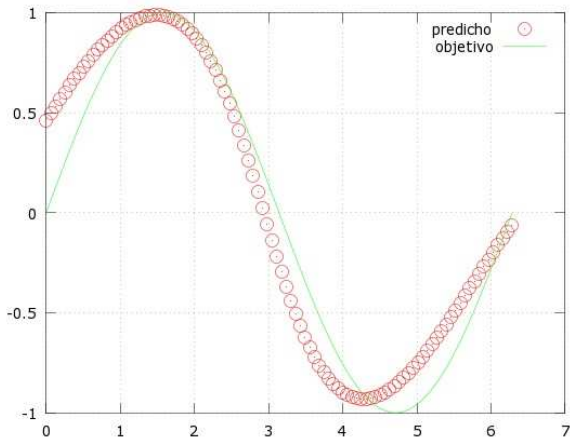


Figura d2

En las siguientes gráficas se mostrará la evolución del error de entrenamiento para dos exigencias: Luego de 134 iteraciones el error de predicción cae por debajo de 0.01; se entrenó a la red con 100 iteraciones más y luego de 234 iteraciones el error de predicción cae por debajo de 0.001:

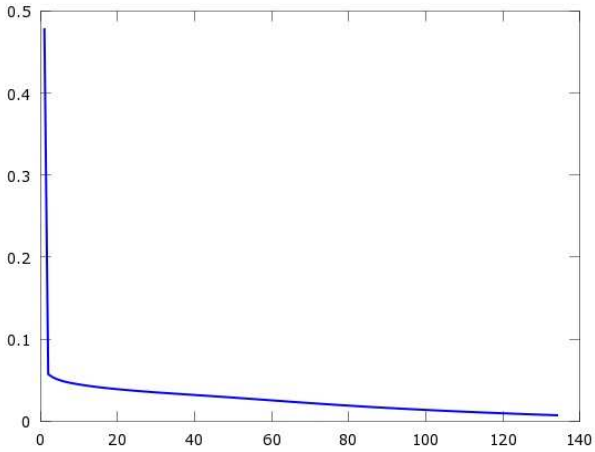


Figura d3

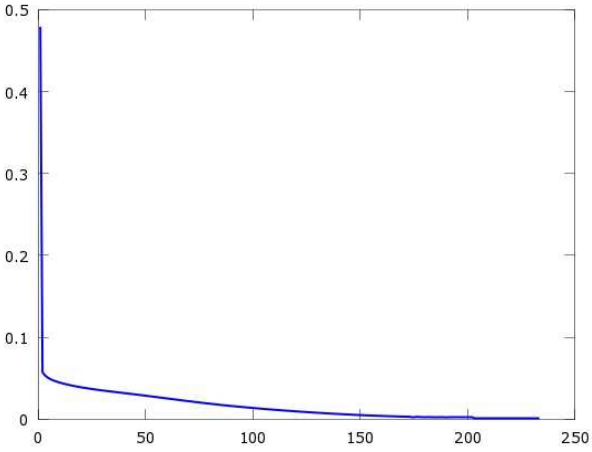


Figura d4

Resultados de entrenamiento y prueba para 134 iteraciones:

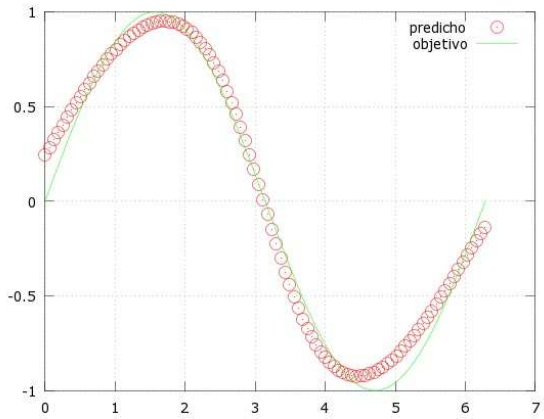


Figura d5

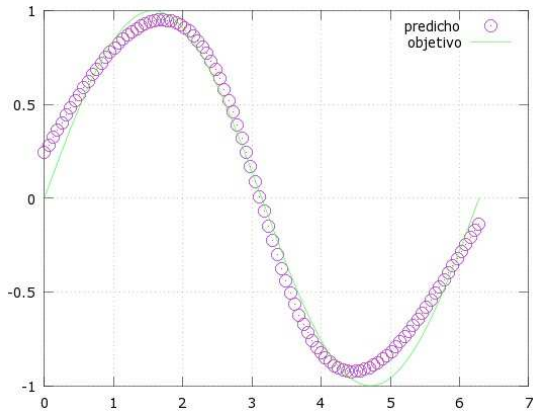


Figura d6

Resultados de entrenamiento y prueba para 234 iteraciones:

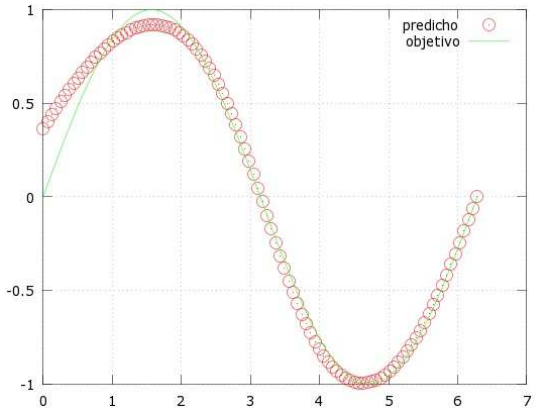


Figura d7

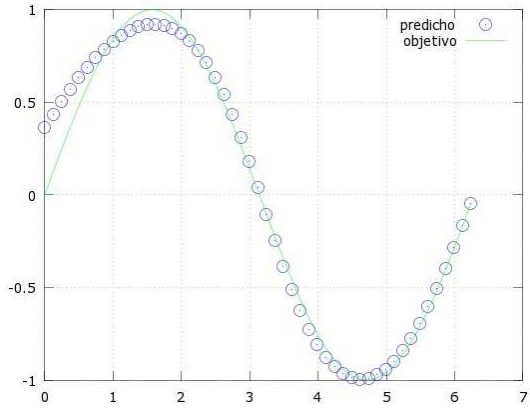


Figura d8

Ejercicio 5:

Programar una red para identificación de un sistema no lineal.

Respuesta:

El sistema a identificar es un sistema combinado de Motor de Corriente Continua con Carga Variable. De esta manera existen dos entradas independientes y dos salidas independientes entre sí, y vinculadas simultáneamente mediante la evolución dinámica del mismo sistema cerrado.

Se realizó el diseño en Simulink de Matlab de este sistema a modo de caja cerrada y se lo alimentó con distintas señales de entrada.

El sistema es la combinación simultánea de las siguientes ecuaciones:

$$J\frac{d^2\theta}{dt^2} = T - b\frac{d\theta}{dt} \Longrightarrow \frac{d^2\theta}{dt^2} = \frac{1}{J}(K_t i - b\frac{d\theta}{dt})$$

$$L\frac{di}{dt} = -Ri + V - e \Longrightarrow \frac{di}{dt} = \frac{1}{L}(-Ri + V - K_e\frac{d\theta}{dt})$$

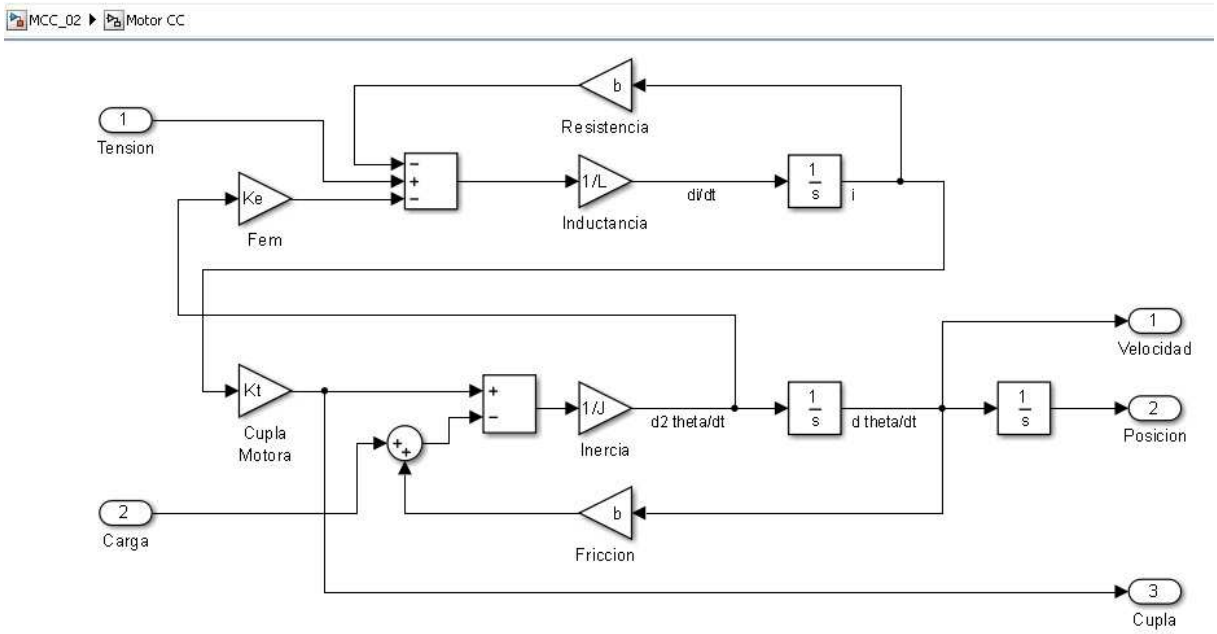


Figura e1

En la *Figura e1* se observa el sistema del motor de corriente continua en conjunto con una alimentación de carga y la realimentación de una carga de fricción común.

Las constantes del motor utilizadas en la simulación son las siguientes:

```
J = 0.02;          % kg.m^2/s          - momento de inercia del rotor
b = 0.2;           % N.m.s             - constante de friccion viscosa
Ke = 0.015;        % V/rad/s           - constante Fem
Kt = 0.015;        % N.m/A             - constante de Cupla Motora
R = 2;             % Ohm               - resistencia electrica
L = 0.5;           % H                 - inductancia electrica
```

En la *Figura e2* se muestra el esquema que se utilizó para excitar el sistema mcc (motor + carga) y almacenar los valores en el workspace de Matlab. Los vectores x e y están compuestos de dos valores cada par de muestras, es decir, cuatro valores por instante de muestreo.

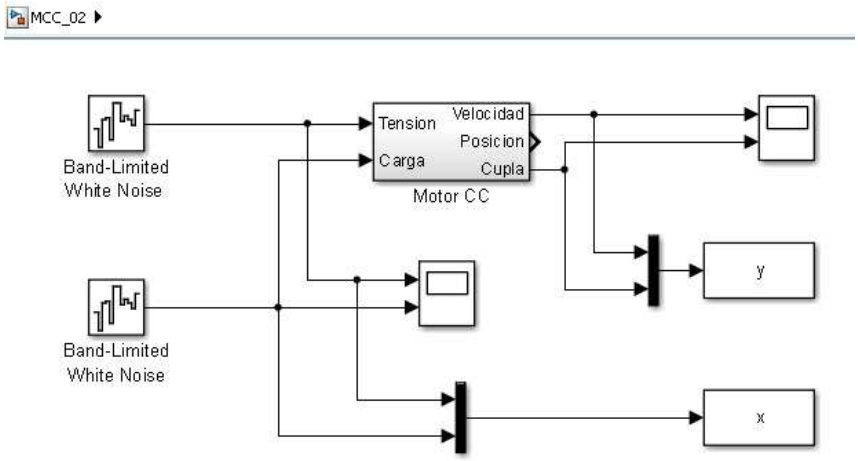


Figura e2

Se produjo señal de Ruido Blanco con distinta frecuencia para alimentar el sistema y se almacenaron 5000 muestras, de las cuales las primeras 3000 fueron utilizadas para entrenamiento y las restantes 2000 fueron aplicadas a la prueba de la red entrenada. Las muestras se tomaron con una frecuencia de 0.2 segundos, con lo cual la ventana de tiempo de entrenamiento es de 600 segundos y la de prueba es de 400 segundos.

La definición de la red es la siguiente:

```
% ~~~~~~  Definicion de estructura de NN  ~~~~~~
% ~~~~~~ Red Neuronal -- Curso INAUT 2014 ~~~~~~
%
%     (El comentario de encabezado es opcional)
%
% Columnas son:
%   1. Capa#, (0=entrada, #=oculta, max(#)=salida)
%   2. # de Neuronas en esa capa (sin Bias) = #input + FIR*#input + FB*#output
%   3. Función de Activación (linear, logistic, tanh)
%   4. #retardos de entrada FIR:
%         default=0, <= ((columna_2-#input)-FB*#output)/#input
%      #retardos de realimentación de salida FB FIR:
%         default=0, <= ((columna_2-#input)-FIR*#input)/#output
%   5. Nombre de Red(pares #in/#out) p.ej.: XOR(2/1), SIN(1/1), mcc(2/2), etc.
%
% --------------{fin de encabezado}--------------
0, 14, none, 3, mcc(2/2)
1, 10, logistic, ,
2, 5, logistic, ,
3, 2, linear, 3,
```

La red tiene cuatro capas, la primera con 14 neuronas y sin función de activación (es la capa de entrada), la segunda y la tercera capas son ocultas, con 10 y 5 neuronas respectivamente y con función de activación logística, en tanto que la capa de salida posee dos neuronas y función de activación lineal.

La primera capa tiene 14 neuronas porque necesita 2 neuronas para recibir el par de datos de entrada, más 2 * 3 = 6 neuronas por los tres retardos en la entrada más 2 * 3 = 6 neuronas por los tres retardos de la señal de salida realimentada.

En la *Figura e3* se presentan las señales de entrada de Tensión (señal de forma de ruido escalón, color azul) y de salida de Velocidad (color rojo):

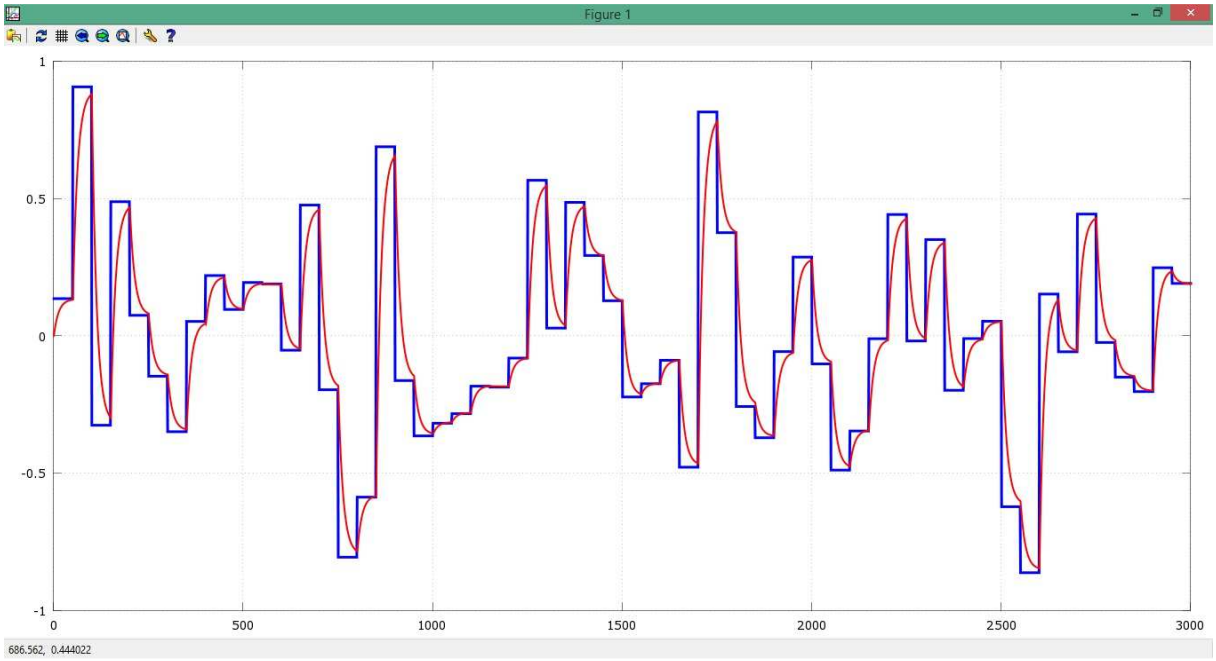


Figura e3

En la *Figura e4* se presentan las señales de entrada de Carga (señal de forma de ruido escalón, color verde claro) y de salida de Cupla (color magenta):

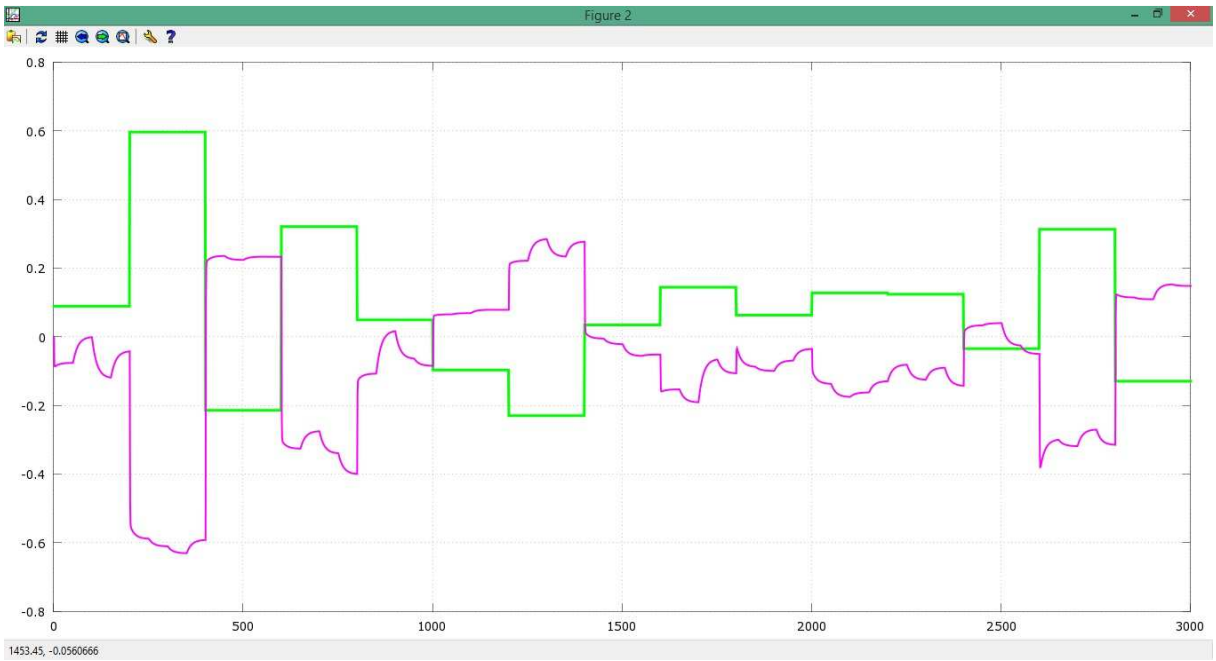


Figura e4

Se incluirán a continuación los resultados de entrenamiento para 120, 140 y 160 iteraciones.

Error de entrenamiento para 120 iteraciones:

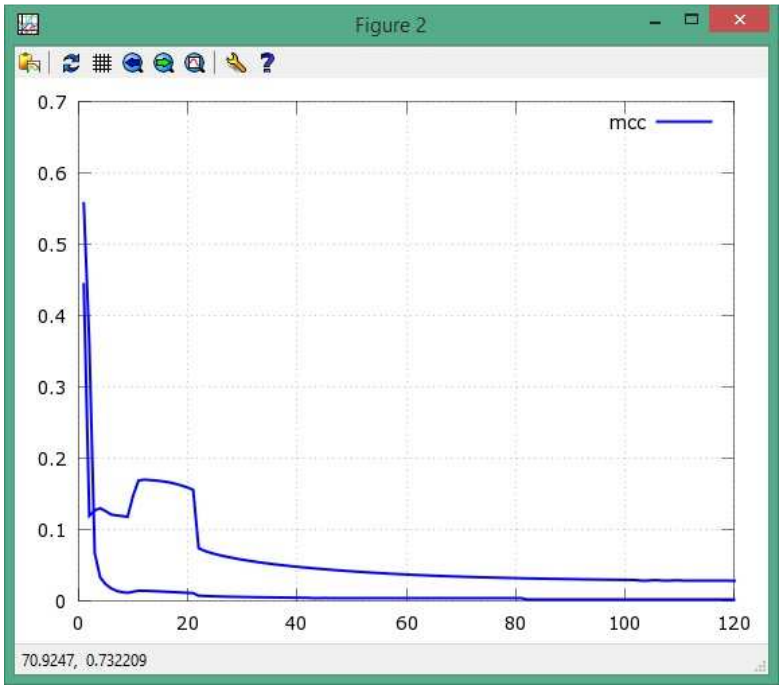


Figura e5

A continuación, para 120 iteraciones, resultados de entrenamiento y prueba, salida de Velocidad:

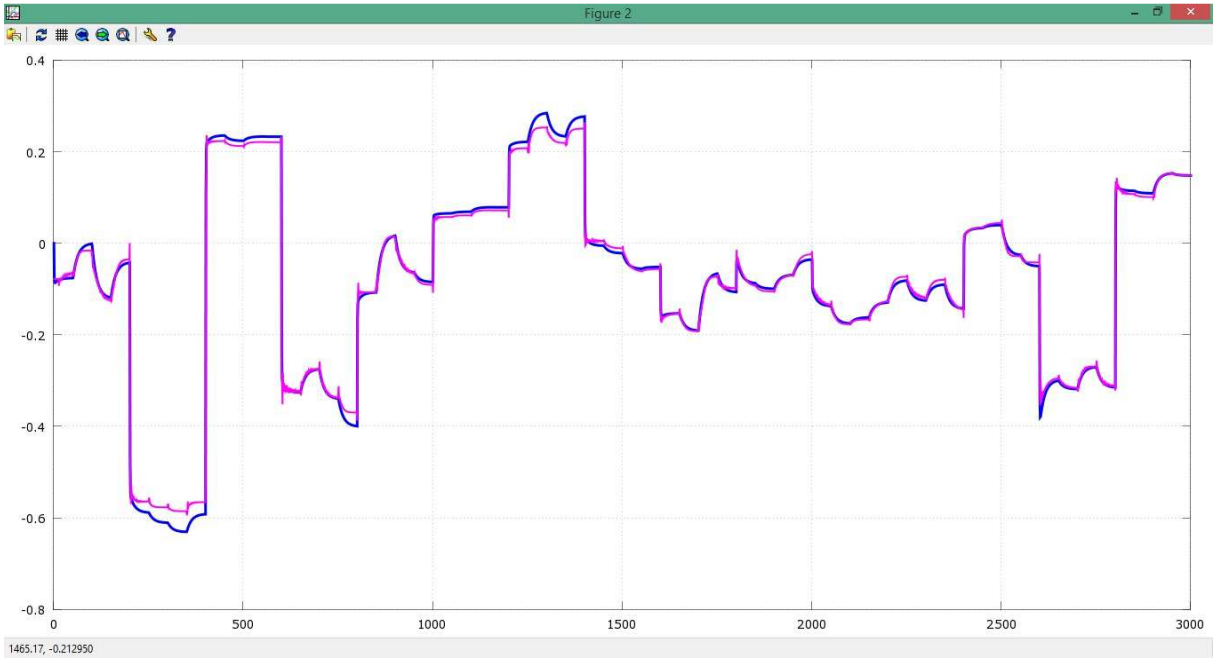


Figura e6

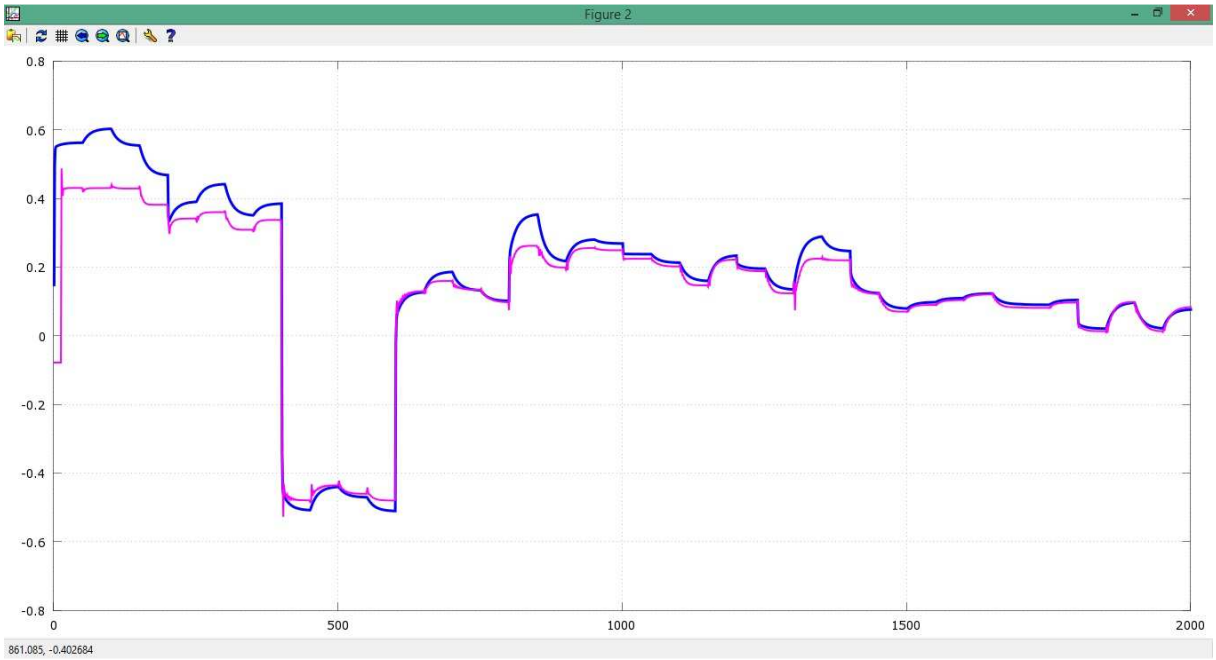


Figura e7

A continuación, para 120 iteraciones, resultados de entrenamiento y prueba, salida de Cupla:

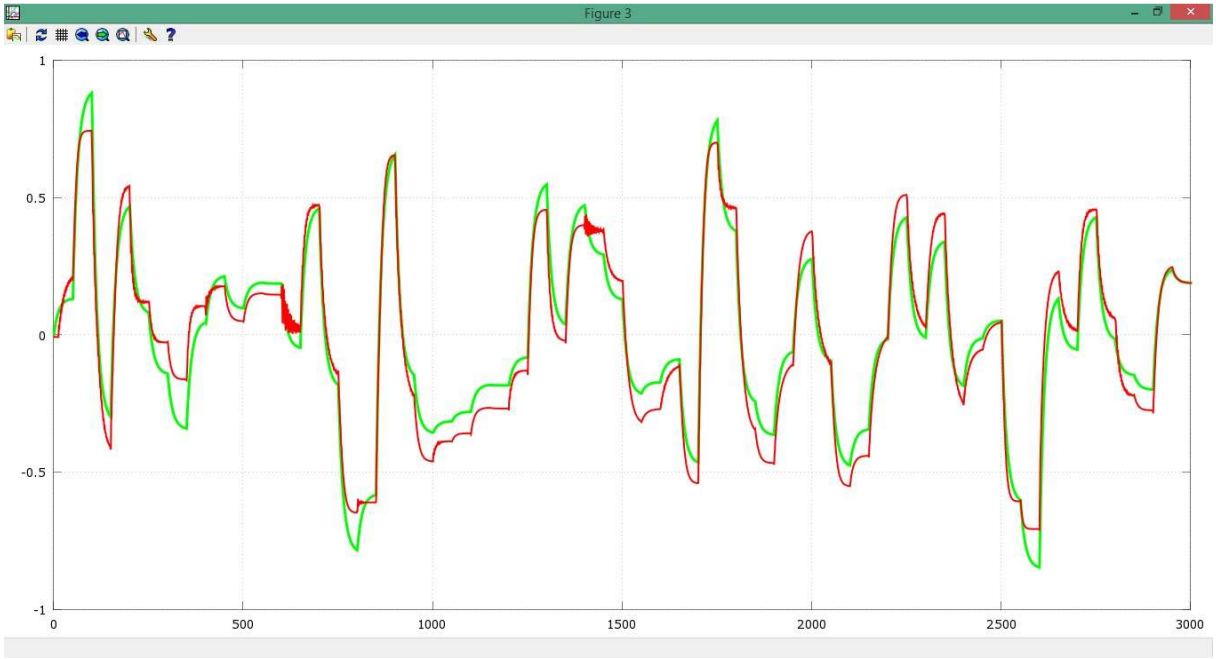


Figura e8

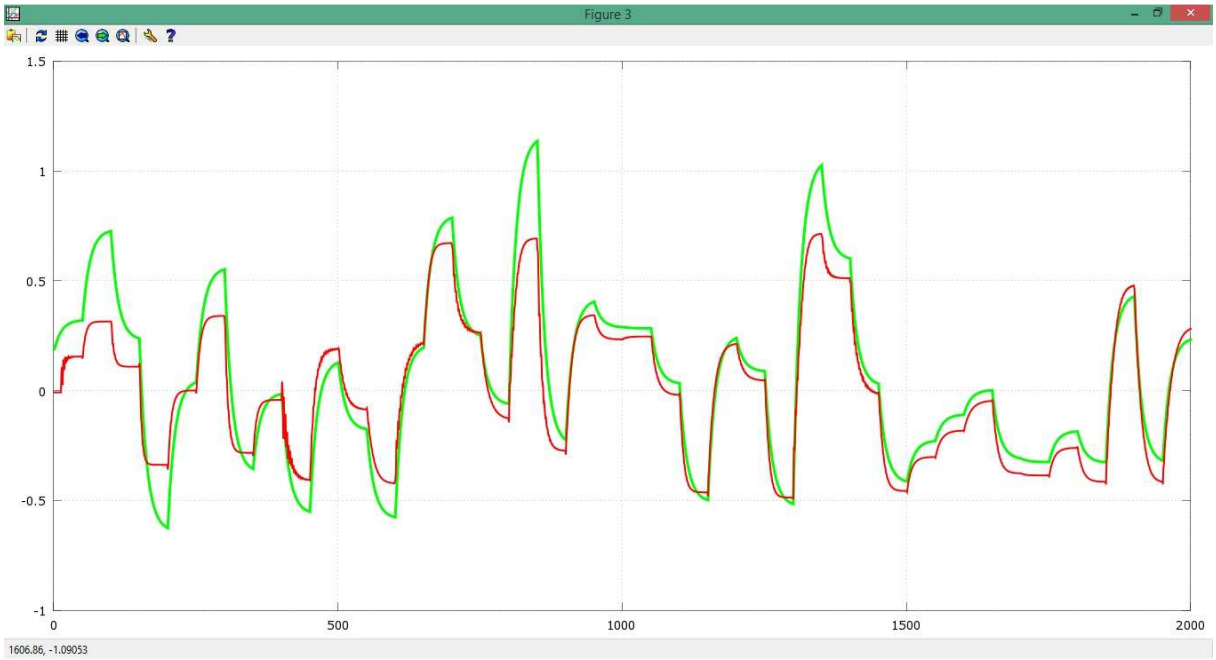


Figura e9

Error de entrenamiento para 140 iteraciones:

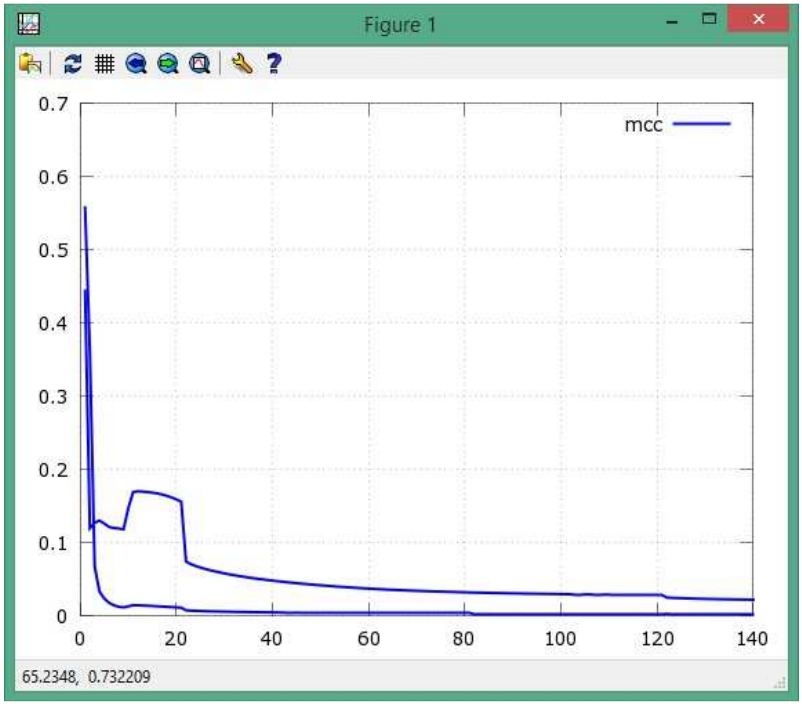


Figura e10

A continuación, para 140 iteraciones, resultados de entrenamiento y prueba, salida de Velocidad:

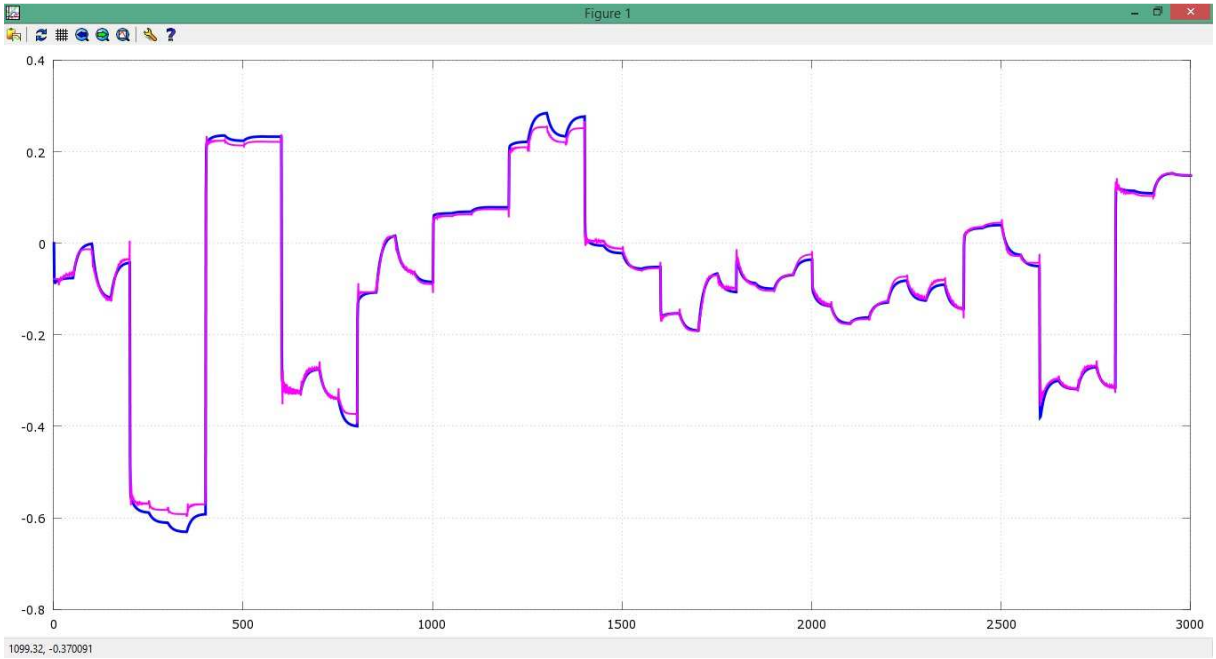


Figura e11

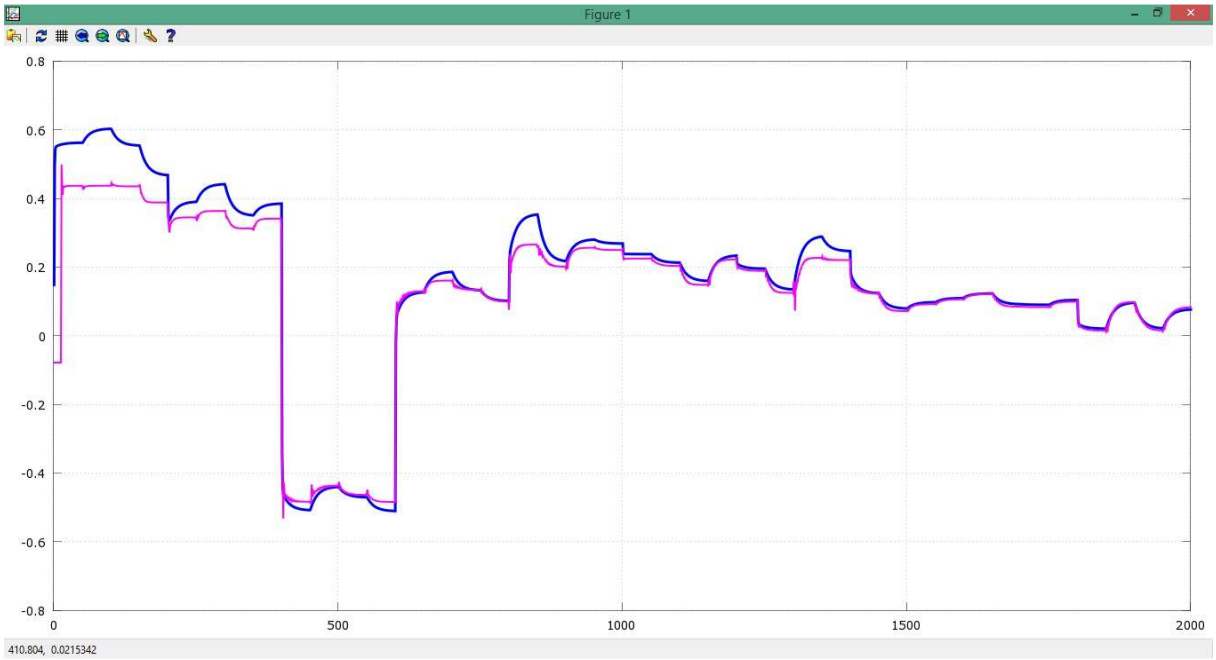


Figura e12

A continuación, para 140 iteraciones, resultados de entrenamiento y prueba, salida de Cupla:

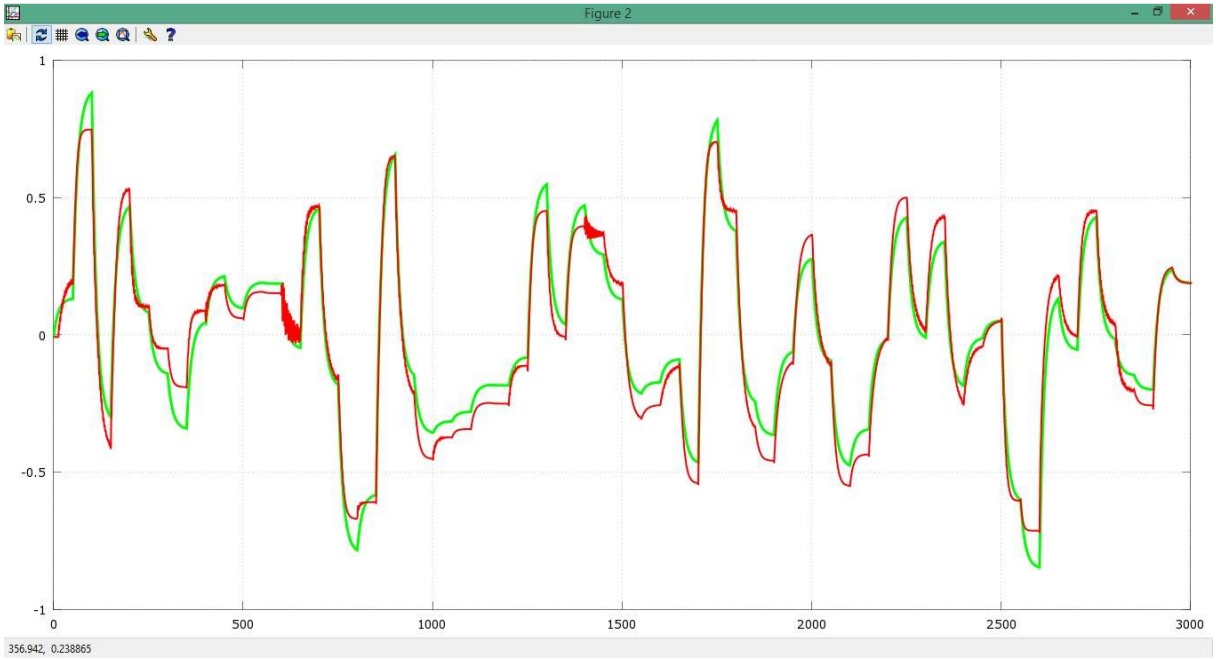


Figura e13

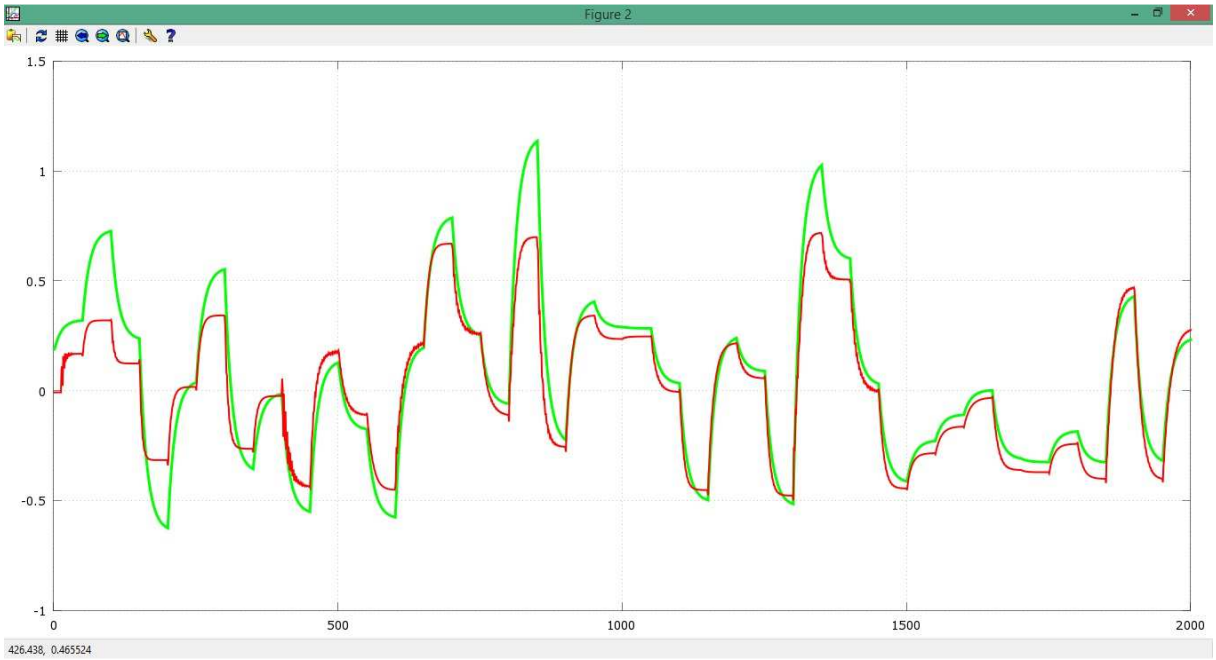


Figura e14

Error de entrenamiento para 160 iteraciones:

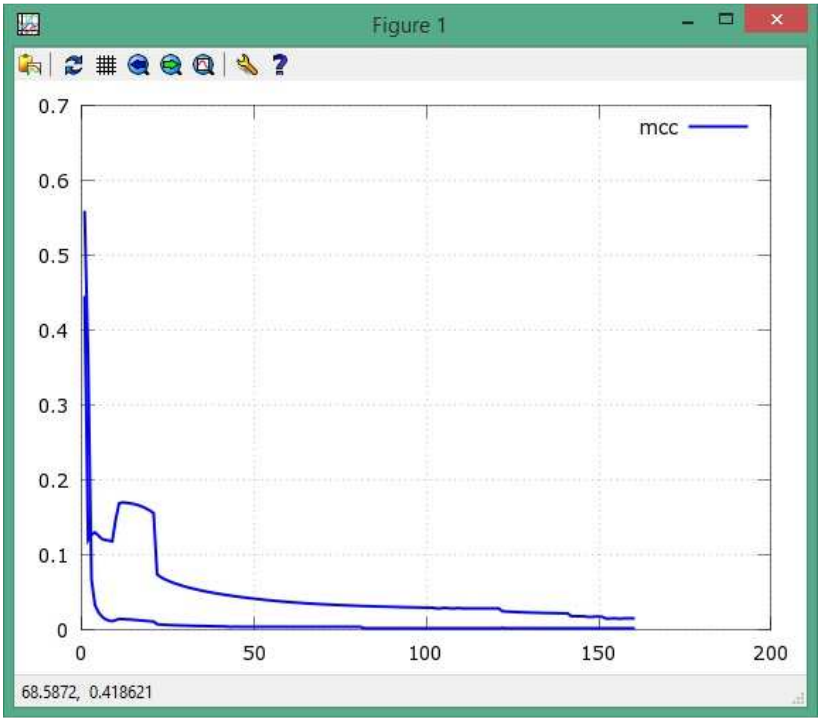


Figura e15

A continuación, para 160 iteraciones, resultados de entrenamiento y prueba, salida de Velocidad:

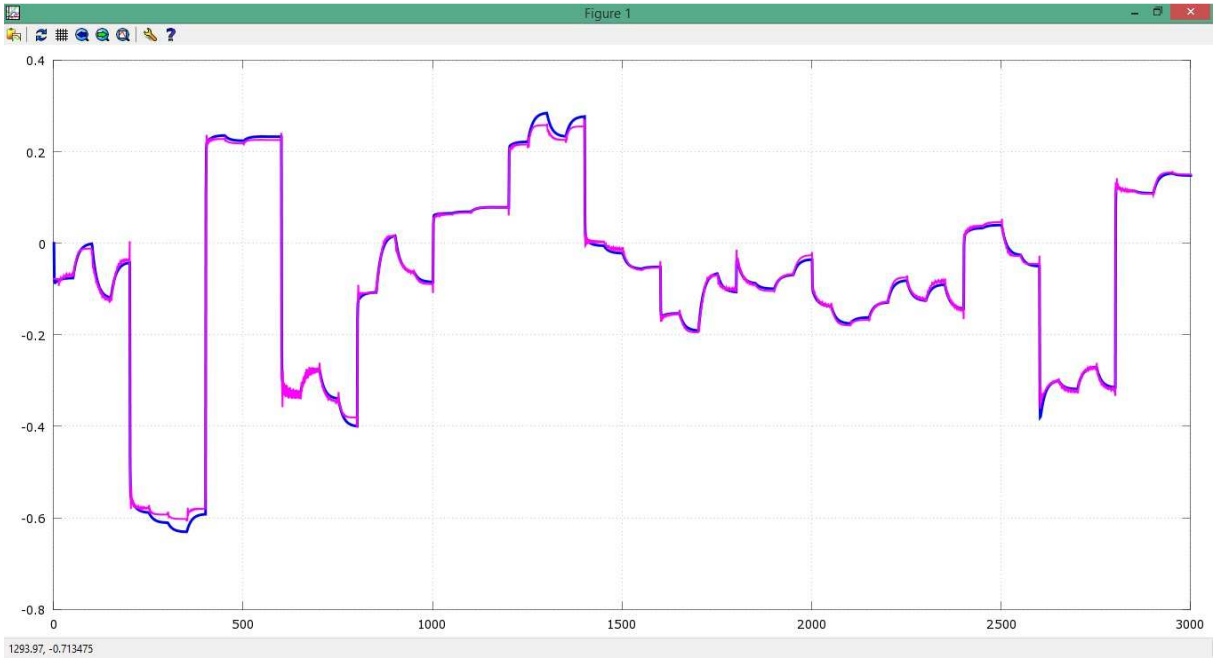


Figura e16

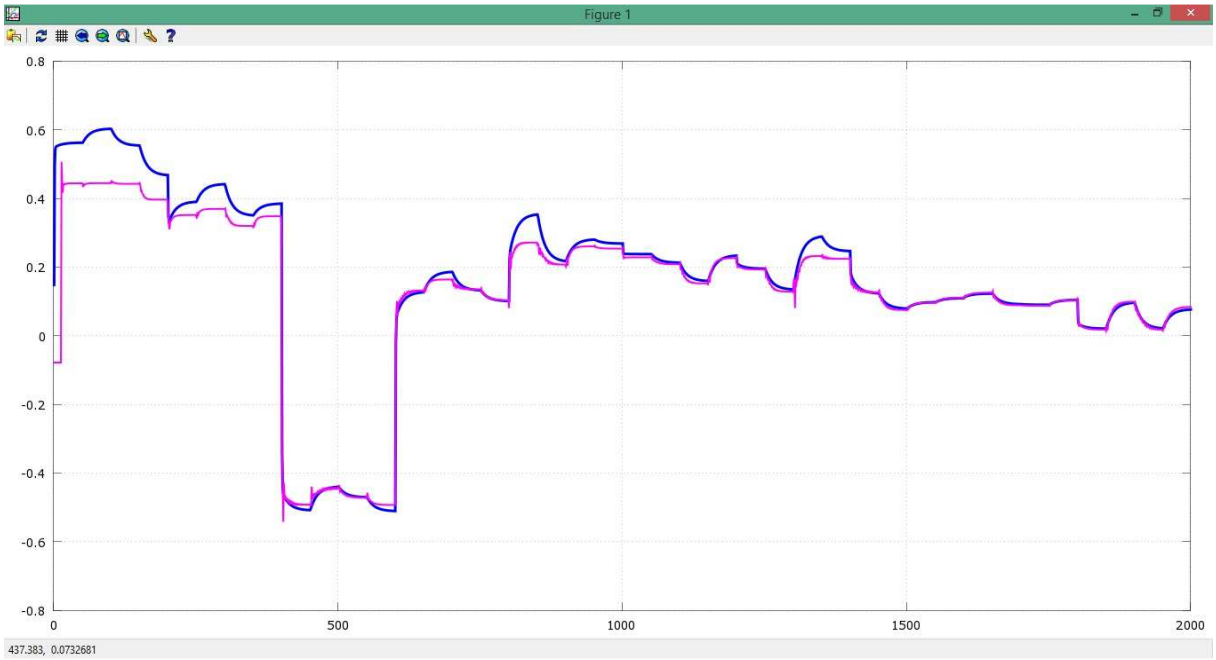


Figura e17

A continuación, para 160 iteraciones, resultados de entrenamiento y prueba, salida de Cupla:

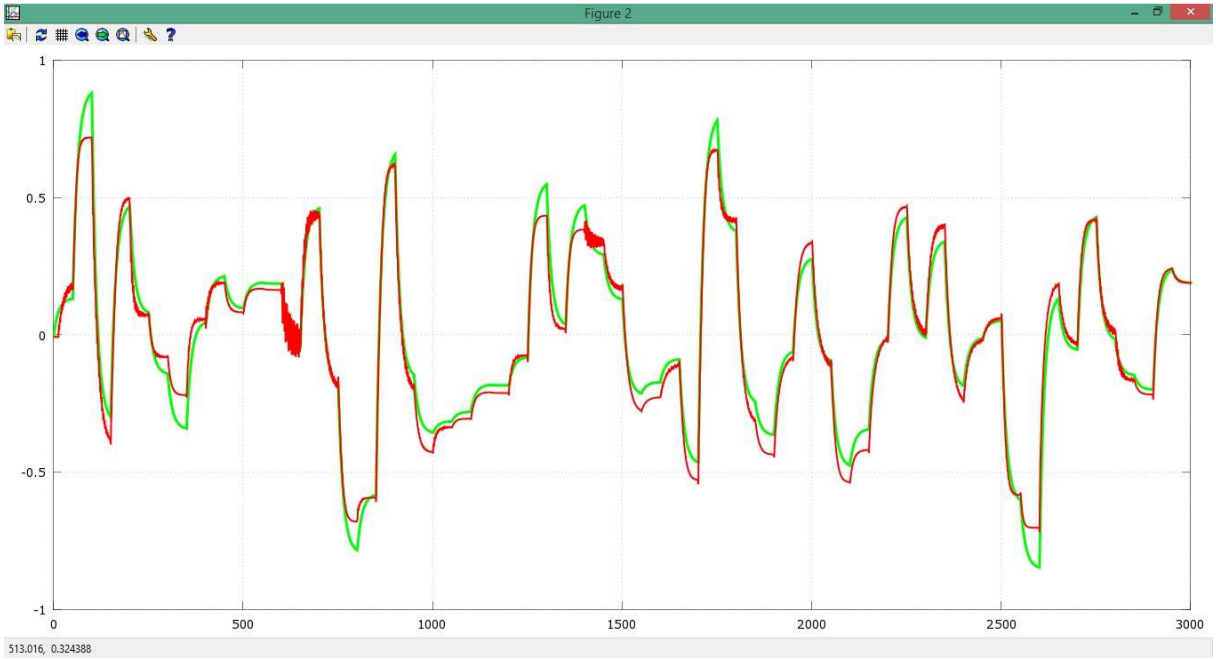


Figura e18

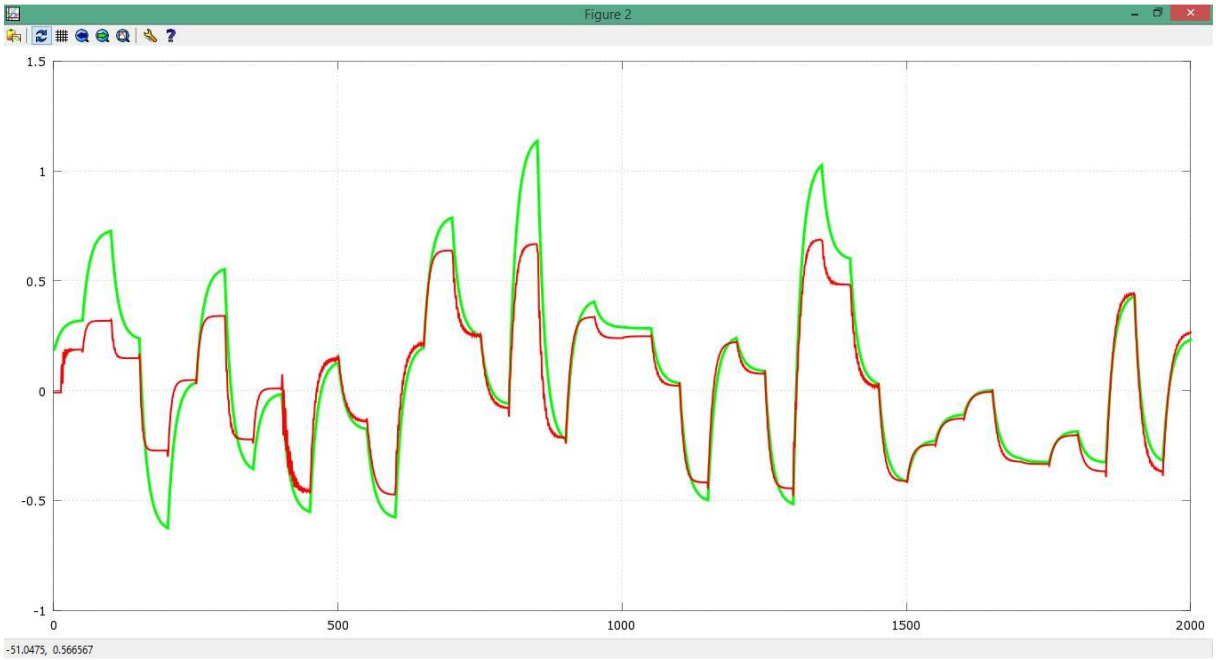


Figura e19

Finalmente, a modo de prueba adicional se somete al sistema en Simulink a una entrada escalón en la puerta de Tensión a los 10 segundos de iniciada la simulación, y a los 40 segundos, con la salida de Velocidad estabilizada, se introduce una función seno de baja amplitud como variable en la entrada de Carga.

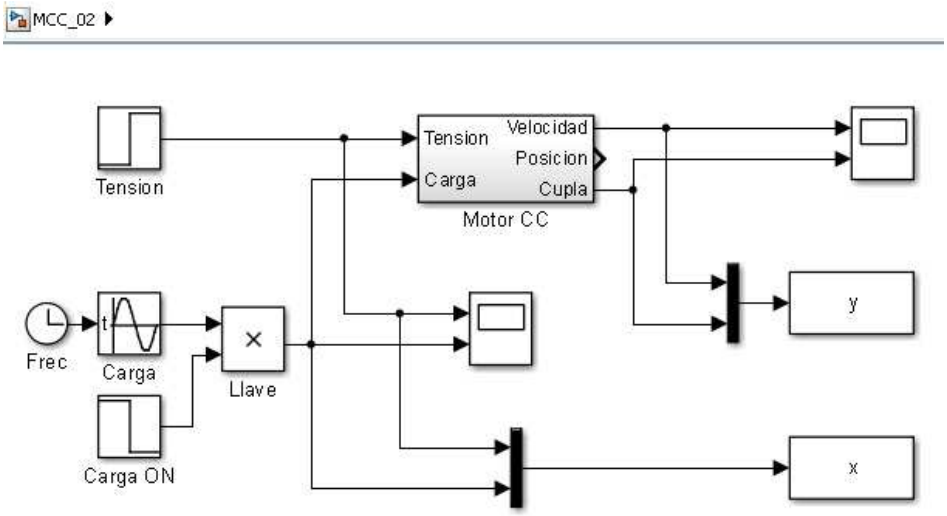


Figura e20

Los resultados de esta simulación se superponen con los obtenidos luego de alimentar la red neuronal entrenada en las siguientes gráficas.

Salida de Velocidad, en azul el resultado del modelo Simulink, en magenta el de la red neuronal:

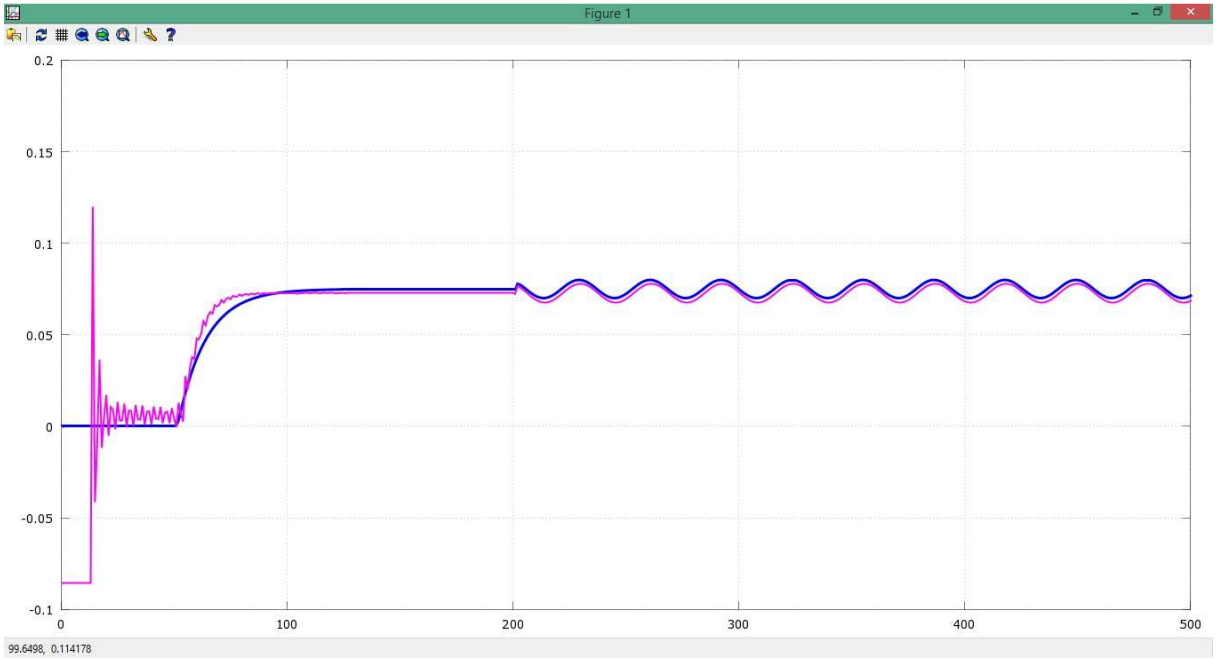


Figura e21

Salida de Cupla, en verde el resultado del modelo Simulink, en rojo el de la red neuronal:

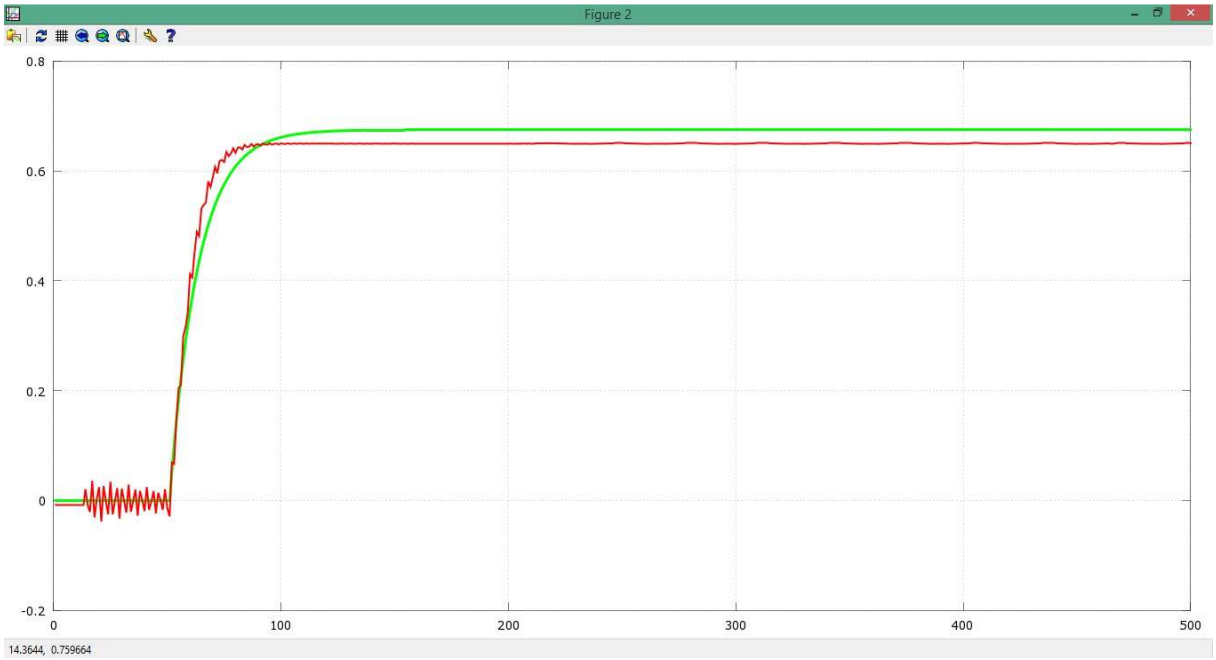


Figura e22

Parte 3: Apéndice con explicación del código nn22 utilizado en la Parte 2

Para el cumplimiento de la Parte 2 del práctico se desarrolló un conjunto de funciones realizados con la aplicación **Octave**, software de licencia libre GNU y de código abierto (FOS), cuyo comportamiento es equivalente a Matlab, exceptuando el uso de los Toolbox que son propietarios de la empresa Mathworks, que no están disponibles para Octave.

En http://es.wikipedia.org/wiki/GNU_Octave es posible encontrar algunas de las ventajas del uso de esta aplicación.

Framework nn22

Se desarrolló un grupo de funciones que, utilizadas en conjunto, permiten obtener la solución de las variantes más comunes de redes neuronales con aprendizaje backpropagation.

La herramienta es de uso genérico y permite la construcción de redes neuronales de estructura Multicapa Feedforward genérica, con retardos de entrada y con retardos también en la realimentación de salida, habilitando así su comportamiento dinámico.

Archivos principales

Los archivos principales se mantienen inalterables para todos los modelos de red. Son los archivos que comienzan con el prefijo "nn_" y se listan a continuación:

```
nn22.m
nn_intializeNN.m
nn_loadNN.m
nn_loadIOdata.m
nn_trainNN.m
nn_feedforwardNN.m
nn_plotXYdata.m
nn_createFileConstants.m
```

Cada uno de los archivos listados en los ítems precedentes tiene un uso especial

Cada uno de los archivos listados en los ítems precedentes tiene un uso especial y su descripción se realiza a continuación:

nn22.m

Este es el script principal cuya función consiste en presentar un menú al usuario que le permite:

a) Inicializar una red
b) Cargar una red ya entrenada
c) Inicializar o cargar datos de entrenamiento
d) Entrenar una red
e) Probar una red, con datos de entrenamiento o con datos de prueba
f) Graficar los resultados

g) Salir temporalmente al workspace para tareas de debug

Para trabajar con una red cualquiera lo primero que se debe hacer es ingresar el nombre base de la red, que pude ser, en los ejercicios incluidos, uno de los siguientes:

- and
- xor
- fir
- seno
- mcc

Si se desea trabajar con otra red, debe elegir la opción de salir del script principal y ejecutarlo nuevamente, habilitando la carga de los archivos principales de esa nueva red.

nn_trainNN.m

Esta función es la encargada de entrenar la red, ya sea desde cero, o continuar un entrenamiento con la presentación de diferentes iteraciones o presentaciones de épocas de entrenamiento. Permite incluso la modificación de las constantes de entrenamiento entre batchs (lotes de entrenamiento).

La función de entrenamiento llama a la función Feedforward pasando como parámetro únicamente el índice del par de entrenamiento que corresponda. Esto permite acelerar el tiempo de entrenamiento dado que ni Matlab ni Octave permiten el uso de punteros y todos los parámetros se pasan por valor.

nn_feedforwardNN.m

Esta función es la encargada de alimentar la red con un vector de entradas y obtener un vector de salidas. El vector a ser alimentado se recibe como puntero a una matriz de datos global, lo que permite que su uso sea el más eficiente posible en cuanto al tiempo en el traspaso de valores.

Esta función es la encargada de administrar la carga de los retardos de entrada y las realimentaciones de salida, que son definidas en la estructura de cada red.

nn_intializeNN, nn_loadNN, nn_loadIOdata, etc.

Luego del script nn22.m, son todas funciones.

La función **nn_intializeNN.m** es la encargada de crear desde cero los datos propios de cada red, sin entrenar.

La función **nn_loadNN.m** es la encargada de leer y cargar en el workspace los datos propios de cada red, en estado de entrenamiento en que se encuentre. Su único parámetro nnFilesBase es la base de los nombres de archivo de cada red (p.ej.: and, xor, seno) que será agregado al comienzo del archivo que contiene los datos de la red y con extensión .mat: es decir, compone el nombre con:

```
nFilesBase + '_nnData.mat'.
```

La función **nn_loadIOdata.m** puede cargar y también generar datos de entrada (pares IO de entrada/salida) para cada red, utilizados para entrenamiento y prueba.

La función **nn_plotXYdata.m** permite graficar de forma genérica cualquier par de datos (en forma vectorial). Es útil cuando se desea obtener pares de gráficas de entrada/salida o salida/salida. De todos modos, cada red en particular tiene su propia función de graficación, dado que la forma de mostrar los datos es muy específica respecto del tipo de datos de la red en cuestión.

La función **nn_createFileConstants.m** permite crear de cero un archivo de constantes para una red, en caso de que el archivo no se encuentre disponible. Es una función genérica que sólo crea un archivo con datos predeterminados y que luego deberán ser editados apropiadamente para entrenar la red en cuestión.

Archivos dependientes de cada red neuronal

Los archivos que dependen de cada red neuronal comienzan con un prefijo correspondiente al Nombre Base de la red, por ejemplo, en el caso de la red utilizada para aprender la función XOR, todos los archivos comienzan con el prefijo "`xor_`", los utilizados para aprender la función SENO comienzan con "`SENO_`" y se listan a continuación para todas las redes correspondientes a los ejercicios de la Parte 2 de este informe:

```
and_Structure.ini
and_nnData.mat
and_Constantes.m
and_generarIOdata.m
and_graficarIOdata.m
and_ioIdealTest.txt
and_ioIdealTrain.txt
and_ioInputTest.txt
and_ioInputTrain.txt

xor_Structure.ini
xor_nnData.mat
xor_Constantes.m
xor_generarIOdata.m
xor_graficarIOdata.m
xor_ioIdealTest.txt
xor_ioIdealTrain.txt
xor_ioInputTest.txt
xor_ioInputTrain.txt

fir_Structure.ini
fir_nnData.mat
fir_Constantes.m
fir_generarIOdata.m
fir_graficarIOdata.m
fir_ioIdealTest.txt
fir_ioIdealTrain.txt
fir_ioInputTest.txt
fir_ioInputTrain.txt

seno_Structure.ini
seno_nnData.mat
seno_Constantes.m
seno_generarIOdata.m
seno_graficarIOdata.m
seno_ioIdealTest.txt
seno_ioIdealTrain.txt
seno_ioInputTest.txt
seno_ioInputTrain.txt

mcc_Structure.ini
mcc_nnData.mat
mcc_Constantes.m
mcc_generarIOdata.m
mcc_graficarIOdata.m
mcc_ioIdealTest.txt
mcc_ioIdealTrain.txt
mcc_ioInputTest.txt
mcc_ioInputTrain.txt
```

Cada uno de los archivos listados en los ítems precedentes tiene un uso especial

Cada uno de los archivos listados en los ítems precedentes tiene un uso especial y su descripción se realiza a continuación:

_Structure.ini

(El comienzo del archivo contiene el Nombre Base de la red)

La estructura de las redes está definida en un archivo con extensión .ini que contiene:

a) Cantidad de capas de neuronas, incluyendo entrada y salida.
b) Cantidad de neuronas por capa.
c) Función de activación utilizada por cada capa de neuronas.
d) Posibilidad de realimentación de salida (vector de salida).
e) Número de retardos de la señal de entrada (vector de entrada).
f) Número de retardos en la realimentación de señal de salida (vector de salida).

A modo de ejemplo se incluye el contenido del archivo correspondiente al entrenamiento de la red mcc, que fue utilizada para la identificación del sistema no lineal:

```
% ~~~~~~  Definicion de estructura de NN  ~~~~~~
% ~~~~~~ Red Neuronal -- Curso INAUT 2014 ~~~~~~
%
%      (El comentario de encabezado es opcional)
%
% Columnas son:
%   1. Capa#, (0=entrada, #=oculta, max(#)=salida)
%   2. # de Neuronas en esa capa (sin Bias) = #input + FIR*#input + FB*#output
%   3. Función de Activación (linear, logistic, tanh)
%   4. #retardos de entrada FIR:
%          default=0, <= ((columna_2-#input)-FB*#output)/#input
%      #retardos de realimentación de salida FB FIR:
%          default=0, <= ((columna_2-#input)-FIR*#input)/#output
%   5. Nombre de Red(pares #in/#out) p.ej.: XOR(2/1), SIN(1/1), mcc(2/2), etc.
%
% --------------{fin de encabezado}--------------
0, 14, none, 3, mcc(2/2)
1, 10, logistic, ,
2, 5, logistic, ,
3, 2, linear, 3,
```

Todas las líneas que comienzan con el signo % son comentarios y no son tenidas en cuenta por el programa al leer el contenido del archivo. En este caso, sólo se leen las cuatro líneas que contienen la definición de las cuatro capas de la red.

Se puede observar que la red mcc se define con cuatro capas, la primera con 14 neuronas y sin función de activación (es la capa de entrada), la segunda y la tercera capas son ocultas, con 10 y 5 neuronas respectivamente y con función de activación logística, en tanto que la capa de salida posee dos neuronas y función de activación lineal.

La primera capa tiene 14 neuronas porque necesita 2 neuronas para recibir el par de datos de entrada, más 2 * 3 = 6 neuronas por los tres retardos en la entrada más 2 * 3 = 6 neuronas por los tres retardos de la señal de salida realimentada.

_nnData.mat

(El comienzo del archivo contiene el Nombre Base de la red)

Este archivo contiene los datos de las variables que definen la red. El archivo se crea en la memoria al inicializar la red y se graba en disco únicamente luego de realizar al menos una sesión de entrenamiento. También se graba luego de cada sesión adicional de entrenamiento.

Una vez que se obtiene una red entrenada con las condiciones finales deseadas, es recomendable guardar una copia de este archivo para evitar su deterioro accidental en caso de que el usuario lo cree nuevamente o lo sobre entrene, o lo entrene con parámetros no deseados.

A modo de ejemplo se muestra el contenido de la red `fir_nnData.mat`.

Para obtener esta información, en la línea de comandos se ejecuta el script principal, `nn22` y se ingresa el nombre base de la red, `fir`:

```
>> nn22

NN Redes Neuronales (INAUT 2014)
================================

Ingrese la Base para los archivos de la NN (sin extension): fir
Las constantes de la red fir seran cargadas en el workspace.

Accion:

  [ 1] Cargar Red existente y datos IO
  [ 2] Inicializar Red (crear o reinicializar)
  [ 3] Regenerar datos de Entrenamiento (y de Evaluacion)
  [ 4] Entrenar un conjunto de Epocas
  [ 5] Probar FF con el conjunto IO.Input
  [ 6] Probar FF con el conjunto IO.InputTest
  [ 7] Graficar entrada/salida
  [ 8] Graficar salida/salida
  [ 9] Editar constantes
  [10] Keyboard
  [11] Salir

pick a number, any number: 1


Se verificara la existencia de los siguientes datos de NN:
-----------------------------------------------------------
Estructura de NN: fir_Structure.ini
Entrenamiento, entrada: fir_ioInputTrain.txt
Entrenamiento, entrada Test: fir_ioInputTest.txt
Entrenamiento, salida: fir_ioIdealTrain.txt
Entrenamiento, entrada Test: fir_ioIdealTest.txt
Datos (Estructura+Pesos) de NN: fir_nnData.mat


Verificacion de archivos:
-------------------------
-> Archivo 'fir_Structure.ini' verificado OK.
-> Archivo 'fir_ioInputTrain.txt' verificado OK.
-> Archivo 'fir_ioInputTest.txt' verificado OK.
-> Archivo 'fir_ioIdealTrain.txt' verificado OK.
-> Archivo 'fir_ioIdealTest.txt' verificado OK.
-> Archivo 'fir_nnData.mat' verificado OK.

Cargando definicion de estructura de NN.
Generando neuronas 'de cero' para la NN.
Generando pesos 'de cero' para la NN.

Cargando red (puede estar entrenada o no).

Cargando data entrenamiento, pares entrada/salida.
Normalizando los datos de entrenamiento, Rango.
```

```
Accion:

  [ 1] Cargar Red existente y datos IO
  [ 2] Inicializar Red (crear o reinicializar)
  [ 3] Regenerar datos de Entrenamiento (y de Evaluacion)
  [ 4] Entrenar un conjunto de Epocas
  [ 5] Probar FF con el conjunto IO.Input
  [ 6] Probar FF con el conjunto IO.InputTest
  [ 7] Graficar entrada/salida
  [ 8] Graficar salida/salida
  [ 9] Editar constantes
  [10] Keyboard
  [11] Salir

pick a number, any number:
```

A continuación se utiliza la `opción 10` del programa para salir al modo `debug` y se pide el contenido de la variable `nnData`. La precaución a tomar, si se imprime la variable en pantalla, es tener en cuenta que el historial del error de entrenamiento está en esta variable y debe ser "achicada", por ejemplo, con el siguiente comando: `nnData.PredictionError = 0;`

En la tabla a continuación se muestran los primeros y últimos valores para esa variable de la estructura `nnData`.

```
nnData =

  scalar structure containing the fields:

    Files =
      scalar structure containing the fields:

        StructIni = fir_Structure.ini
        ioInputTrain = fir_ioInputTrain.txt
        ioInputTest = fir_ioInputTest.txt
        ioIdealTrain = fir_ioIdealTrain.txt
        ioIdealTest = fir_ioIdealTest.txt
        nnData = fir_nnData.mat

    Structure =
      scalar structure containing the fields:

        layerNumber =
           0
           1
           2

        layerNeurons =
           10
            1
            1

        layerActivation =
        {
          [1,1] = none
          [2,1] = tanh
          [3,1] = linear
        }

        FIR_FB =
           10
            0
            0

        fileNames =
        {
          [1,1] = fir
          [2,1] =
        }

    layersTotal =  3
```

```
Neurons =
{
  [1,1] =
     1.00000   1.00000
     0.00000  -0.36108
     0.00000  -0.22502
     0.00000   0.09121
     0.00000   0.34668
     0.00000  -0.14637
     0.00000   0.09081
     0.00000   0.45685
     0.00000   0.38862
     0.00000   0.06049
     0.00000  -0.12388

  [1,2] =
     1.00000   1.00000
    -0.23847  -0.23405

  [1,3] =
     1.00000   1.00000
    -0.20707  -0.20707

}
Weights =
{
  [1,1] = 0
  [1,2] =
   Columns 1 through 6:
     0.045001   0.657306   0.680847   0.582300   0.417576   0.076747

   Columns 7 through 11:
     0.017875  -0.137187  -0.061866   0.087078   0.027570

  [1,3] =
    -0.039555   0.729704

}

Deltas =
{
  [1,1] =
     0
     0

  [1,2] =
     0.00000
    -0.26567

  [1,3] =
     0.00000
    -0.38462
}

Epochs =  400

PredictionError =
  0.032617
  0.032112
  0.031449
  0.030850
  ...
  0.0050448
  0.0050434
  0.0050420
  0.0050406

State = 0

FilesBase = fir
```

Para volver al programa se puede tipear `dbcont` o para salir definitivamente, `dbquit`.

_Constantes.m

(El comienzo del archivo contiene el Nombre Base de la red)

Este archivo contiene las constantes que serán utilizadas para el entrenamiento de la red. Es un script que se ejecuta previo al entrenamiento, y que puede editarse eligiendo una de las opciones del menú principal de `nn22`. A modo de ejemplo se incluye el archivo `xor_Constantes.m`:

```
% ~~~~~~~ Constantes de inicializacion  ~~~~~~~
% ------- Red: XOR -------
%

% ~~~~~~~ Constantes de inicializacion de nn_intializeNN (Weights) ~~~~~~~
%
rndWeightsDistribution = 'normal';  % normal, uniform
rndWeightsMean = 0;
rndWeigthsDev = 0.25;

% ~~~~~~~ Constantes de inicializacion de nn_trainNN ~~~~~~~
%
trainIterations = 1000;      % Epocas
trainError = .01;            % Error que finaliza
trainEtaGain = 0.5;          % Ganancia de aprendizaje
trainShowTime = 1.0;         % Cada cuanto tiempo avisa por donde va (en segundos)

% ~~~~~~~ Constantes de inicializacion de nn_loadIOdata ~~~~~~~
%
ioCargarGenerar = 0;         % Cargar Datos (1=Cargar, 0=Generar)
ioNormalizeData = 1;         % ioNormalizeData (1=si, 0=no)
ioMaxMinVsMeanDev = 0;       % ioMaxMinVsMeanDev (1=MaxMin, 0=MeanDev)

% ~~~~~~~ Constantes de inicializacion de _generarIOdata ~~~~~~~
%
genExactOrRandom = 0;        % 1=exact, 0=random; en desuso, definida por nn_loadIOdata
genNumPairs = 40;            % f(tamaño de la red); ver % al final de nn_loadIOdata
```

_generarIOdata.m

(El comienzo del archivo contiene el Nombre Base de la red)

Esta función es específica de cada red y genera los datos de entrenamiento y prueba. Puede ejecutarse eligiendo una de las opciones del menú principal de `nn22`.

Su definición es la siguiente:

```
function [returnOK, X, Y] =
  xor_generarIOdata(nnFilesBase, genExactOrRandom, genNumPairs)
```

Sus parámetros cumplen las siguientes funciones:

a) nnFilesBase: se utiliza como base para el nombre en la creación de los archivos de datos
b) genExactOrRandom: 0, y=f(x)exacta; 1, random alrededor de y=f(x)
c) genNumPairs: cantidad de pares de entrada/salida en la epoca

A modo de ejemplo se incluye la función `xor_generarIOdata.m`:

```
function [returnOK, X, Y] = xor_generarIOdata(nnFilesBase, genExactOrRandom, genNumPairs)
    % ========== Carga de Datos de Entrenamiento =============
    %     function [returnOK, X, Y] =
    %    #_generarIOdata(nnFilesBase, genExactOrRandom, genNumPairs)
    %    - Generar datos. Funcion propia de cada planteo de red.
    %
    %    Parametros:
    %    ~~~~~~~~~~
    %    - nnFilesBase (xor, seno, etc.)
    %    - genExactOrRandom (0:y=f(x)exacta, 1:random alrededor de y=f(x))
    %    - genNumPairs (cantidad de pares de entrada/salida en la epoca)
    %
        _a = '';    % nnFilesBase, no puede estar vacio
        _b = 0;        % genExactOrRandom (0=no=random, 1=si=exact)
        _c = 20;    % genNumPairs
    switch nargin     % Completar argumentos opcionales.
        case 0
            fprintf('El argumento nnFilesBase no puede estar vacio\r\n');
            returnOK = 0; return
            % nnFilesBase = _a; genExactOrRandom = _b; genNumPairs = _c;
        case 1
            genExactOrRandom = _b; genNumPairs = _c;
        case 2
            genNumPairs = _c;
    end

    if (genExactOrRandom)
        X = [0.0, 0.0; 1.0, 0.0; 0.0, 1.0; 1.0, 1.0];
        Y = [0.0;      1.0;      1.0;      0.0];
    else
        % definir los cuatro grupos de entrada
        K = floor(genNumPairs/4); % muestras en cada clase
        q = .6; % offset

        A = [rand(K,1)-q, rand(K,1)-q];
        B = [rand(K,1)+q, rand(K,1)-q];
        C = [rand(K,1)-q, rand(K,1)+q];
        D = [rand(K,1)+q, rand(K,1)+q];

        % Armar las entradas y salidas
        X = [A; B; C; D];
        Y = [zeros(size(A,1),1); ones(size(B,1),1); ones(size(C,1),1); zeros(size(D,1),1)];
    end

    aux_graficarIOdata = str2func([nnFilesBase, '_graficarIOdata']);
    [returnOK] = aux_graficarIOdata(nnFilesBase, X, Y);
        if (~returnOK) returnOK = 0; return, end

    returnOK = 1; return
end
```

_graficarIOdata.m

(El comienzo del archivo contiene el Nombre Base de la red)

Esta función es específica de cada red y grafica los datos de que recibe en sus parámetros. Es llamada siempre a continuación de la generación de datos IO. También puede ejecutarse eligiendo una de las opciones del menú principal de `nn22`.

A modo de ejemplo se incluye la función `xor_graficarIOdata.m`:

```
function [returnOK] = xor_graficarIOdata(nnFilesBase, X, Y)
    % =========== Graficar Datos de Entrenamiento =============
    %    function [returnOK] =
    %       (...)_graficarIOdata(nnFilesBase, X, Y)
    %    - Graficar los datos Y contra X
    %
    %    Parametros:
    %    ~~~~~~~~~~~
    %    nnFilesBase (xor, seno, etc.)
    %    X, Y (pares de entrada/salida en la epoca)
    %

    h = figure();
    hold on;
    grid on;
    plot3(X([0*(size(X, 1)/4)+1:1*(size(X, 1)/4)],1), ...
          X([0*(size(X, 1)/4)+1:1*(size(X, 1)/4)],2), ...
          Y([0*(size(Y, 1)/4)+1:1*(size(Y, 1)/4)],1), 'ro', 'LineWidth', 2);
    plot3(X([1*(size(X, 1)/4)+1:2*(size(X, 1)/4)],1), ...
          X([1*(size(X, 1)/4)+1:2*(size(X, 1)/4)],2), ...
          Y([1*(size(Y, 1)/4)+1:2*(size(Y, 1)/4)],1), 'k+', 'LineWidth', 2);
    plot3(X([2*(size(X, 1)/4)+1:3*(size(X, 1)/4)],1), ...
          X([2*(size(X, 1)/4)+1:3*(size(X, 1)/4)],2), ...
          Y([2*(size(Y, 1)/4)+1:3*(size(Y, 1)/4)],1), 'b+', 'LineWidth', 2);
    plot3(X([3*(size(X, 1)/4)+1:4*(size(X, 1)/4)],1), ...
          X([3*(size(X, 1)/4)+1:4*(size(X, 1)/4)],2), ...
          Y([3*(size(Y, 1)/4)+1:4*(size(Y, 1)/4)],1), 'go', 'LineWidth', 2);
    hold off;

    fprintf('\nPresione Enter para continuar...\n');
    pause;
    close(h);

    returnOK = 1; return
end
```

_ioInputTrain.txt, _ioIdealTrain.txt, _ioInputTest.txt, _ioIdealTest.txt

(El comienzo del archivo contiene el Nombre Base de la red)

Estos archivos contienen respectivamente:

a. _ioInputTrain.txt: vector de datos de **entrada** para *entrenamiento*
b. _ioIdealTrain.txt: vector de datos de **salida** para *entrenamiento*
c. _ioInputTest.txt: vector de datos de **entrada** para *prueba*
d. _ioIdealTest.txt: vector de datos de **salida** para *prueba*

Están en formato de texto (`-ascii`) para que puedan ser editados manualmente. Esto es particularmente útil en el caso de redes sencillas como `AND` o `XOR` y también en el caso de vectores con muchos pares de entrada salida, como en el caso de la red `mcc` en la que se generó un vector con 5000 pares de entrada/salida cada entrada y cada salida a su vez con dos valores. Luego, manualmente mediante un editor de textos se partió ese conjunto en entrenamiento (3000 pares) y prueba (2000 pares).

A modo de ejemplo se incluyen los archivos de base `xor`:

xor_ioInputTrain.txt

```
0.00000000e+000 0.00000000e+000
1.00000000e+000 0.00000000e+000
0.00000000e+000 1.00000000e+000
1.00000000e+000 1.00000000e+000
```

xor_ioIdealTrain.txt

```
0.00000000e+000
1.00000000e+000
1.00000000e+000
0.00000000e+000
```

xor_ioInputTest.txt

```
-5.73877844e-001 1.70625845e-001
-5.78258757e-002 -5.69839140e-001
2.38182576e-001 -4.69331872e-002
-5.10030106e-001 -3.06508384e-001
-1.14633707e-001 -2.25060659e-001
-6.48748135e-002 -3.96460815e-001
-3.60170009e-002 1.45631118e-001
8.38521023e-002 -2.57411849e-001
-1.00437925e-001 -1.38463472e-001
-4.55517109e-001 1.82866229e-001
1.29022881e+000 -1.90012670e-001
1.18565185e+000 1.49331171e-001
1.33478045e+000 -4.41868771e-001
1.54425511e+000 -5.86699133e-001
1.10692635e+000 -3.87432296e-001
1.22310294e+000 -5.04985924e-001
1.48291371e+000 1.44877400e-001
9.83565330e-001 -5.88898985e-002
9.10606928e-001 -5.28902627e-001
1.19496644e+000 3.21985383e-001
2.95767886e-001 7.30824249e-001
-1.16080634e-001 1.53484370e+000
9.21655129e-002 1.16222493e+000
1.62539744e-001 7.81926797e-001
-1.65593902e-001 8.69914789e-001
1.83960246e-002 1.08395728e+000
2.96327197e-001 1.53939312e+000
1.68753801e-001 1.31597569e+000
1.51197946e-001 1.26558304e+000
-3.06001368e-001 1.48049232e+000
9.92143676e-001 6.15884027e-001
6.49693302e-001 7.31949628e-001
6.85844437e-001 1.03663969e+000
1.52203128e+000 8.89177225e-001
1.27322829e+000 1.07489650e+000
9.99612513e-001 1.06569157e+000
1.25996916e+000 6.62696026e-001
1.30957916e+000 9.18048395e-001
1.44504709e+000 1.03696685e+000
1.39516832e+000 1.26653021e+000
```

xor_ioIdealTest.txt

```
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
1.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
0.00000000e+000
```