Control Inteligente
Tarea 4

Jose Sebastian Osorio, David Vela Muñoz, Cristian David Lavacude

Índice

1. Punto 1:

Presente un pseudocódigo para el entrenamiento de una red neuronal tipo Perceptrón Multicapa (MLP) con funciones de activación sigmoidales unipolares e implemente y reporte el correspondiente código MatLab (bajo su estricta autoría) para aprendizaje supervisado bajo retro-propagación del error. La red tiene N entradas, M neurodos en la capa oculta, J neurodos en la capa de salida y tasa de aprendizaje Eta (η) [1]. Tenga en cuenta las recomendaciones de clase.

Para realizar el código se utilizó como base el diagrama de flujo que se encuentra en la figura 1.1 y de esta forma se separa el código.

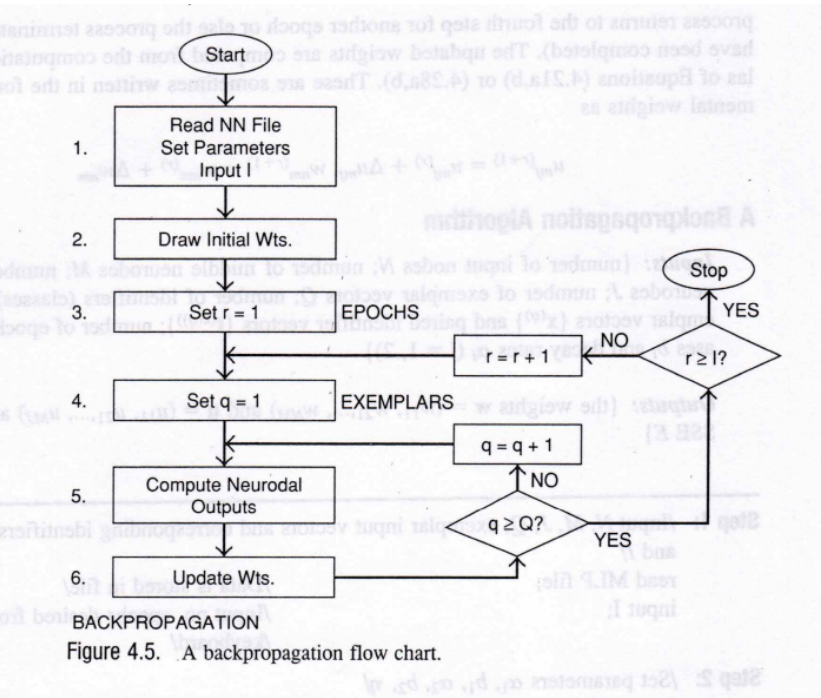

Figura 1.1: Diagrama de flujo.

Lo primero que se debe hacer es definir los parámetros de la Red Neuronal los cuales son:

- Número de entradas (N).
- Número de Neurodos de la capa Oculta (M).
- Número de Neurodos de la capa de Salida (J).
- Número de Ejemplos (Q).
- Número de Épocas (L).
- Taza de Aprendizaje (η).

- Inclinación función de activación (α).

Luego de esto se define la función de activación que se va a utilizar en la red neuronal, en este caso se va a trabajar con una función sigmoidal unipolar que tiene la siguiente forma:

$$f(x) = \frac{1}{1 + e^{-\alpha x}} \tag{1.1}$$

Se utilizan este tipo de funciones debido a que para entrenar la red por el método de retro-propagación del erro es necesario utilizar la derivada de la función de activación. Para integrar esto dentro del código se creó una función de MatLab la cual recibe la entrada x y el parámetro α y arroja como resultado la salida de la función $f(x)$ y su derivada $f'(x)$.

```
function [fun,der] = activation1(s,alpha)

fun = 1./(1+exp(-alpha*s));
der = fun*(1-fun);

end
```

Figura 1.2: Función de activación sigmoidal.

Primero, en el código se deben cargar/generar los datos de entrenamiento y validación que se utilizan en la red teniendo en cuenta que estos deben ser comparables entre si y además se debe evitar valores de 0 y 1 con el fin de no saturar los pesos y que el proceso de entrenamiento sea efectivo.

Posteriormente, se procede a inicializar los pesos y *bias* asociados con la capa oculta y la de salida. En este caso se seleccionan valores aleatorios al rededor de los cuales se quiere inicializar los pesos (esto depende de la aplicación en la que se va a utilizar la Red Neuronal). En el caso de los *bias* se inicializan en un valor de -1.

A partir de este momento se comienza a realizar el ajuste de los pesos el cual se encuentra dividido por épocas (L). El proceso que se realiza en cada época es el siguiente:

1. Seleccionar una cantidad Q de ejemplares para entrenamiento de la red, los cuales consisten en una combinación especifica de entradas para la cual se tiene conocimiento de la salida.

2. Con el primer ejemplo se procede a realizar el cálculo de las salidas de la Red Neuronal con los pesos que se tiene en el momento.

3. Conociendo la salida se procede a realizar la actualización de los pesos de la capa de salida usando la siguiente ecuación:

$$W_{o,mj} = W_{o,mj} + \eta \cdot (t_j - z_j) \cdot g'(s_j) \cdot y_m \tag{1.2}$$

Dónde η es la tasa de aprendizaje, t_j es el valor objetivo de ese ejemplo, z_j es la salida del neurodo j de la red neuronal, $g'(s_j)$ es la derivada de la salida del neurodo j de la red neuronal y y_m es la salida del neurodo m de la capa de oculta.

4. Teniendo en cuenta la salida del sistema y los pesos calculados para la capa de salida se procede a realizar la actualización de los pesos de la capa oculta usando la siguiente ecuación:

$$W_{h,nm} = W_{h,nm} + \eta \cdot \{\sum_{j=1}^{J}(t_j - z_j) \cdot g'(s_j) \cdot W_{o,mj}\} \cdot h'(r_m) \cdot x_n \tag{1.3}$$

   Dónde η es la tasa de aprendizaje, t_j es el valor objetivo de ese ejemplo, z_j es la salida del neurodo j de la red neuronal, $g'(s_j)$ es la derivada de la salida del neurodo j de la red neuronal, $W_{o,mj}$ es el peso que conecta el neurodo m de la capa oculta con el neurodo j de la capa de salida, $h'(r_m)$ es la derivada de la salida del neurodo m de la capa oculta y x_n es la entrada n a la red neuronal.

5. Se va acumulando el error que se tiene entre la salida deseada y la obtenida, con el fin de posteriormente calcular el *Error Total Medio Cuadrático.*

6. Los pasos 2 a 5 se realizan para cada uno de los ejemplares seleccionados (Q)

7. Cuando se ha recorrido los Q ejemplares se procede a medir el error que se obtuvo durante la época para lo cual se utilizan las siguientes medidas:

   - Error Total Medio Cuadrático.
   - *Variance Accounted For*(VAF).

8. Ya para terminar se realiza un proceso de validación del modelo usando unos datos diferentes a los de entrenamiento, pero con sus misma estructura, en el cual se calcula nuevamente la salida de la Red Neuronal pero usando los pesos actualizados, es decir, se realiza la etapa de validación.

9. Finalmente se puede calcular las medidas de error para los datos de validación o en el caso de que la red esté modelando un sistema dinámico, se pude graficar la salida del modelo y comprobar visualmente si esta coincide con los datos objetivo de validación.

Una vez alcanzado el número de épocas (L) se puede concluir a partir de las gráficas de error si la red se encuentra correctamente entrenada y de no ser el caso se pude variar los parámetros mostrados inicialmente según lo considere el diseñador.

El código desarrollado es el que se utilizó para realizar los puntos 2 y 3 de este taller y se encuentra en los archivos *ANN1.m* y *ANN2.m* con sus respectivos comentarios y modificaciones. Las funciones de activación usadas se encuentran en los archivos *activation.m* y *activation2.m.*

2. Punto 2:

Considere el ejemplo de la función lógica XOR como un caso de clasificación binaria con dos entradas x_1 y x_2 y dos salidas z_1 y z_2. Conforme la base de datos requerida para entrenamiento y validación. Aplique el programa desarrollado en el punto anterior y reporte el número de pasadas de épocas requeridas para el aprendizaje. Presente gráficas y comentarios de análisis de:

Tabla de verdad

Para la tabla de verdad de la función lógica XOR se se decide remplazar los 1 por 0,9 y los 0 por 0,1, con el objetivo de evitar saturaciones en la red neuronal artificial.

x_1	x_2	z_1	z_2
0.1	0.1	0.1	0.9
0.1	0.9	0.9	0.1
0.9	0.1	0.9	0.1
0.9	0.9	0.1	0.9

Tabla 2.1: Tabla de verdad

Representación de XOR

Los valores x_1 y x_2 corresponden a las coordenadas (x_1, x_2) de un elemento. El objeto de la ANN es clasificar estos elementos en dos clases. Los valores z_1 y z_2 son los valores deseados, en donde cada salida corresponde a una de las dos clases y su valor de salida es la pertenencia o no de los datos de entrada para la clase. En la siguiente figura se ejemplifica la situación, en donde existen cuatros círculos y se clasificarían en dos clases las clase círculos amarillos (z_2) y círculos azules (z_1).

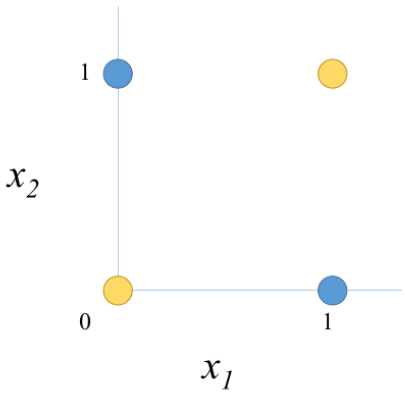

Figura 2.1: XOR

Características de la ANN

Para clasificar los objetos se emplea una ANN con las siguientes características:

- Número de entradas N=2
- Número de Neurodos de la capa Oculta M=3
- Número de Neurodos de la capa de Salida J=2
- Número de Ejemplo Q=4
- Número de Épocas L=2000
- Taza de Aprendizaje η=1
- Inclinación función de activación α=1

Código de inicialización de peso

Para inicializar los pesos de la capa oculta y de la capa de salida se emplean el siguiente código de $Matlab$.

```
% Capa oculta
Wh = (0.1)*rand(N,M);
Bh = ones(1,M)*-1;

% Capa salida
Wo = (0.1)*rand(M,J);
Bo = ones(1,J)*-1;
```

Figura 2.2: Código de inicialización de peso

Se emplea Bh y Bo para inicializar los arreglos de los bias en -1. Los Wh y los Wo corresponde a los pesos de la capa oculta y a la capa de salida respectivamente, estos pesos se inicializan en el intervalo (0, 0.1).

Datos de entrenamiento

Para alimentar la red es necesario conformar la base de datos para entrenamiento y para validación. Los datos de entrenamiento estarán dados por la siguiente algoritmo.

```
for i = 1:L
     X = [rand(1)*(0.15 - 0.05) + 0.05, rand(1)*(0.15 - 0.05) + 0.05;
          rand(1)*(0.15 - 0.05) + 0.05, rand(1)*(0.95 - 0.85) + 0.85;
          rand(1)*(0.95 - 0.85) + 0.85, rand(1)*(0.15 - 0.05) + 0.05;
          rand(1)*(0.95 - 0.85) + 0.85, rand(1)*(0.95 - 0.85) + 0.85];
end
```

Figura 2.3: Código para datos de entrenamiento

Los pares de entradas se representan por $X = [x_1, x_2]$. Se busca que las entradas varíen en valores cercanos, se realiza el algoritmo para las cuatro posibles combinaciones . Para la primer combinación $x_1 = 0{,}1$ y $x_2 = 0{,}1$ van a variar en un intervalo (0.05, 0.15). Para el segundo par de ejemplares $x_1 = 0{,}1$ y $x_2 = 0{,}9$, $x_1 = 0{,}1$ varia en un intervalo (0.05, 0.15) y $x_2 = 0{,}9$ variar en intervalo (0.85, 0.95). Para el tercera combinación $x_1 = 0{,}9$ y $x_2 = 0{,}1$, x_1 varia en un intervalo (0.85, 0.95) y x_2 variar en intervalo (0.05, 0.15). Para la cuarta combinación $x_1 = 0{,}9$ y $x_2 = 0{,}9$ van a variar en un intervalo (0.85, 0.95). Los datos por combinación sera de 2000 datos y en total serán 8000 datos para entrenamiento. La generación de los datos se realiza cada época.

Target

El target con el cual se entrena la red neuronal es:

```
targ = [0.1 0.9;
        0.9 0.1;
        0.9 0.1;
        0.1 0.9];
```

Figura 2.4: Código para el target

Datos de verificación

Para los datos de verificación se emplea el siguiente algoritmo.

```
%% Matriz con datos de validación

for i = 1:L
    valid(i+(i-1)*3,:)   = [rand(1)*(0.15 - 0.05) + 0.05,rand(1)*(0.15 - 0.05) + 0.05,0.1,0.9];
    valid(i+1+(i-1)*3,:) = [rand(1)*(0.15 - 0.05) + 0.05,rand(1)*(0.95 - 0.85) + 0.85,0.9,0.1];
    valid(i+2+(i-1)*3,:) = [rand(1)*(0.95 - 0.85) + 0.85,rand(1)*(0.15 - 0.05) + 0.05,0.9,0.1];
    valid(i+3+(i-1)*3,:) = [rand(1)*(0.95 - 0.85) + 0.85,rand(1)*(0.95 - 0.85) + 0.85,0.1,0.9];
end
```

Figura 2.5: Código para datos de verificación

Los datos de validación se guardan en un arreglo llamado *valid*, el cual cuenta con 4 columnas y 2000 pares de entradas. Las dos primeras columnas corresponden a los valores aleatorios

de las entradas generados para las cuatro posibles combinaciones, mientras que las dos ultimas columnas son los valores de salida esperados y que se emplearan para la verificación.

Del total (10000), el 80 % (8000 datos) se emplean para el entrenamiento y el restante 20 % (2000 datos) se emplean para la validación.

2.1. Comportamiento del error medio cuadrático de entrenamiento y validación

Error total medio cuadrático

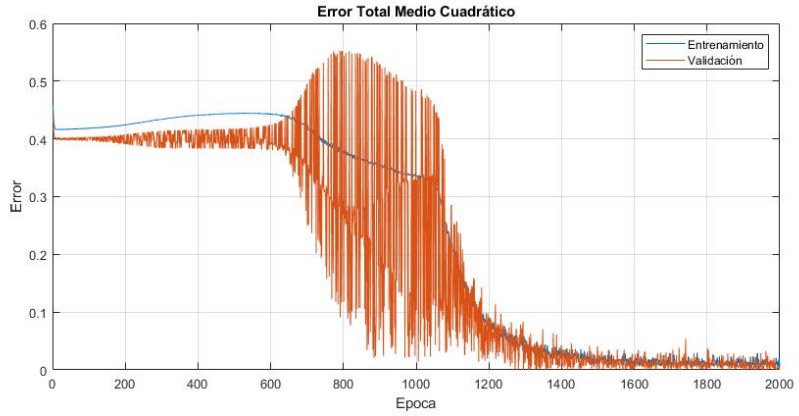

Figura 2.6: Error total medio cuadrático

VAF

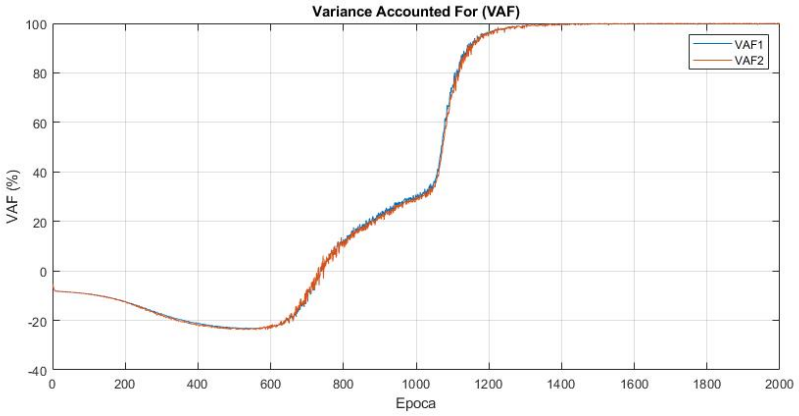

Figura 2.7: VAF

El error medio cuadrático de validación en comparación con el error medio cuadrático de entrenamiento empiezan a decrecer, sin embargo, al rededor de las 650 épocas los datos de

validación tiene un error elevado para algunos ejemplares, lo que significa que se necesitan más épocas hasta que se logre una convergencia.
El error medio cuadrático de entrenamiento y de validación convergen alrededor de las 1400 épocas lo cual se puede verificar también en la gráfica del VAF. Se pude concluir que en las 1400 épocas se logra una granulidad o robustez suficiente, lo cual permitirá que la red neuronal artificial se comporte bien con datos con los cuales no fue entrenada.

2.2. Evolución de los pesos de la red

Pesos capa oculta

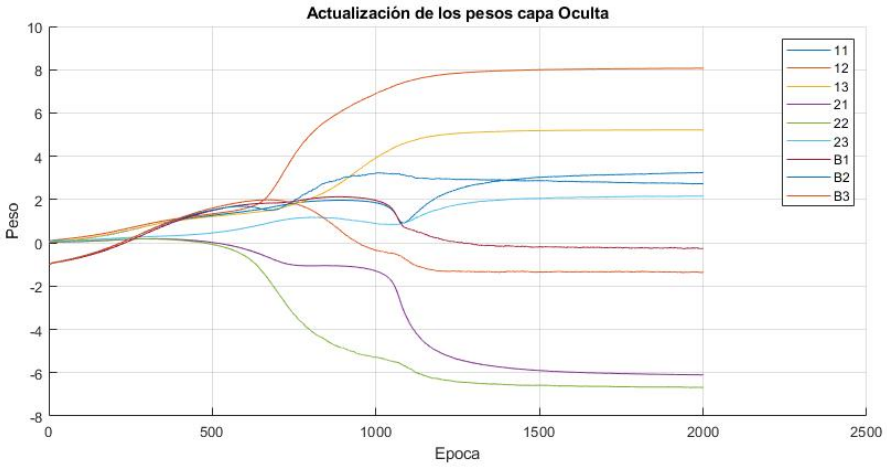

Figura 2.8: Convergencia de los pesos

Los seis pesos $(11, 12, 13, 21, 22, 23)$ relaciona las dos entradas con los tres neurodos de la capa oculta y los pesos $(B1, B2, B3)$ corresponde a los pesos de los bias. Se comprueba por medio de la gráfica anterior que todo los pesos a las 1400 épocas se estabilizan.

Pesos capa de salida

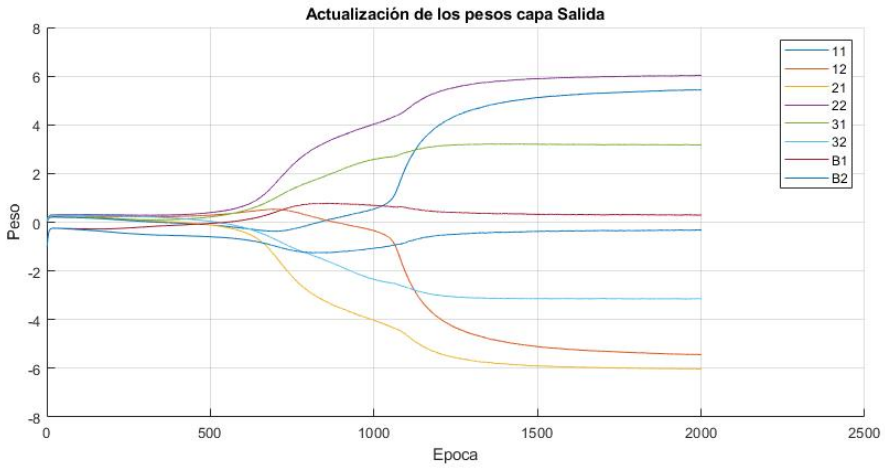

Figura 2.9: Convergencia de los pesos

Los seis pesos $(11, 12, 21, 22, 31, 32)$ relaciona los neurodos de la capa oculta con los neurodos de la capa de salida y los pesos $(B1, B2)$ corresponde a los pesos de los bias de la capa de salida. Se comprueba por medio de la gráfica anterior que todo los pesos a las 1400 épocas se estabilizan.

2.3. Rectas separadoras (asociadas a salidas de neurodos) iniciales y finales

Rectas separadoras de la capa oculta

Se puede ver (figura 2.10) el comportamiento de cada uno de los tres neurodo en la capa oculta. Se gráfica las salidas de los neurodos antes del entrenamiento, las cuales son las gráficas de la izquierda. En la derecha se gráfica las salidas de los tres neurodos después del entrenamiento.

Se emplea $X_1 = 0 : 0{,}025 : 1$; y $X_2 = 0 : 0{,}025 : 1$; para formar una arreglo de $41x41$, con todas las combinaciones se obtienen la salidas de todos los neurodos y se grafican por medio de un mapa de calor.

Para las rectas separadoras de la capa oculta, el primer neurodo refleja el entrenamiento en donde los valores que se encuentren en la parte superior izquierda es decir cercanos a los valores de (0,1) activaran la función de activación de ese neurodo obteniendo valores cercanos a 1.

Para las rectas separadoras de la capa oculta, el segundo y tercer neurodo refleja el entrenamiento en el cual la función de activación se obtendrá valores cercanos a 1 para las posiciones de color amarillo.

Rectas separadoras de la capa de salida

En la figura 2.11 se puede evidenciar el comportamiento de los dos neurodos de la capa de salida. Los gráficos de los dos neurodos son complementarios lo cual tiene sentido con la pertenencia o no a un grupo.

En la salida del primer neurodo z_1(Después del entrenamiento), los valores que se encuentran cercanos a los puntos (0, 1) y (1, 0) activaran la función sigmoide. Se puede concluir que el comportamiento es el deseado para estos valores.

En la salida del segundo neurodo z_2(Después del entrenamiento), los valores que se encuentran cercanos a los puntos (0, 0) y (1, 1) activaran la función sigmoide. Se puede concluir que el comportamiento es el deseado para estos valores. Aunque se arrojan valores intermedios (0.5, 0.5) para las dos clases, estas salidas carecen de sentido..

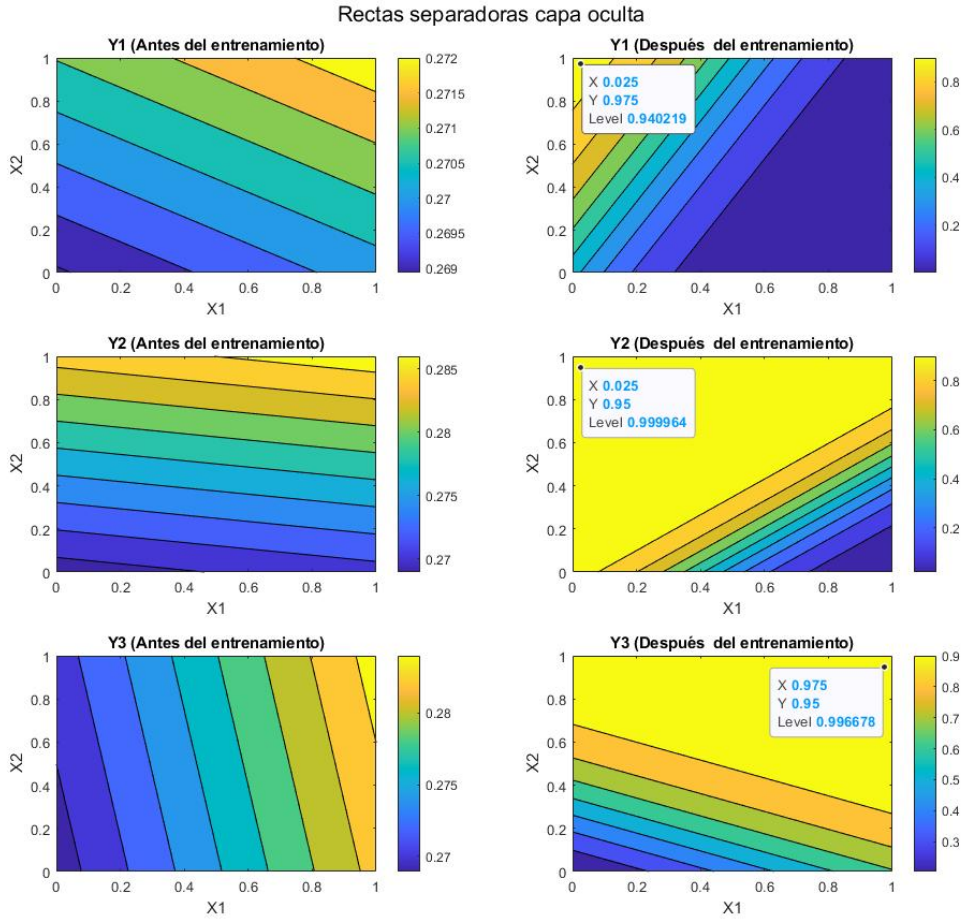

Figura 2.10: Rectas separadoras de la capa oculta

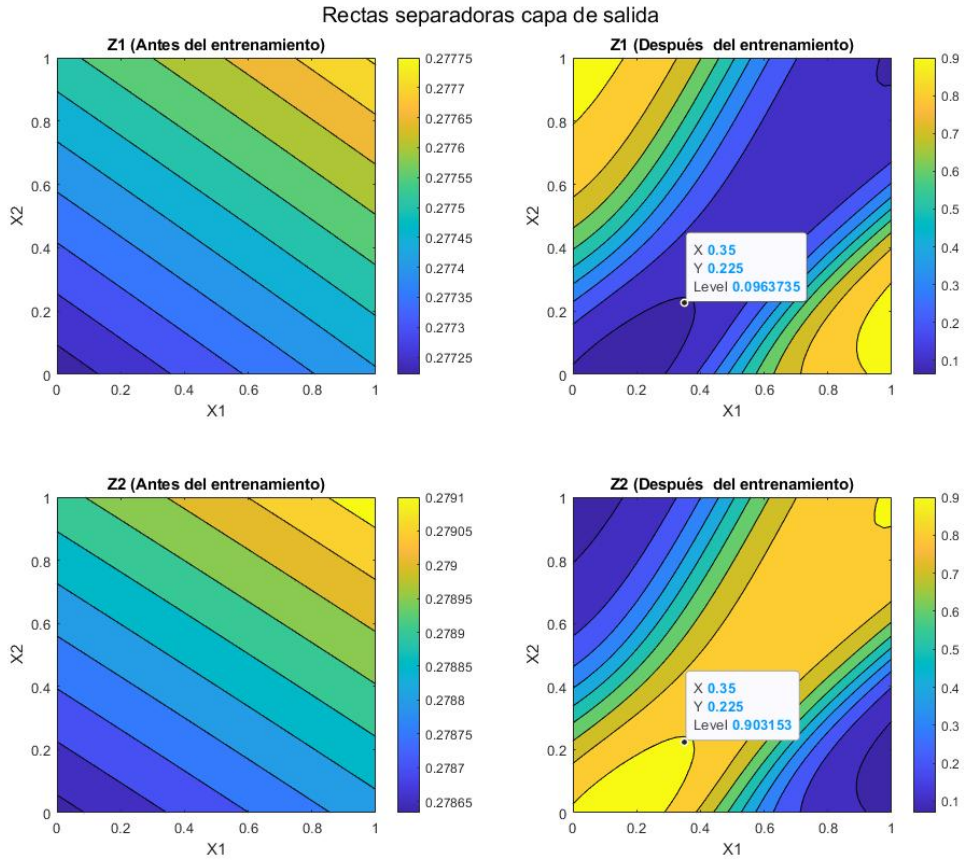

Figura 2.11: Rectas separadoras de la capa de salida

3. Punto 3:

Con el apoyo la literatura del Deep Learning Toolbox de Matlab (que incorporó el Neural Network Toolbox), se considera el sistema dinámico no lineal del CSTR (Continuous Stirred Tank Reactor, p.2664) con estructura NARX conocida. Haciendo uso del código desarrollado en el punto 1 y los datos de entrada-salida provistos en el modelo Simulink, se construye y entrena una red neuronal artificial tipo MLP para aproximar la función de salida y(k+1) del sistema dinámico.

3.1. Arquitectura final de la red (N-M-J):

La arquitectura final de la red neuronal entrenada es:

- Se emplean un total de cuatro entradas (**N=4**) en los que se toma un segundo orden para los datos de entrada y salida provistos por el modelo Simulink del Toolbox, es decir, como entrada se tomarán los valores *y(k), y(k-1), u(k) y u(k-1)* , teniendo como objetivo que $y(k+1) = f(y(k), y(k-1), u(k), u(k-1))$. Por lo que en la red neuronal artificial el target depende de estas cuatro entradas.
- Se realiza el entrenamiento con siete neurodos en la capa oculta **(M=7)** y un neurodo en la capa de salida **(J=1)**.

Con esta arquitectura de red también se tiene en cuenta que el factor de aprendizaje $\boldsymbol{\eta = 0{,}001}$,este dato se escoge empíricamente mediante prueba y error, ya que si este factor es muy alto no se logra un resultado satisfactorio; en el caso de la función de activación $\boldsymbol{\alpha = 0{,}25}$, para la capa oculta:$f(x) = 1/(1 + e^{-\alpha x})$ y para la capa de salida:$f(x) = x$. Se emplean un total de 8000 ejemplos **(Q=8000)** de los cuales 6500 son usados para entrenamiento y 1500 para validación, en total se realizan 350 épocas **(L=350)**.

3.2. Gráfica y resultado numérico de validación en el tiempo usando Mean Squared Error (MSE) y VAF (Variance Accountig For)

Las respectivas gráficas de de MSE y VAF para un total de 350 epocas se muestran a continuación:

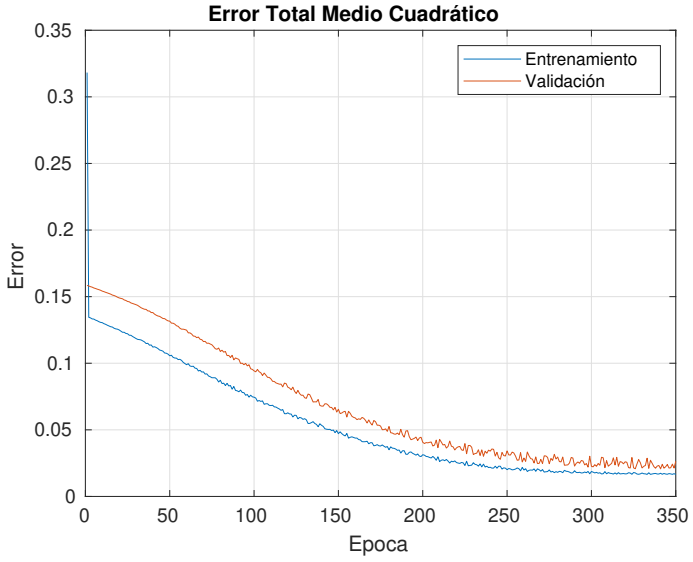

Figura 3.1: Error medio cuadrático. L=350

Según la figura 3.1, el valor final del MSE para el *entrenamiento* es de **0.0165** y en el caso de la *validación* es de **0.0264**.

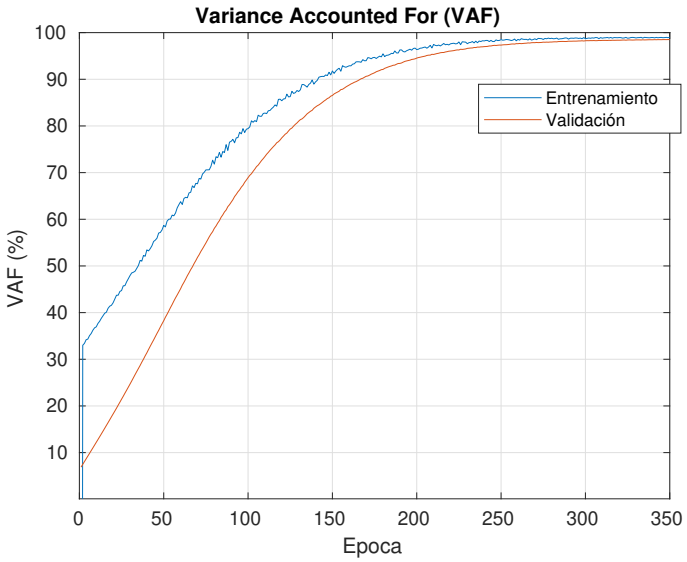

Figura 3.2: VAF. L=350

En el caso del VAF, según la figura 3.2, el valor final para *entrenamiento* es de **98.9947 %** y para la *validación* de **98.4935 %**.

Como gráficas adicionales se presenta la comparación entre el target (datos de salida de la planta de simulink normalizados entre 0.1 y 0.9) y la salida de la red neuronal en la época 350.

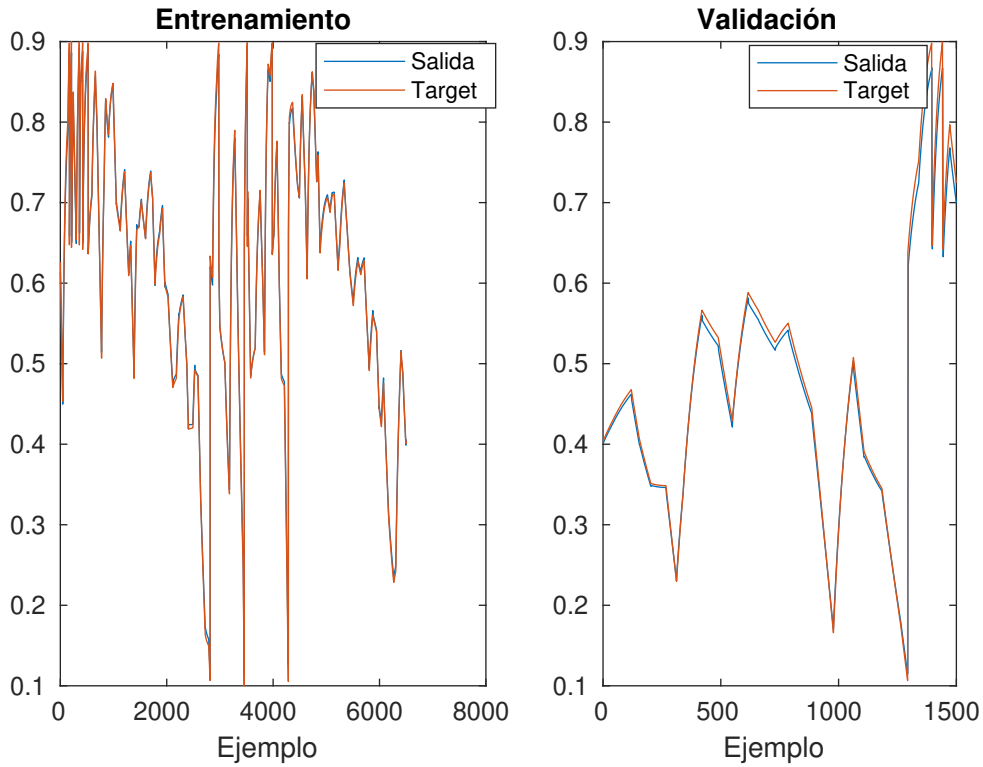

Figura 3.3: Target y salida para entrenamiento y validación época 350.

Como se observa en la figura 3.3 la salida para los 8000 ejemplos en la época 350 es casi igual a el target de la red neuronal, verificando las gráficas de MSE y VAF.También se presenta en la figura 3.4 la salida de la red neuronal (convirtiendo los datos normalizados a la misma escala de la salida) vs la salida de la planta con los datos originales (sin normalizar), además de su respectivo error.

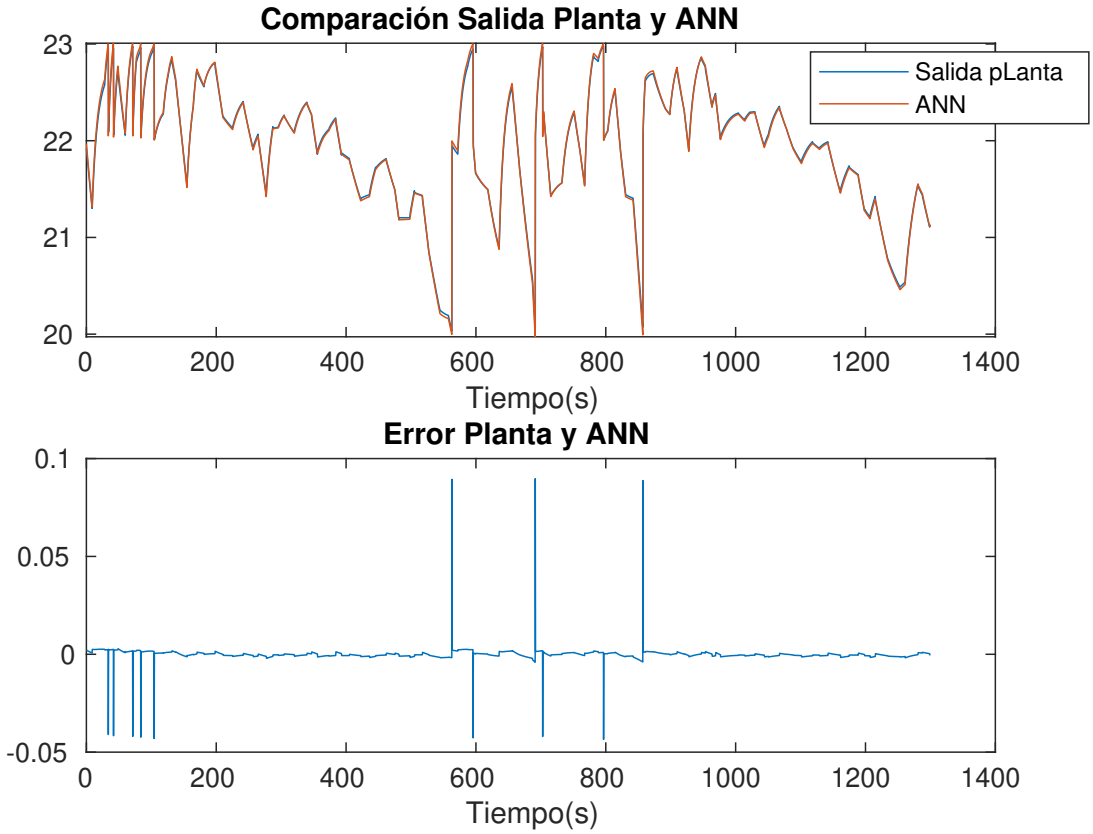

Figura 3.4: Salida Planta vs Red neuronal

3.3. Enumeración y breve explicación de aspectos a tener en cuenta en el modelado de sistemas dinámicos no lineales usando ANN (versus modelado de sistemas estáticos).

En el modelado de sistemas dinámicos no lineales usando ANN de deben tener en cuenta los siguientes aspectos:

- La información de entrada y salida del sistema no lineal al ser mas compleja, se debe organizar y operar adecuadamente. Además, de elegir los ordenes de entrada y salida de la planta, también se debe normalizar la información que se va a ingresar a la red neuronal, tanto las entradas como los targets, en este ejemplo se normalizaron de tal manera que los valores quedaran expresados entre 0.1 y 0.9,para no saturarla, de tal manera que el entrenamiento se pueda realizar adecuadamente.
- Se debe prestar atención especial a la tasa de aprendizaje η, ya que si esta es inadecuada, la red presentará problemas cuando recorra los ejemplos, llevando a un resultado incorrecto en el entrenamiento de la misma.