Instituto Politécnico Nacional

Escuela Superior de Cómputo

Asignatura:

Neural Networks

Grupo: 3CM2

Práctica .5 **MLP (Perceptrón Multicapa)**

Alumno:

Garcia Garcia Rafael

Profesor: Moreno Armendáriz Marco Antonio

Introducción:

Después de que las redes neuronales perdieron relevancia por muchos años la segunda clave para el desarrollo de esta área de conocimiento fue el desarrollo de redes neuronales multicapa la y su enorme cantidad de creativas variaciones las cuales abrieron nuevos conceptos y reglas de aprendizaje.

De igual manera cuando se tenía la arquitectura, y contexto del funcionamiento de estas redes neuronales hubo un gran enigma de como se podían entrenar aquí es donde reluce la aportación del algoritmo de backpropagation elaborado por David Rumelhart y James McClelland el algoritmo de backpropagation el cuál nos ayudo a crear redes neuronales mucho más poderosas con soluciones más generalizadas

En general el MLP pondrá a prueba todos los conocimientos adquiridos en el curso, desde abarcando conceptos de múltiples neuronas y capas en las redes neuronales, aprendizaje de las redes, verificación de errores, múltiples neuronas en las capas, funciones de transferencia, etc.

Marco teórico.

El MLP es una de las arquitecturas que sirven tanto como clasificares como de modo regresivo, en el caso de la práctica nuestro MLP está configurado en modo regresivo debido a las funciones de transferencia que la componen destacando en ellas *purelin* de 1 neurona en la capa de salida.

El MLP ocupa como método de aprendizaje el algoritmo de backpropagation este algoritmo como su nombre lo dice trata de propagar desde la última capa hasta la red buscando encontrar las derivadas de las funciones de transferencia de las neuronas de cada una de las capas, encontrando esta derivada se pueden ajustar los valores de *pesos* y *bias* de la red para poder aprender.

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a}),$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1,$$

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \ldots & 0 \\ 0 & \dot{f}^m(n_2^m) & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & \dot{f}^m(n_{S^m}^m) \end{bmatrix},$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}.$$

Imagen 1. Ecuaciones para calcular la sensibilidad y derivadas de la función de transferencia (Hagan)

El algoritmo está definido por las siguientes ecuaciones:

Calculando las sensibilidades de cada una de las capas y las derivadas de las capas de las funciones de transferencia pasamos a la parte de la asignación de los nuevos *pesos* y *bias* como se observa en la **imagen 2**.

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T,$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m.$$

Imagen 2. Ecuaciones para calcular nuevos pesos y bias (Hagan)

Cabe mencionar que estas ecuaciones de aprendizaje solo se aplican en el caso de que la red presente un error más elevado de lo esperado, este error se calcula por medio de la ecuación *"Mean square error"* dada por la siguiente ecuación.

$$E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2]$$

Imagen 3. Error por suma de cuadros (Hagan)

Cabe que para calcular el error total de la red se tienen que sumar todos los errores y dividirse en el número de ejemplos con los que entrenó.

Explicación de la arquitectura

Pasando a la explicación de la arquitectura nuestro MLP fue programado para arquitecturas de 1 o 2 capas ocultas teniendo una última capa de salida la arquitectura para la que tiene 2 capas ocultas queda muy similar a la que se muestra en la ***imagen 4*** faltando por definir las funciones de transferencia y neuronas, para la de 1 capa oculta es similar solo que la capa de salida se encuentra en la segunda capa.

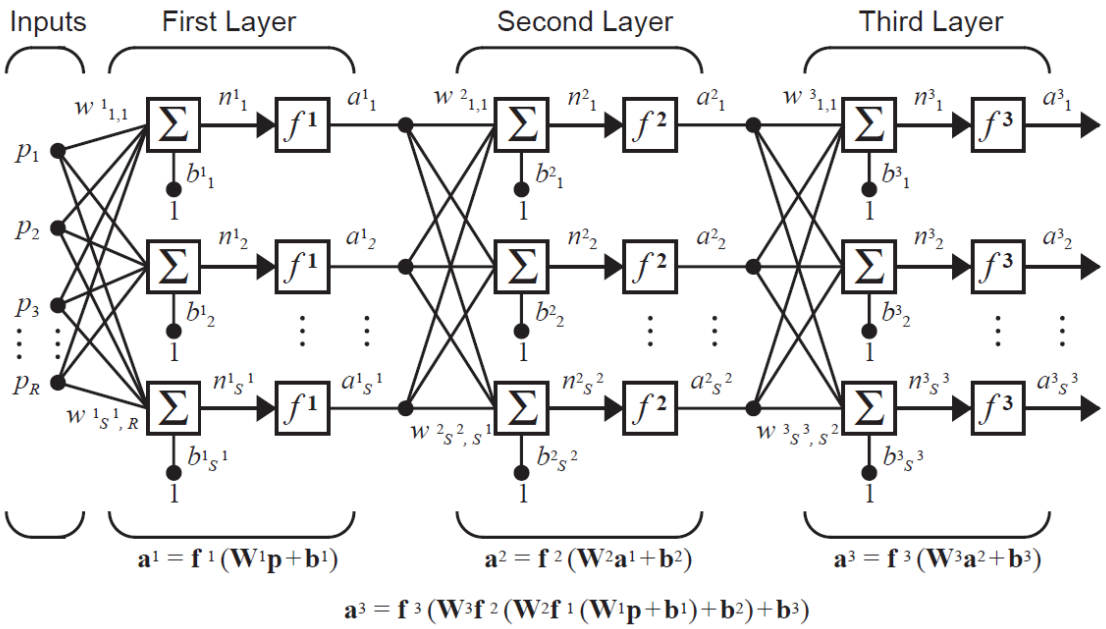


Imagen 4. MLP 3 capas (Hagan)

Definiendo nuestra arquitectura queda por último aclarar que para la definición del número de neuronas y funciones de transferencia se usarán 2 vectores de entrada para configurarla siendo definidos por la siguiente manera

v1: [1 S^1 1] ó [1 S^1 S^2 1]

v2: [2 1] ó [3 2 1]

en donde v1 indica en el primer índice el número de R del vector p y S^1 el número de neuronas de la capa oculta, en el último índice tenemos un 1 por defecto que índica que la capa de salida tiene 1 neurona (valor que esperamos obtener perteneciente a $\mathbb{R}$).

Ahora, v2 es un vector que define la función de transferencia de la red, desde la primera capa hasta la última, para entender esto mejor vamos a definir la numeración de las funciones de la siguiente manera:

1 = Purelin

2 = LogSig

3 = TanSig

Podemos observar que por default nuestro valor en la última capa es 1, esto porque se busca un MLP en modo regresivo y para esta configuración se necesita tener la función de transferencia purelin () en la última capa.

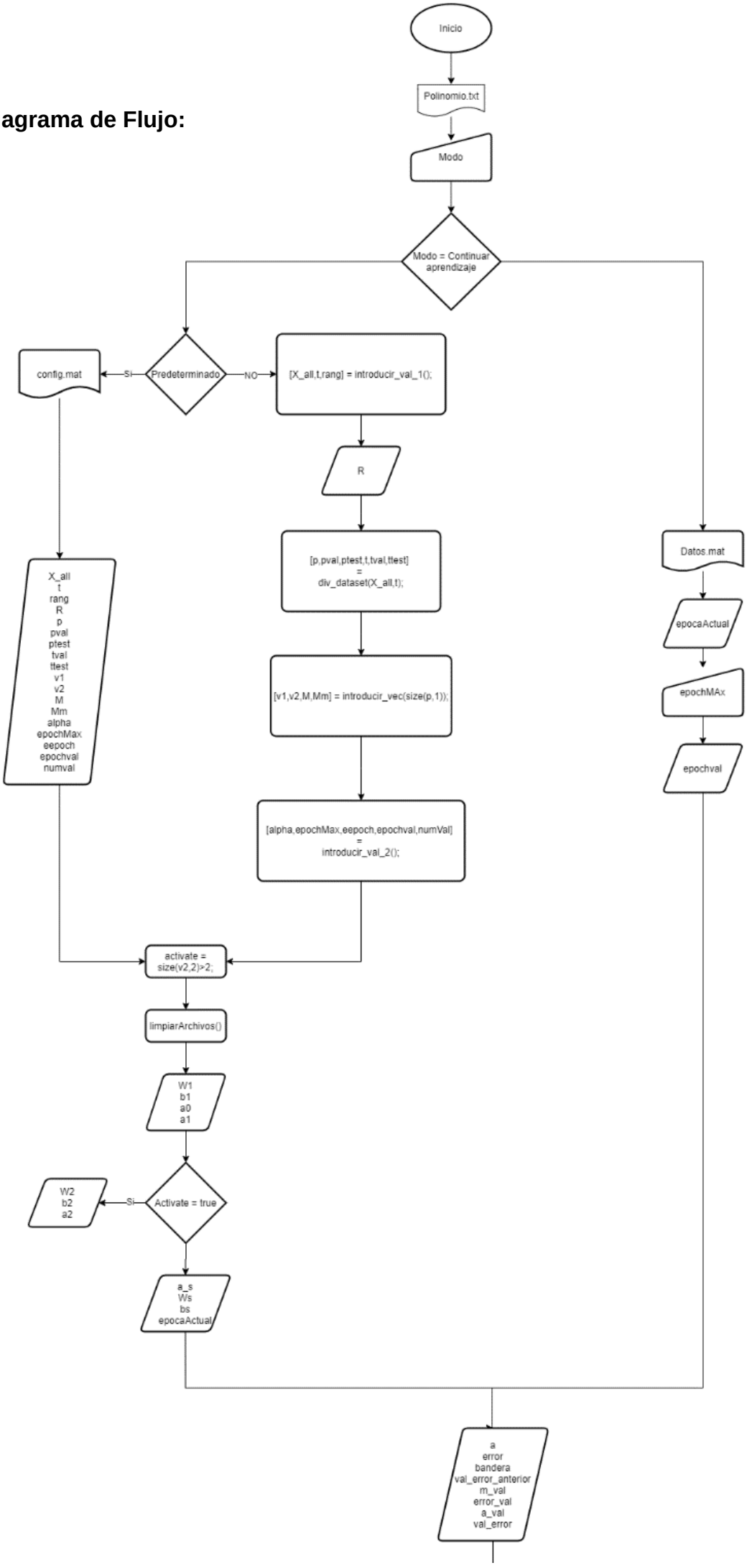
Explicada la arquitectura de nuestra red pasamos a definir 2 últimas cosas:

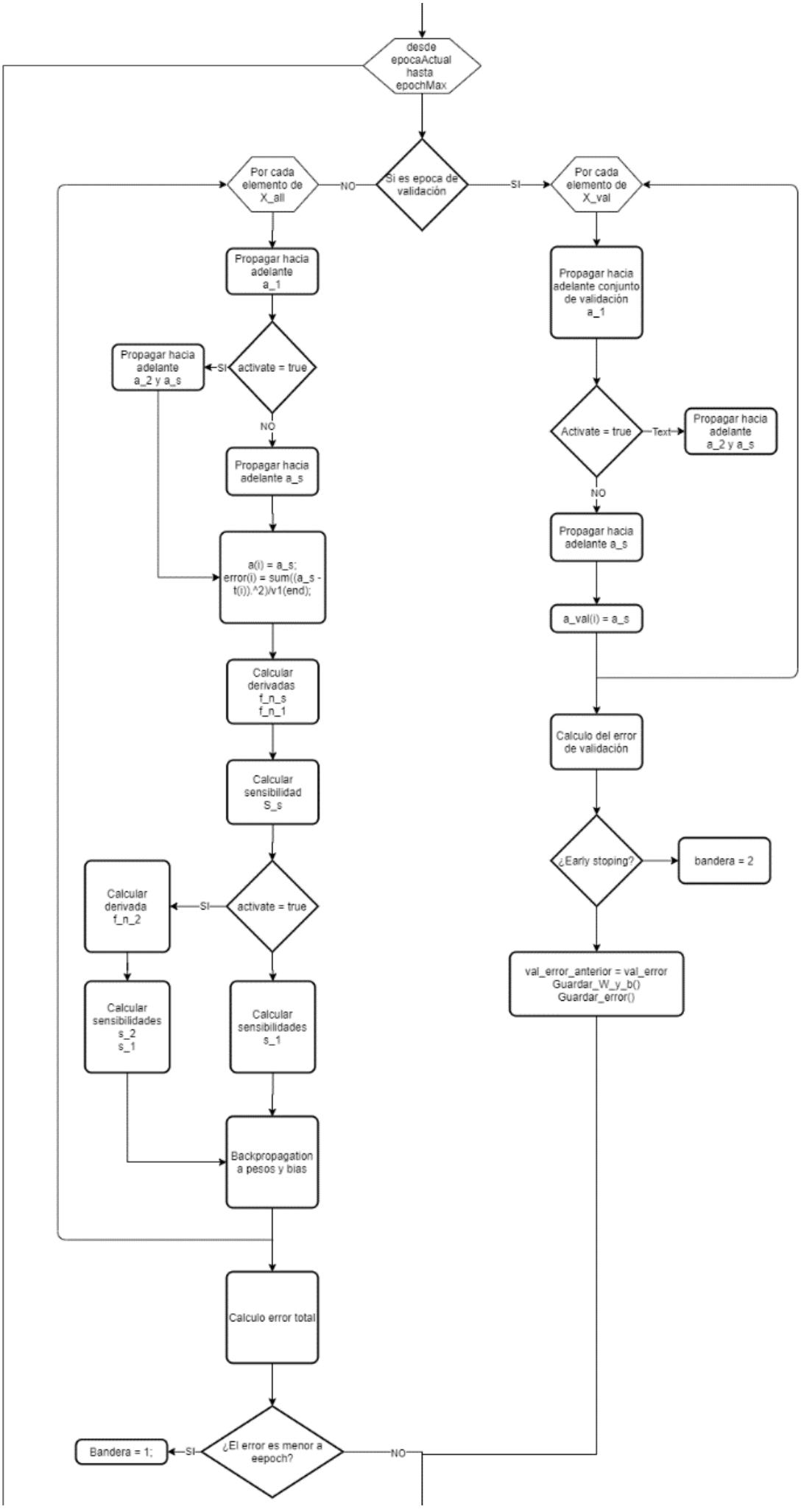
- Para la creación del dataset de entrenamiento se dividirá el dataset original en tres partes: Entrenamiento, Validación, Prueba.

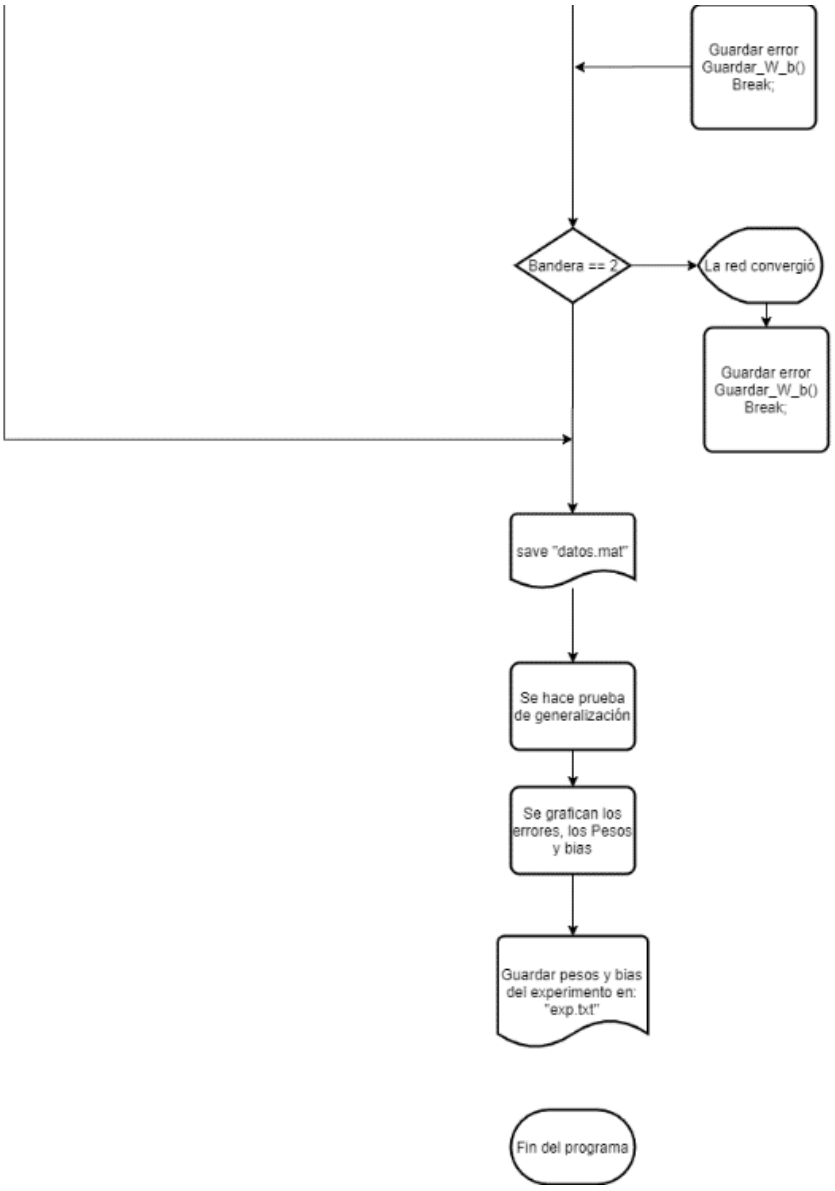
- Se implementará el algoritmo Early Stopping para que en cada %10 del total de iteraciones se valide si nuestra red está aprendiendo

Los puntos antes mencionados nos ayudarán para hacer pruebas con nuestro entrenamiento de la red y podamos asegurarnos de que está aprendiendo.

Diagrama de Flujo:




desde epocaActual hasta epochMax
Si es epoca de validación
SI
NO
Por cada elemento de X_all
Propagar hacia adelante a_1
activate = true
SI
Propagar hacia adelante a_2 y a_s
NO
Propagar hacia adelante a_s
a(i) = a_s; error(i) = sum((a_s - t(i)).^2)/v1(end);
Calcular derivadas f_n_s f_n_1
Calcular sensibilidad S_s
activate = true
SI
Calcular derivada f_n_2
Calcular sensibilidades s_2 s_1
Calcular sensibilidades s_1
Backpropagation a pesos y bias
Calculo error total
¿El error es menor a eepoch?
SI
Bandera = 1;
NO
Por cada elemento de X_val
Propagar hacia adelante conjunto de validación a_1
Activate = true
Text
Propagar hacia adelante a_2 y a_s
NO
Propagar hacia adelante a_s
a_val(i) = a_s
Calculo del error de validación
¿Early stoping?
bandera = 2
val_error_anterior = val_error
Guardar_W_y_b()
Guardar_error()



Experimentos.

La primera parte de los experimentos fue de vital importancia ya que nos ayudo si la implementación de nuestro MLP fue de manera efectiva teniendo en general resultados bastantes positivos. A continuación, se anexarán unas tablas donde se registraron los resultados de cada experimento.

Primera parte:

Grado del polinomio	Intento	Early Stopping	Arquitectura	# de épocas	α	TrainError	TestError	ValError
1	1	SI	V1 = [1,2,1,1] V2 = [1,1,1]	20999	0.0008	t_error 0.4565	test_error 0.6412	val_error 0.4285
1	2	NO	V1 = [1,5,4,1] V2 = [3,3,1]	330000	0.0008	t_error 0.1879	test_error 0.2132	val_error 0.0631
1	3	NO	V1 = [1,5,4,1] V2 = [3,3,1]	50000	0.008	t_error 8.4505e-04	t_error 8.4505e-04	val_error 5.0439e-04
1	5	NO	V1 = [1,10,10,1] V2 = [2,2,1]	457,611	0.0004	t_error 1.0000e-04	test_error 3.7796e-04	val_error 3.0975e-04

Salida de la terminal:

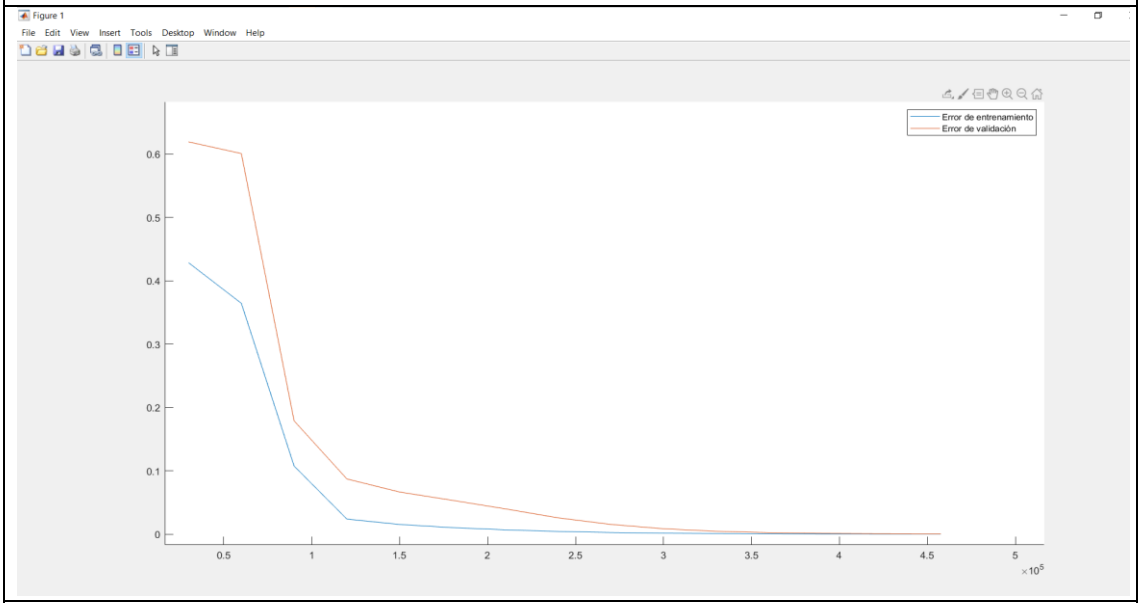
```
Error: 0.000100 de la epoca 457610
Error: 0.000100 de la epoca 457611
La red convergió
Prueba de generalización superada con un error de:
   3.7796e-04

Error final
   1.0000e-04

Valores de pesos finales:
W1
   -3.2430
   -4.2519
   -4.5863
    2.2076
   -7.4825
    2.6792
    5.0213
    2.8120
    2.6989
   -2.4960

b_1
   -0.9486
   -1.9576
    3.8524
   -0.6019
   -6.5952
   -1.1770
    7.6004
   -5.3827
    4.9660
    3.2648

W_2
   -1.8009   -3.2051    0.9621    0.1994    2.9691    1.6334    4.3700   -3.2704   -0.8310    1.4138
    0.6324   -1.0266    1.4013    0.5614    2.1951    0.9166   -0.5035   -0.8399   -0.1472   -0.4701
   -2.0790   -2.4708   -0.6476    1.1179    6.1202    3.7717    2.0066   -3.4185   -3.5120    0.0815
   -0.4396    0.2772    0.5196    0.6301   -1.0144    0.8117   -1.3404   -1.2240   -1.4968    0.1331
    0.7038   -0.3566    1.6199   -0.5891    0.7803    1.2948   -0.3546   -1.2585    0.3180    0.7862
   -0.1318   -0.8671   -0.5763    0.3591    0.6159    0.9070    0.7774   -1.5299   -2.0530    1.1143
   -0.3794    0.6107    1.0170   -0.4683    0.9720    0.5410   -0.4348   -0.2277    0.3111    0.7990
    1.4053    1.2439    5.5664   -1.6193    0.7921    1.3380   -0.2382   -5.8404    0.5431    4.3439
    2.0473    1.4337   -4.2378   -0.1438    1.1214   -1.7267    0.3722    5.0324    3.1198   -2.1847
    0.9192   -1.2328   -1.1997    1.5454   -3.2731    9.7619    0.1067    0.8386   -0.4118   -0.5895
```

b_2

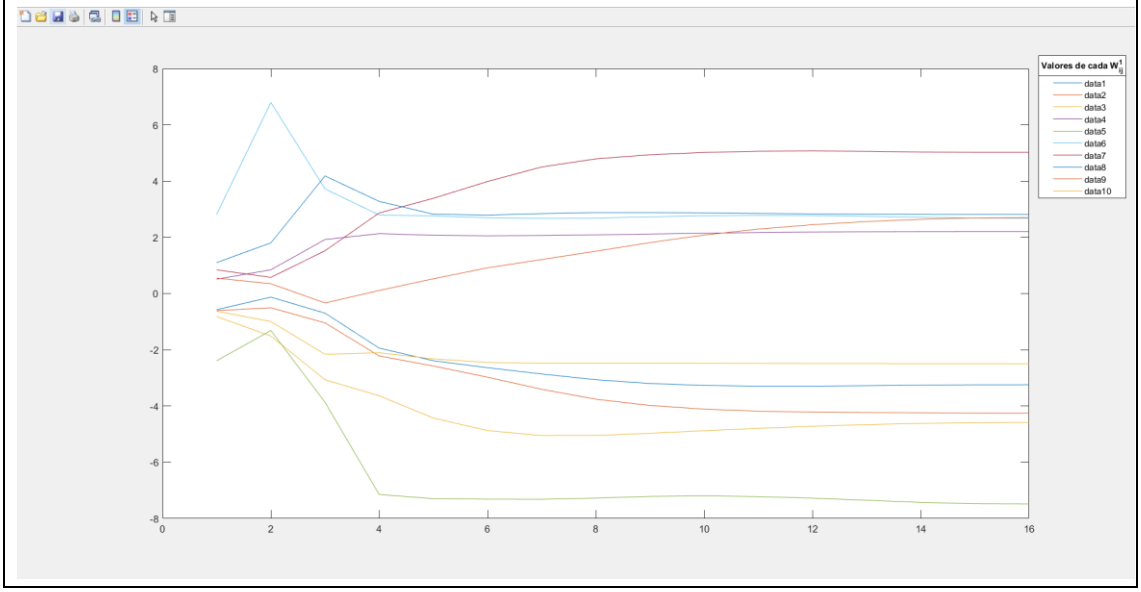
-1.7497
0.0261
-1.0659
0.5085
1.5840
0.3030
-0.3916
-0.3805
0.7563
-0.5074

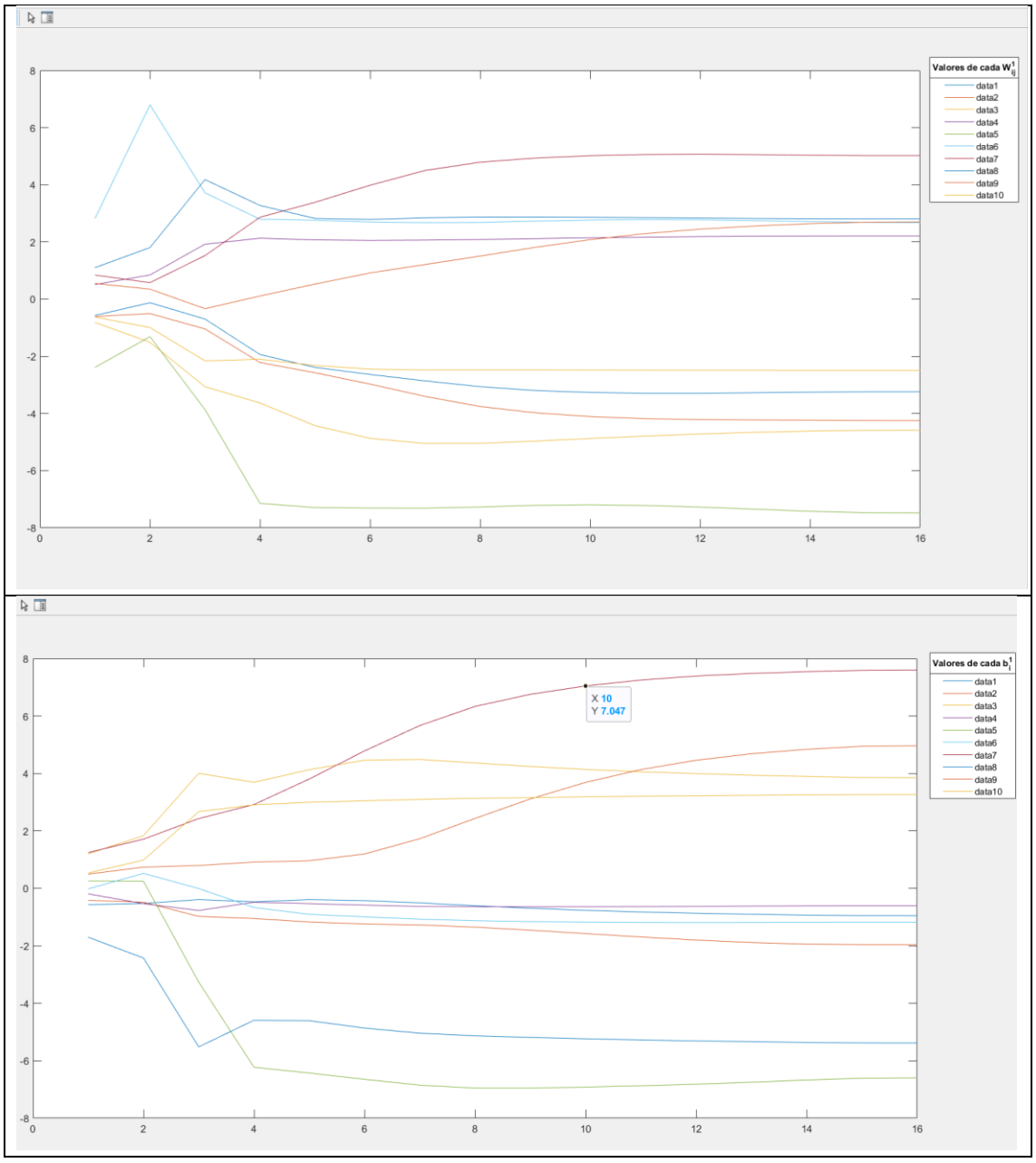
Gráficas:

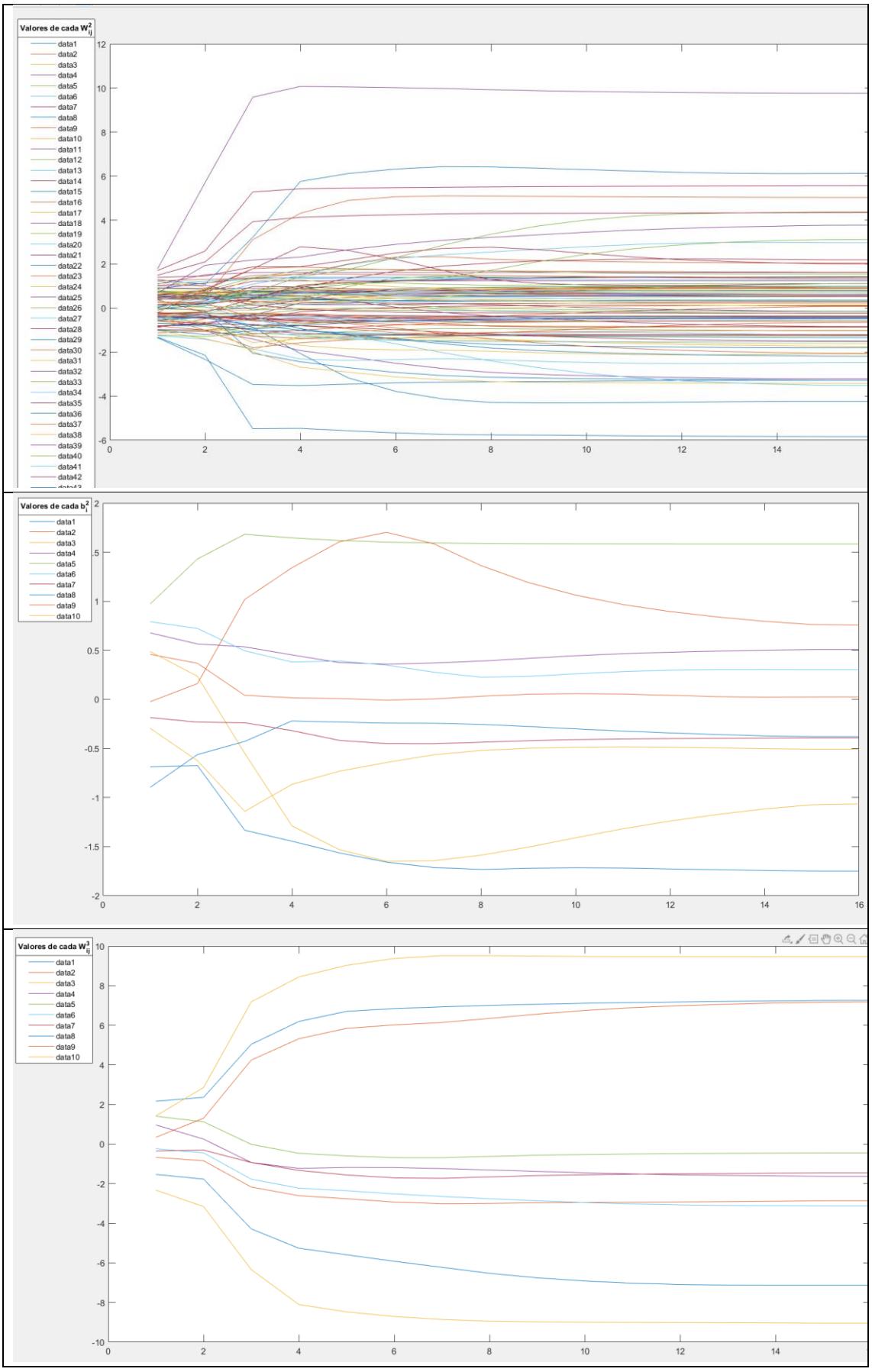
Gráfica del error

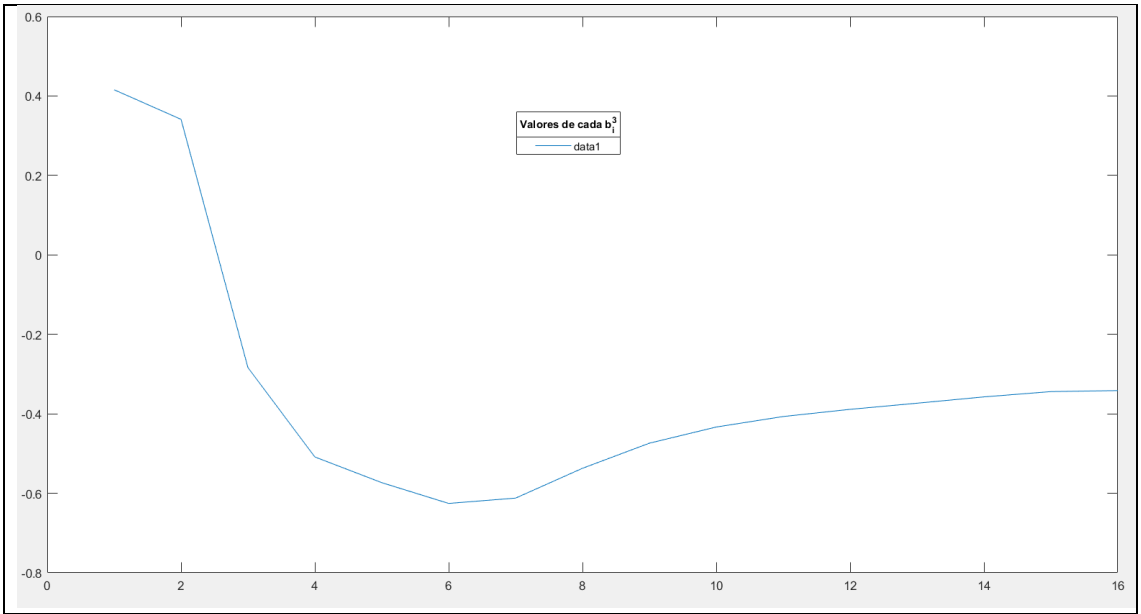


Graficas de W y b




Valores de cada W_{ij}^1
data1
data2
data3
data4
data5
data6
data7
data8
data9
data10
Valores de cada b_i^1
X 10
Y 7.047


Valores de cada W_{ij}^2
data1
data2
data3
data4
data5
data6
data7
data8
data9
data10
data11
data12
data13
data14
data15
data16
data17
data18
data19
data20
data21
data22
data23
data24
data25
data26
data27
data28
data29
data30
data31
data32
data33
data34
data35
data36
data37
data38
data39
data40
data41
data42
Valores de cada b_i^2
data1
data2
data3
data4
data5
data6
data7
data8
data9
data10
Valores de cada W_{ij}^3
data1
data2
data3
data4
data5
data6
data7
data8
data9
data10



Salida de la Red Xtrain, Xtest

Segunda parte:

Discusión

Hasta el momento nuestra red neuronal ha sido un reto no solo en el lado de la programación sino también en el lado del análisis y muestra de resultados ya que se tiene que probar una serie de valores tanto de inicialización como de los vectores v1 y v2 para definir la arquitectura. De igual usar más neuronas y capas tiene un costo de computación bastante elevado por lo cual hay que ir buscando cuál es la más optima de las soluciones.

Complementando lo antes definido es importante analizar e implementar mecanismos de validación supervisión del entrenamiento ya que nos ahorrará mucho tiempo buscando errores de arquitectura o de implementación.

Conclusiones:

Hasta este punto del desarrollo no se puede dar una conclusión concreta de los resultados de nuestro MLP ya que su funcionamiento ha sido limitado a la entrada de un seno el cuál nuestra red ha podido imitar en comportamiento de las entradas contra las salidas, pero en tanto en aspectos técnicos y de desarrollo he quedado bastante satisfecho debido a que he entendido el total

funcionamiento de la red perceptrón multicapa que a su vez es bastante bueno porque me ayudará a facilitar mi entendimiento en otras redes neuronales.

Pienso que es el primer paso solido en las redes neuronales que me abrirá un sinfín de posibilidades en un futuro siendo consciente de las buenas prácticas de programación que se involucran en la implementación de redes neuronales evaluando los diversos costos de computación contra las arquitecturas que se pueden utilizar

Bibliografía

Hagan, M. T. (s.f.). Neural Network Design. En M. T. Hagan, *Neural Network Design* (pág. 1010).