Práctica #5 Perceptrón Multicapa

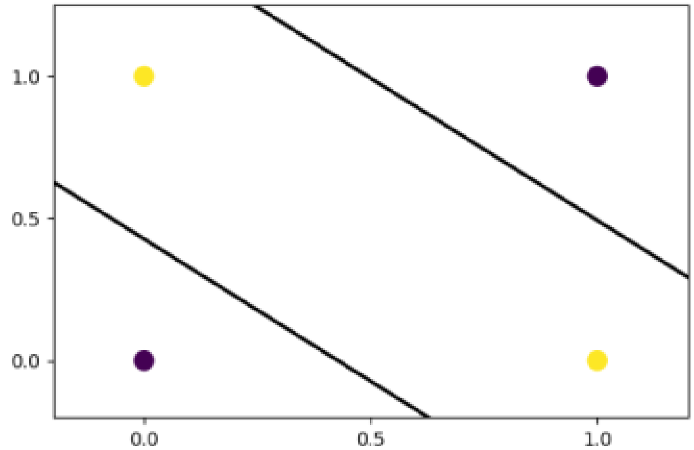
Jorge Gómez Reus

Índice

1. Introducción

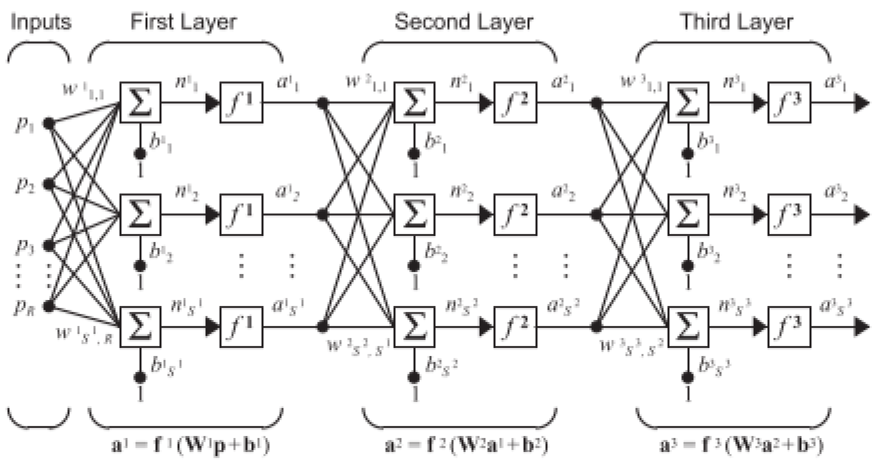
Un perceptrón multicapa es tipo de red neuronal artifical compuesta de varias neuronas y varias capas, puede resolver problemas que un perceptrón simple puede. El clásico ejemplo es el de la compuerta XOR:
En este caso se necesitan dos fronteras de desición, por ende dos neuron y se necesita otra capa para “combinar los resultados”. La manera la cual se actualizan los pesos y bías del MLP es usando un algoritmo para propagar los resultados a las neuronas de la capa actual y de las anteriores, es algoritmo es llamado backpropagation, el cual es un algoritmo de minización basado en el descenso en gradiente el cual encuentra los mínimos locales de una función. En esta práctica se usan MLP’s para aproximar señales leyendo datos de archivos de texto.

Figura 1: Compuerta XOR



1.1. Modelo

Figura 2: Modelo



1.1.1. Foward Propagation

$$a^0 = p$$

$$a^{m+1} = f^{m+1}(W^{m+1} \cdot a^m + b^{m+1}), \text{m} = 0, 1, 2, 3, \ldots, M - 1$$

$$a = a^M$$

donde M es el número de capas.

1.1.2. Foward Propagation

Para poder usar backpropagation se necesita que las funciones de activación en cada capa sean continuas y derivables. El algoritmo es el siguiente:

1. Calcular las sensitividades de cada capa desde la última capa hasta la primera:

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

$$s^m = \dot{F}^m(n^m)(W^{m+1})^T s^{m+1}$$

2. Actualizar los pesos y bias:

$$w^m(k+1) = W^m(k) - \alpha s^m(a^m - 1)^T$$

$$b^m(k+1) = b^m(k) - \alpha s^m$$

2. Diagrama de Flujo

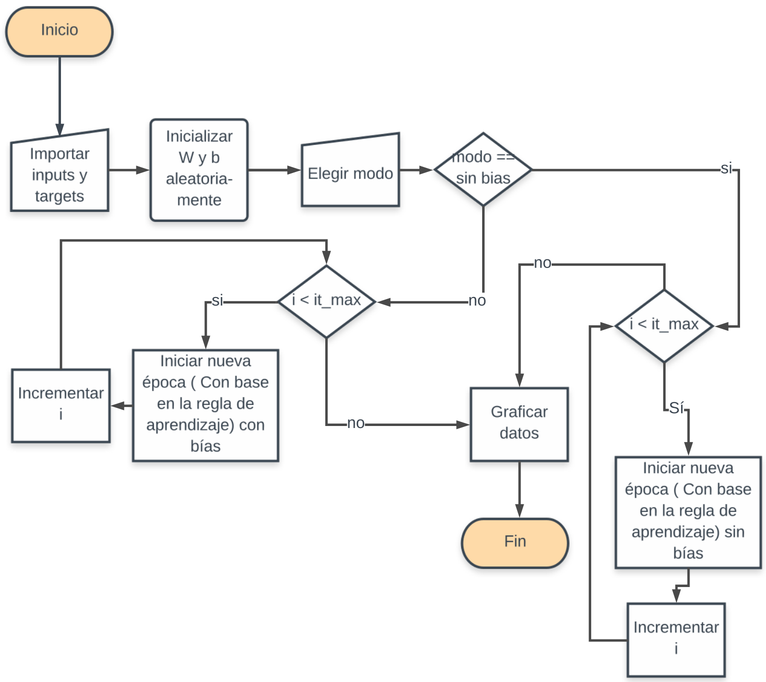


Figura 3: Diagrama de Flujo

3. Resultados

3.1. Polinomio 1

3.2. Inputs

1. -2.0000
2. -1.9600
3. -1.9200
4. -1.8800
5. -1.8400
6. -1.8000
7. -1.7600
8. -1.7200
9. -1.6800
10. -1.6400
11. -1.6000
12. -1.5600
13. -1.5200
14. -1.4800
15. -1.4400
16. -1.4000
17. -1.3600
18. -1.3200
19. -1.2800
20. -1.2400
21. -1.2000
22. -1.1600
23. -1.1200
24. -1.0800
25. -1.0400
26. -1.0000
27. -0.9600
28. -0.9200
29. -0.8800
30. -0.8400
31. -0.8000
32. -0.7600
33. -0.7200
34. -0.6800
35. -0.6400
36. -0.6000
37. -0.5600
38. -0.5200
39. -0.4800
40. -0.4400
41. -0.4000
42. -0.3600
43. -0.3200
44. -0.2800
45. -0.2400
46. -0.2000
47. -0.1600
48. -0.1200
49. -0.0800
50. -0.0400
51. 0
52. 0.0400
53. 0.0800
54. 0.1200
55. 0.1600
56. 0.2000
57. 0.2400
58. 0.2800
59. 0.3200
60. 0.3600
61. 0.4000
62. 0.4400
63. 0.4800
64. 0.5200
65. 0.5600
66. 0.6000
67. 0.6400
68. 0.6800
69. 0.7200
70. 0.7600
71. 0.8000
72. 0.8400
73. 0.8800
74. 0.9200
75. 0.9600
76. 1.0000
77. 1.0400
78. 1.0800
79. 1.1200
80. 1.1600
81. 1.2000
82. 1.2400
83. 1.2800
84. 1.3200
85. 1.3600
86. 1.4000
87. 1.4400
88. 1.4800
89. 1.5200
90. 1.5600
91. 1.6000
92. 1.6400
93. 1.6800
94. 1.7200
95. 1.7600
96. 1.8000
97. 1.8400
98. 1.8800
99. 1.9200
100. 1.9600
101. 2.0000

3.3. Targets

1. 1.0000
2. 1.2487
3. 1.4818
4. 1.6845
5. 1.8443
6. 1.9511
7. 1.9980
8. 1.9823
9. 1.9048
10. 1.7705
11. 1.5878
12. 1.3681
13. 1.1253
14. 0.8747
15. 0.6319
16. 0.4122
17. 0.2295
18. 0.0952
19. 0.0177
20. 0.0020
21. 0.0489
22. 0.1557
23. 0.3155
24. 0.5182
25. 0.7513
26. 1.0000
27. 1.2487
28. 1.4818
29. 1.6845
30. 1.8443
31. 1.9511
32. 1.9980
33. 1.9823
34. 1.9048
35. 1.7705
36. 1.5878
37. 1.3681
38. 1.1253
39. 0.8747
40. 0.6319
41. 0.4122
42. 0.2295
43. 0.0952
44. 0.0177
45. 0.0020
46. 0.0489
47. 0.1557
48. 0.3155
49. 0.5182
50. 0.7513
51. 1.0000
52. 1.2487
53. 1.4818
54. 1.6845
55. 1.8443
56. 1.9511
57. 1.9980
58. 1.9823
59. 1.9048
60. 1.7705
61. 1.5878
62. 1.3681
63. 1.1253
64. 0.8747
65. 0.6319
66. 0.4122
67. 0.2295
68. 0.0952
69. 0.0177
70. 0.0020
71. 0.0489
72. 0.1557
73. 0.3155
74. 0.5182
75. 0.7513
76. 1.0000
77. 1.2487
78. 1.4818
79. 1.6845
80. 1.8443
81. 1.9511
82. 1.9980
83. 1.9823
84. 1.9048
85. 1.7705
86. 1.5878
87. 1.3681
88. 1.1253
89. 0.8747
90. 0.6319
91. 0.4122
92. 0.2295
93. 0.0952
94. 0.0177
95. 0.0020
96. 0.0489
97. 0.1557
98. 0.3155
99. 0.5182
100. 0.7513
101. 1.0000

3.3.1. Datos

$$V1 = \begin{bmatrix}1116101\end{bmatrix}$$

$$V2 = \begin{bmatrix}321\end{bmatrix}$$

epochmax = 10000

Múltiplo para las épocas de validación = 500
numval = 7
alpha=.0701
error de validación = .0000000000000001
Configuración: 80-15-15

3.4. Resultado

3.5. Imágenes

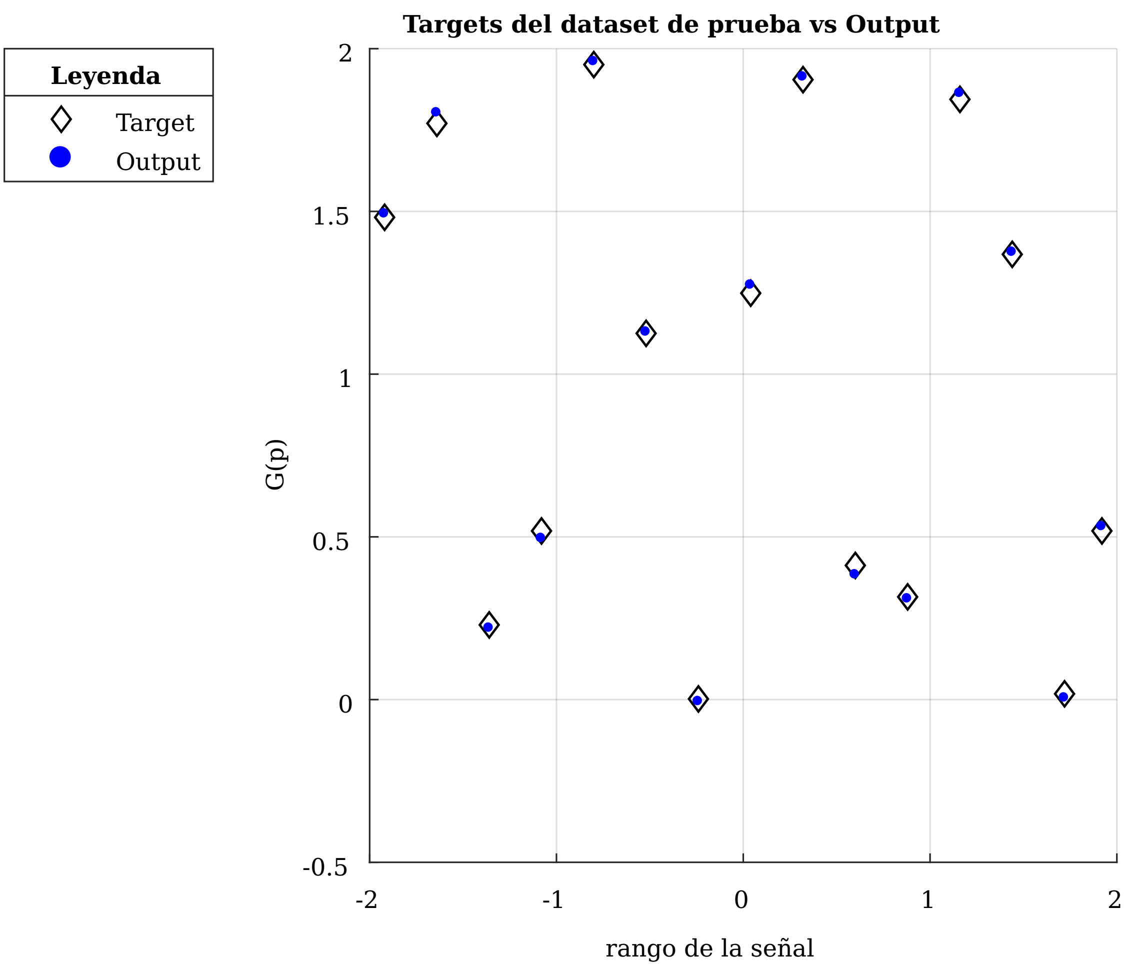


Figura 4: Gráfica 1.1

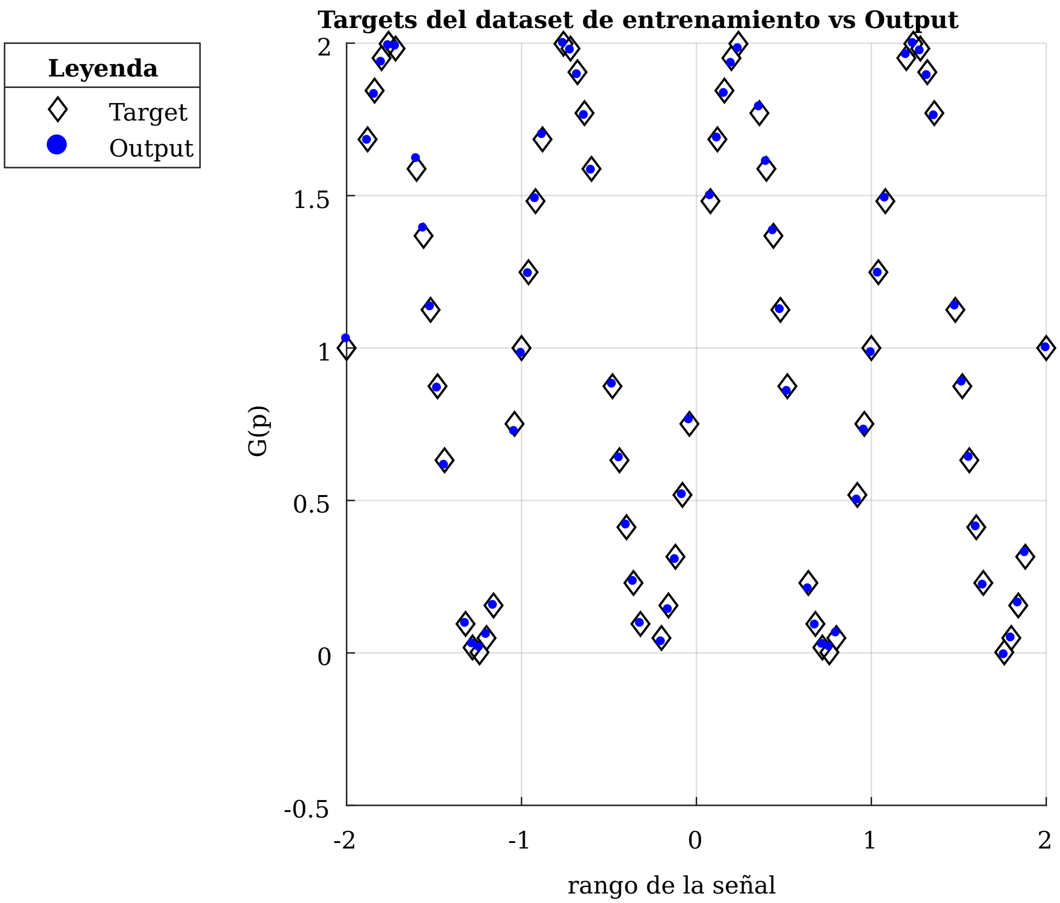


Figura 5: Gráfica 1.2

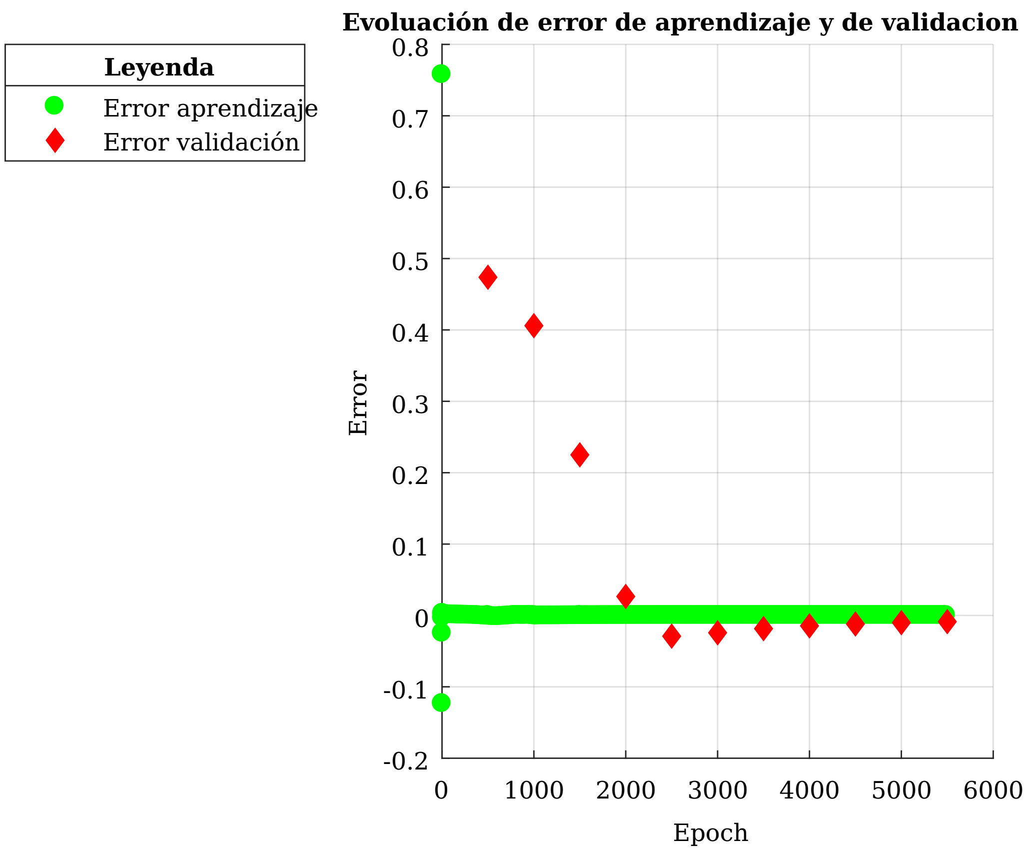


Figura 6: Gráfica 1.3

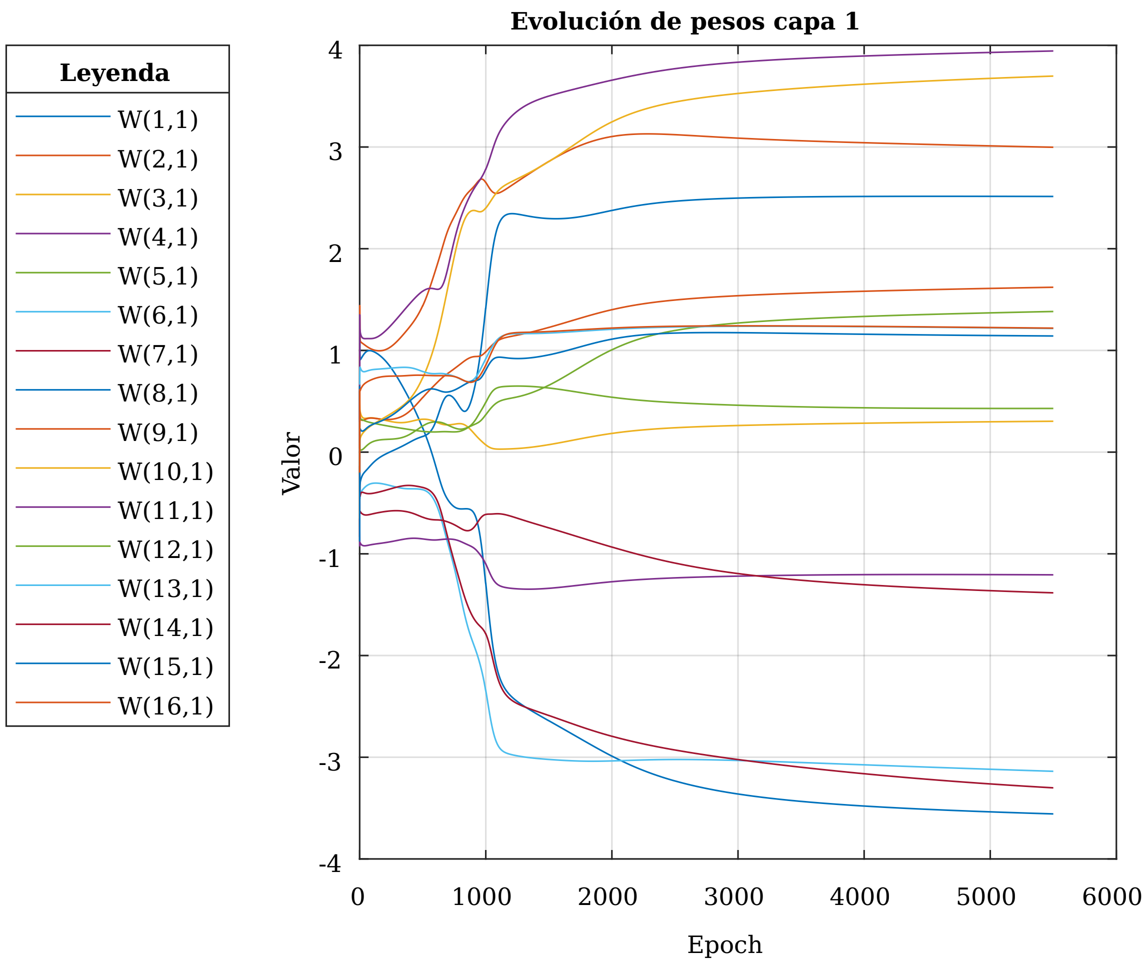


Figura 7: Gráfica 1.4

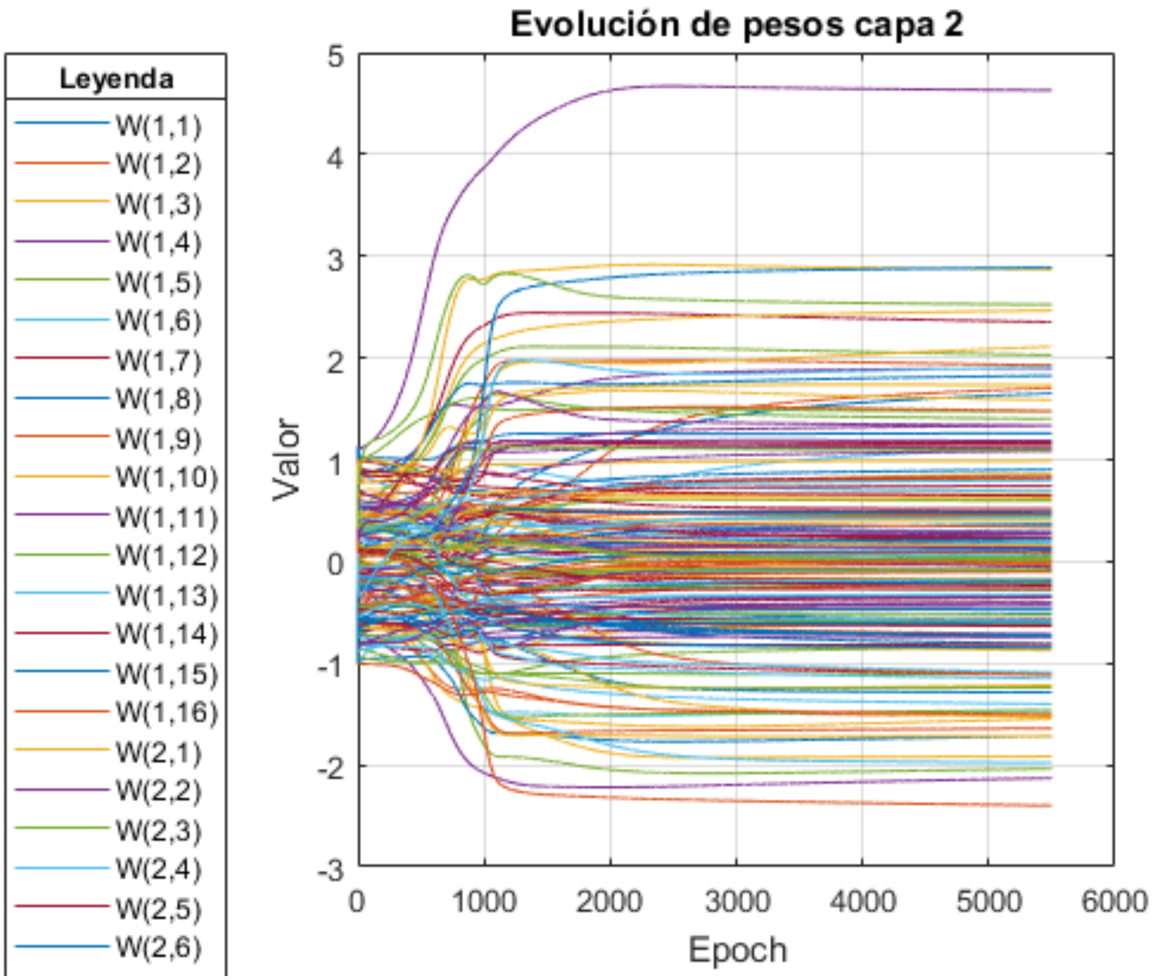


Figura 8: Gráfica 1.5

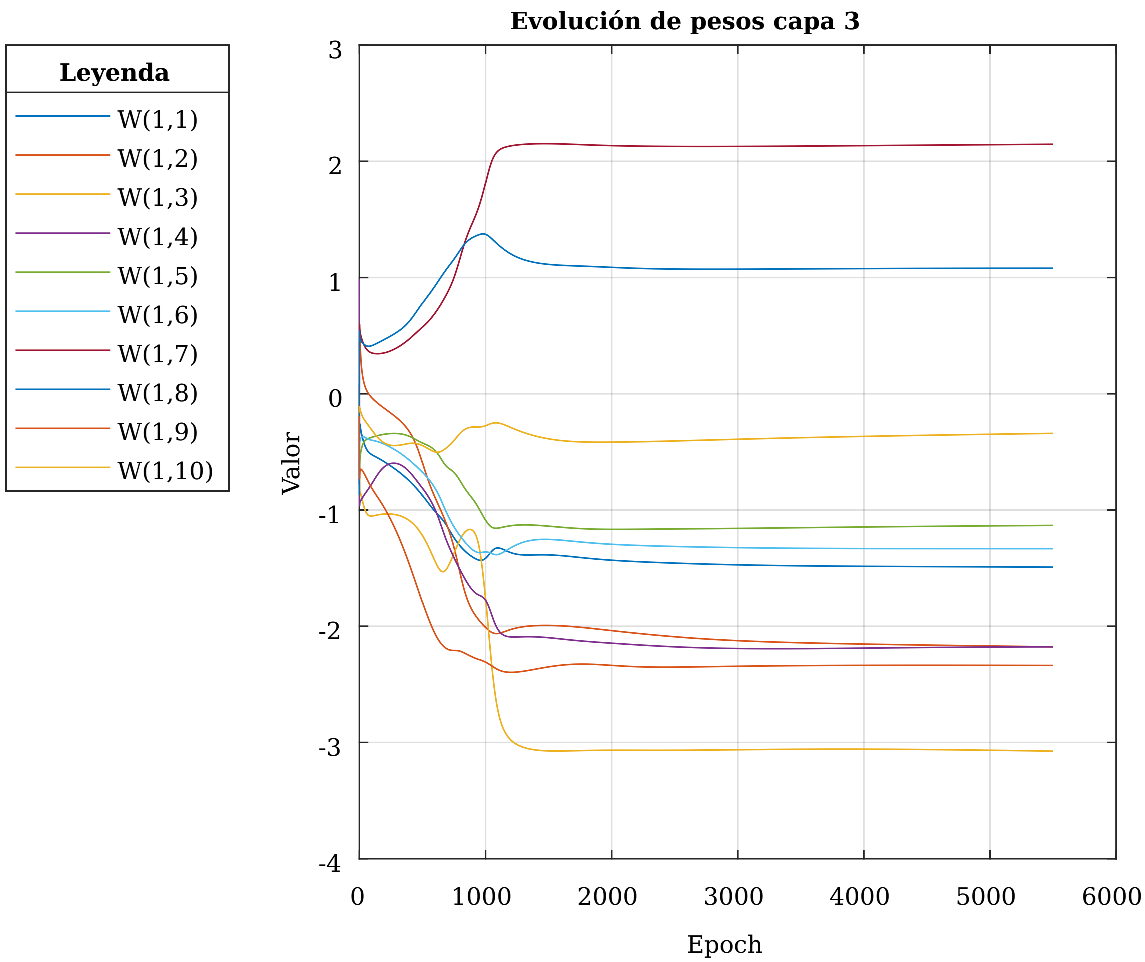


Figura 9: Gráfica 1.6

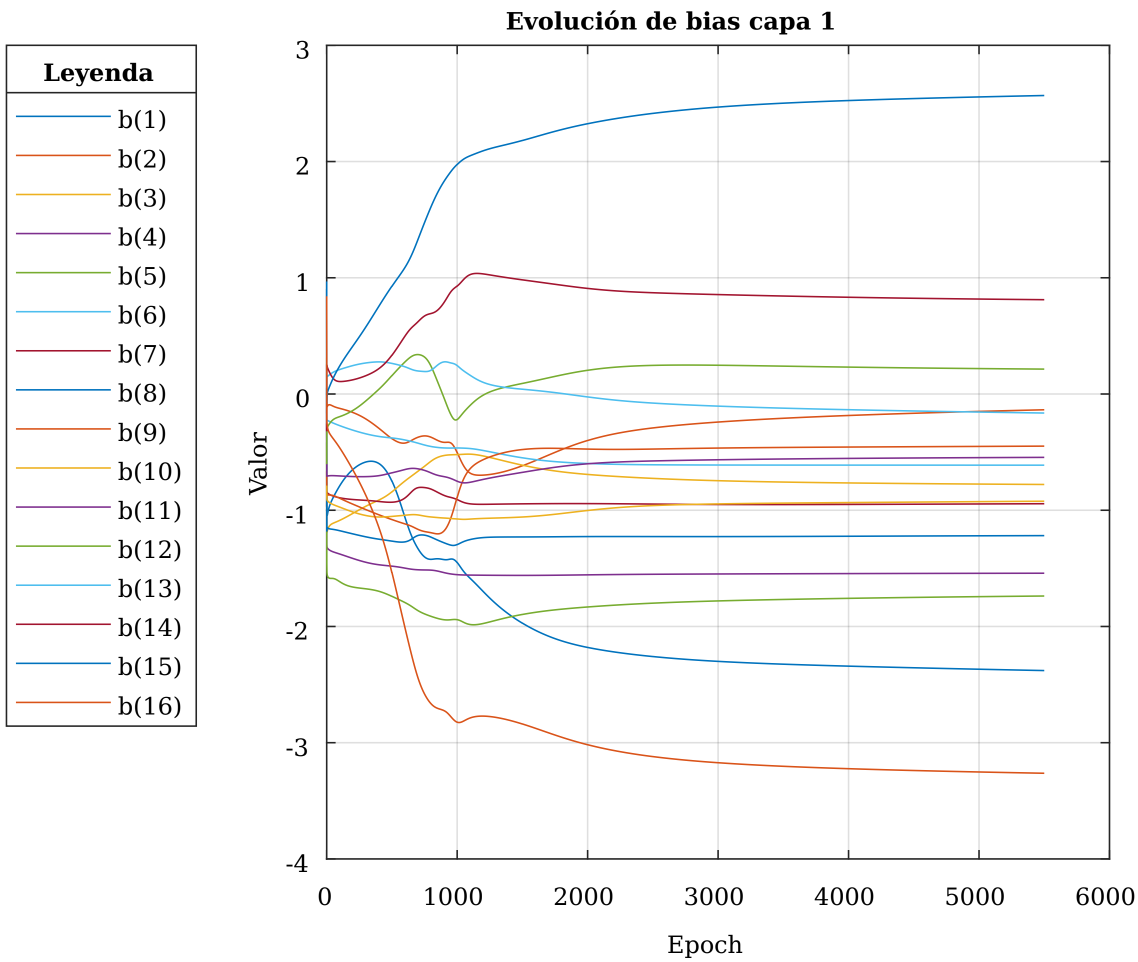


Figura 10: Gráfica 1.7

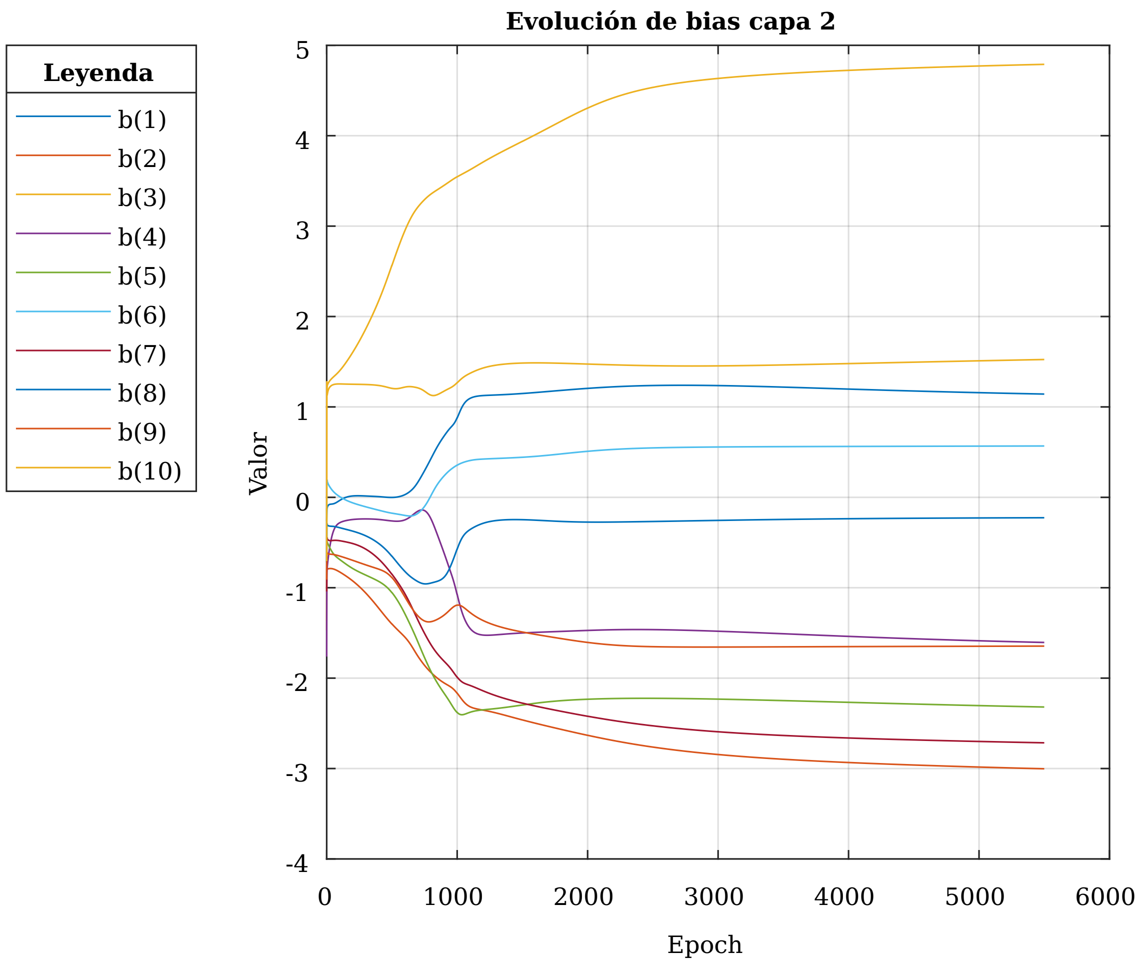


Figura 11: Gráfica 1.8

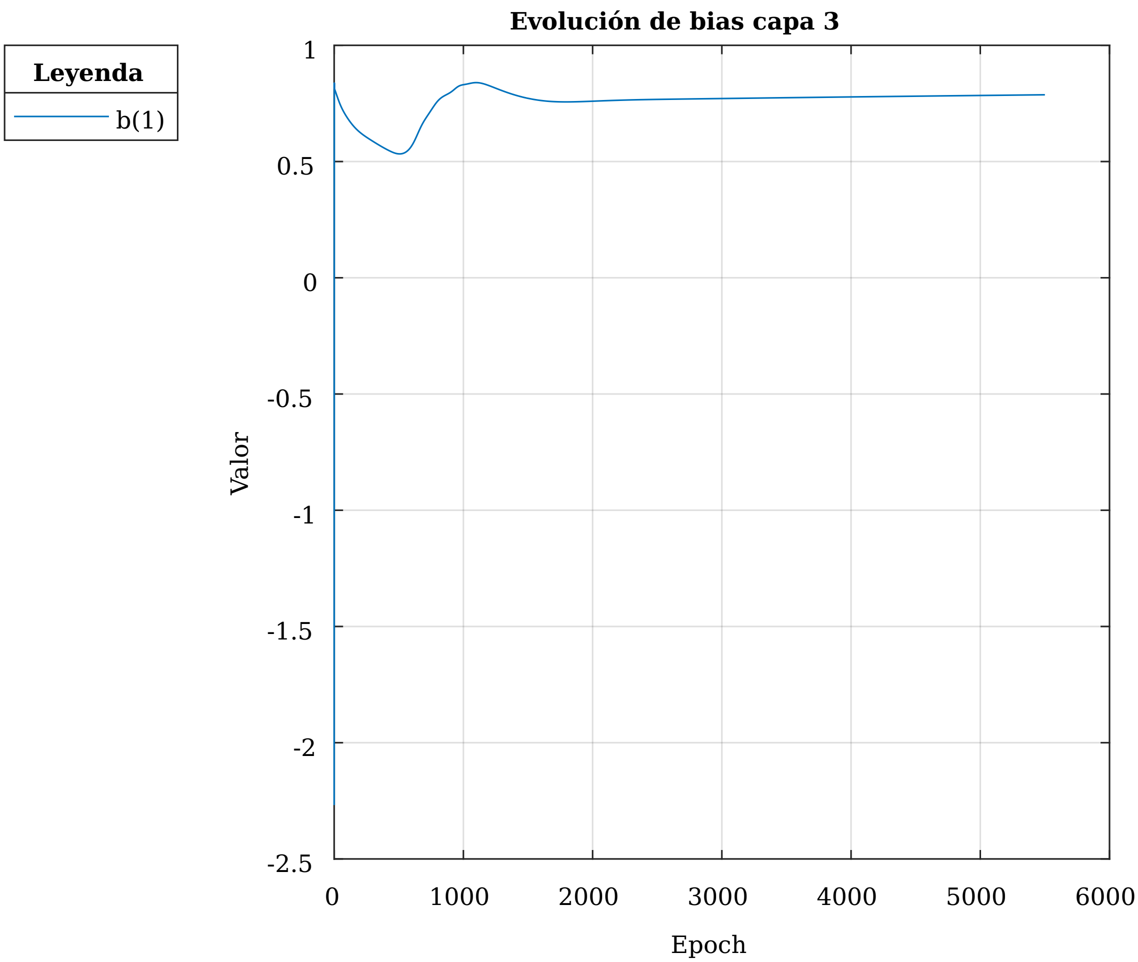


Figura 12: Gráfica 1.9

3.6. Polinomio 2

3.7. Inputs

1. 4.0000
2. 3.9667
3. 3.9333
4. 3.9000
5. 3.8667
6. 3.8333
7. 3.8000
8. 3.7667
9. 3.7333
10. 3.7000
11. 3.6667
12. 3.6333
13. 3.6000
14. 3.5667
15. 3.5333
16. 3.5000
17. 3.4667
18. 3.4333
19. 3.4000
20. 3.3667
21. 3.3333
22. 3.3000
23. 3.2667
24. 3.2333
25. 3.2000
26. 3.1667
27. 3.1333
28. 3.1000
29. 3.0667
30. 3.0333
31. 3.0000
32. 2.9667
33. 2.9333
34. 2.9000
35. 2.8667
36. 2.8333
37. 2.8000
38. 2.7667
39. 2.7333
40. 2.7000
41. 2.6667
42. 2.6333
43. 2.6000
44. 2.5667
45. 2.5333
46. 2.5000
47. 2.4667
48. 2.4333
49. 2.4000
50. 2.3667
51. 2.3333
52. 2.3000
53. 2.2667
54. 2.2333
55. 2.2000
56. 2.1667
57. 2.1333
58. 2.1000
59. 2.0667
60. 2.0333
61. 2.0000
62. 1.9667
63. 1.9333
64. 1.9000
65. 1.8667
66. 1.8333
67. 1.8000
68. 1.7667
69. 1.7333
70. 1.7000
71. 1.6667
72. 1.6333
73. 1.6000
74. 1.5667
75. 1.5333
76. 1.5000
77. 1.4667
78. 1.4333
79. 1.4000
80. 1.3667
81. 1.3333
82. 1.3000
83. 1.2667
84. 1.2333
85. 1.2000
86. 1.1667
87. 1.1333
88. 1.1000
89. 1.0667
90. 1.0333
91. 1.0000
92. 0.9667
93. 0.9333
94. 0.9000
95. 0.8667
96. 0.8333
97. 0.8000
98. 0.7667
99. 0.7333
100. 0.7000
101. 0.6667
102. 0.6333
103. 0.6000
104. 0.5667
105. 0.5333
106. 0.5000
107. 0.4667
108. 0.4333
109. 0.4000
110. 0.3667
111. 0.3333
112. 0.3000
113. 0.2667
114. 0.2333
115. 0.2000
116. 0.1667

117. 0.1333
118. 0.1000
119. 0.0667
120. 0.0333
121. 0.0000
122. -0.0333
123. -0.0667
124. -0.1000
125. -0.1333
126. -0.1667
127. -0.2000
128. -0.2333
129. -0.2667
130. -0.3000
131. -0.3333
132. -0.3667
133. -0.4000
134. -0.4333
135. -0.4667
136. -0.5000
137. -0.5333
138. -0.5667
139. -0.6000
140. -0.6333
141. -0.6667
142. -0.7000
143. -0.7333
144. -0.7667
145. -0.8000
146. -0.8333
147. -0.8667
148. -0.9000
149. -0.9333
150. -0.9667
151. -1.0000
152. -1.0333
153. -1.0667
154. -1.1000
155. -1.1333
156. -1.1667
157. -1.2000
158. -1.2333
159. -1.2667
160. -1.3000
161. -1.3333
162. -1.3667
163. -1.4000
164. -1.4333
165. -1.4667
166. -1.5000
167. -1.5333
168. -1.5667
169. -1.6000
170. -1.6333
171. -1.6667
172. -1.7000
173. -1.7333
174. -1.7667
175. -1.8000
176. -1.8333
177. -1.8667
178. -1.9000
179. -1.9333
180. -1.9667
181. -2.0000
182. -2.0333
183. -2.0667
184. -2.1000
185. -2.1333
186. -2.1667
187. -2.2000
188. -2.2333
189. -2.2667
190. -2.3000
191. -2.3333
192. -2.3667
193. -2.4000
194. -2.4333
195. -2.4667
196. -2.5000
197. -2.5333
198. -2.5667
199. -2.6000
200. -2.6333
201. -2.6667
202. -2.7000
203. -2.7333
204. -2.7667
205. -2.8000
206. -2.8333
207. -2.8667
208. -2.9000
209. -2.9333
210. -2.9667
211. -3.0000
212. -3.0333
213. -3.0667
214. -3.1000
215. -3.1333
216. -3.1667
217. -3.2000
218. -3.2333
219. -3.2667
220. -3.3000
221. -3.3333
222. -3.3667
223. -3.4000
224. -3.4333
225. -3.4667
226. -3.5000
227. -3.5333
228. -3.5667
229. -3.6000
230. -3.6333
231. -3.6667
232. -3.7000
233. -3.7333
234. -3.7667
235. -3.8000
236. -3.8333
237. -3.8667
238. -3.9000
239. -3.9333
240. -3.9667
241. -4.0000

3.8. Targets

1. -0.01292
2. -0.01451
3. -0.01628
4. -0.01825
5. -0.02043
6. -0.02284
7. -0.02550
8. -0.02844
9. -0.03167
10. -0.03523
11. -0.03914
12. -0.04342
13. -0.04812
14. -0.05325
15. -0.05886
16. -0.06497
17. -0.07162
18. -0.07886
19. -0.08671
20. -0.09522
21. -0.10442
22. -0.11437
23. -0.12510
24. -0.13666
25. -0.14909
26. -0.16243
27. -0.17674
28. -0.19204
29. -0.20839
30. -0.22583
31. -0.24440
32. -0.26413
33. -0.28507
34. -0.30725
35. -0.33070
36. -0.35544
37. -0.38150
38. -0.40891
39. -0.43767
40. -0.46778
41. -0.49926
42. -0.53210
43. -0.56628
44. -0.60178
45. -0.63858
46. -0.67663
47. -0.71589
48. -0.75630
49. -0.79779
50. -0.84028
51. -0.88368
52. -0.92790
53. -0.97281
54. -1.01829
55. -1.06421
56. -1.11042
57. -1.15676
58. -1.20305
59. -1.24912
60. -1.29478
61. -1.33982
62. -1.38403
63. -1.42719
64. -1.46909
65. -1.50947
66. -1.54811
67. -1.58477
68. -1.61920
69. -1.65116
70. -1.68040
71. -1.70668
72. -1.72976
73. -1.74941
74. -1.76541
75. -1.77754
76. -1.78559
77. -1.78938
78. -1.78873
79. -1.78348
80. -1.77349
81. -1.75865
82. -1.73885
83. -1.71402
84. -1.68412
85. -1.64912
86. -1.60902
87. -1.56387
88. -1.51372
89. -1.45866
90. -1.39883
91. -1.33437
92. -1.26546
93. -1.19232
94. -1.11519
95. -1.03433
96. -0.95005
97. -0.86267
98. -0.77252
99. -0.67999
100. -0.58546
101. -0.48934
102. -0.39205
103. -0.29402
104. -0.19569
105. -0.09754
106. 0.00000
107. 0.09646
108. 0.19138
109. 0.28432
110. 0.37483
111. 0.46247
112. 0.54683
113. 0.62751
114. 0.70412
115. 0.77632
116. 0.84375
117. 0.90613
118. 0.96317
119. 1.01463
120. 1.06030
121. 1.10000
122. 1.13359
123. 1.16097

124. 1.18207
125. 1.19687
126. 1.20536
127. 1.20760
128. 1.20367
129. 1.19368
130. 1.17779
131. 1.15617
132. 1.12905
133. 1.09666
134. 1.05928
135. 1.01720
136. 0.97075
137. 0.92025
138. 0.86605
139. 0.80854
140. 0.74809
141. 0.68508
142. 0.61990
143. 0.55296
144. 0.48465
145. 0.41536
146. 0.34547
147. 0.27537
148. 0.20543
149. 0.13599
150. 0.06741
151. 0.00000
152. -0.06593
153. -0.13009
154. -0.19222
155. -0.25207
156. -0.30943
157. -0.36409
158. -0.41589
159. -0.46467
160. -0.51031
161. -0.55272
162. -0.59180
163. -0.62752
164. -0.65983
165. -0.68874
166. -0.71424
167. -0.73636
168. -0.75517
169. -0.77072
170. -0.78309
171. -0.79239
172. -0.79871
173. -0.80218
174. -0.80293
175. -0.80109
176. -0.79682
177. -0.79027
178. -0.78158
179. -0.77093
180. -0.75846
181. -0.74434
182. -0.72874
183. -0.71180
184. -0.69370
185. -0.67457
186. -0.65456
187. -0.63383
188. -0.61251
189. -0.59073
190. -0.56861
191. -0.54628
192. -0.52384
193. -0.50140
194. -0.47905
195. -0.45688
196. -0.43498
197. -0.41341
198. -0.39224
199. -0.37153
200. -0.35132
201. -0.33168
202. -0.31262
203. -0.29419
204. -0.27640
205. -0.25928
206. -0.24285
207. -0.22710
208. -0.21205
209. -0.19770
210. -0.18405
211. -0.17108
212. -0.15879
213. -0.14717
214. -0.13619
215. -0.12586
216. -0.11614
217. -0.10702
218. -0.09847
219. -0.09048
220. -0.08302
221. -0.07607
222. -0.06961
223. -0.06360
224. -0.05804
225. -0.05288
226. -0.04812
227. -0.04373
228. -0.03969
229. -0.03597
230. -0.03256
231. -0.02943
232. -0.02656
233. -0.02395
234. -0.02156
235. -0.01938
236. -0.01741
237. -0.01561
238. -0.01398
239. -0.01250
240. -0.01117
241. -0.00996

3.8.1. Datos

$$V1 = \begin{bmatrix} 1116101 \end{bmatrix}$$

$$V2 = \begin{bmatrix} 321 \end{bmatrix}$$

epochmax = 2200
alpha=.0103
Múltiplo para las épocas de validación = 500
numval = 7
error de validación = .000000000001
Configuración: 80-10-10

3.9. Resultado

3.10. Imágenes

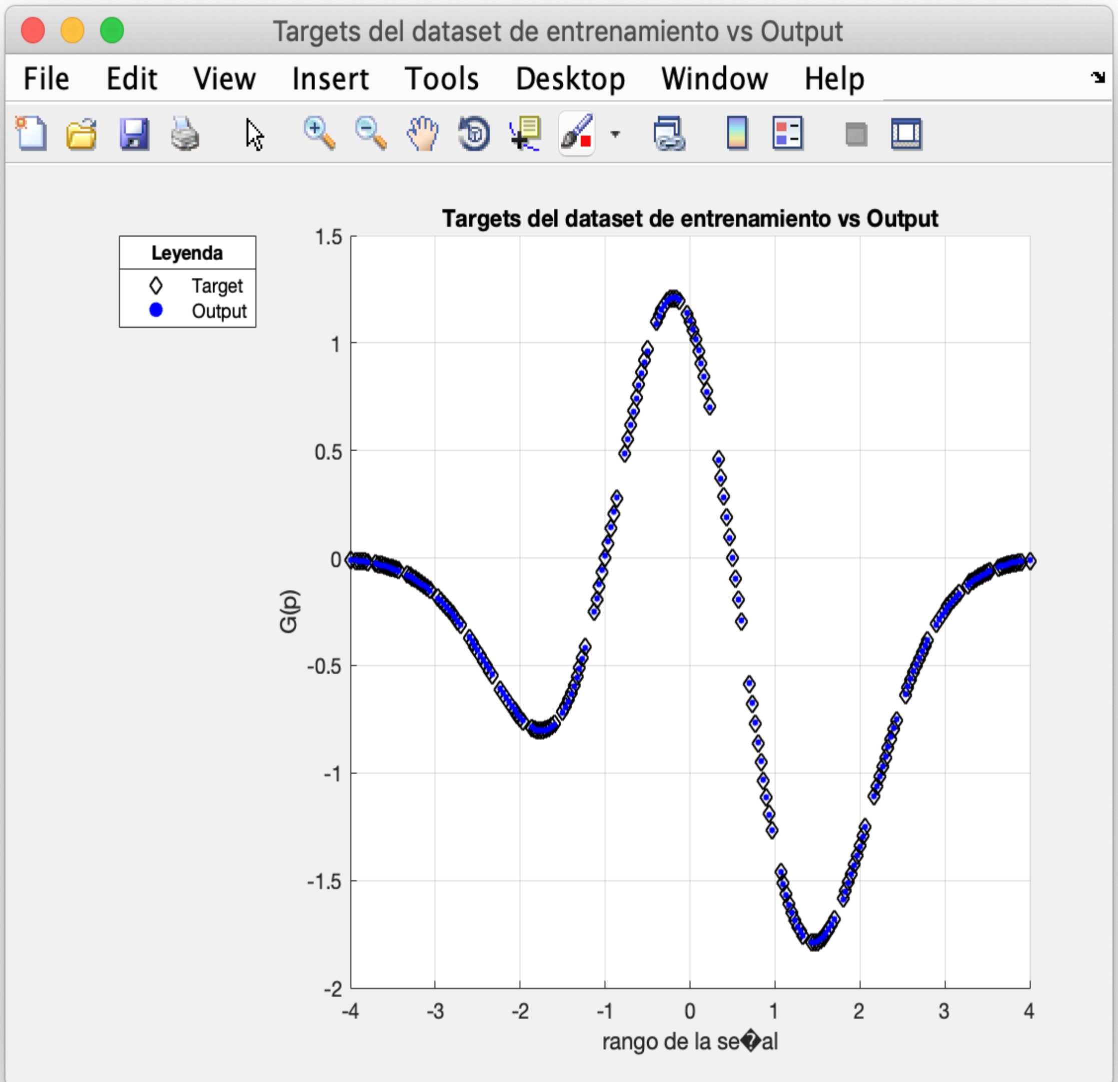


Figura 13: Gráfica 1.1

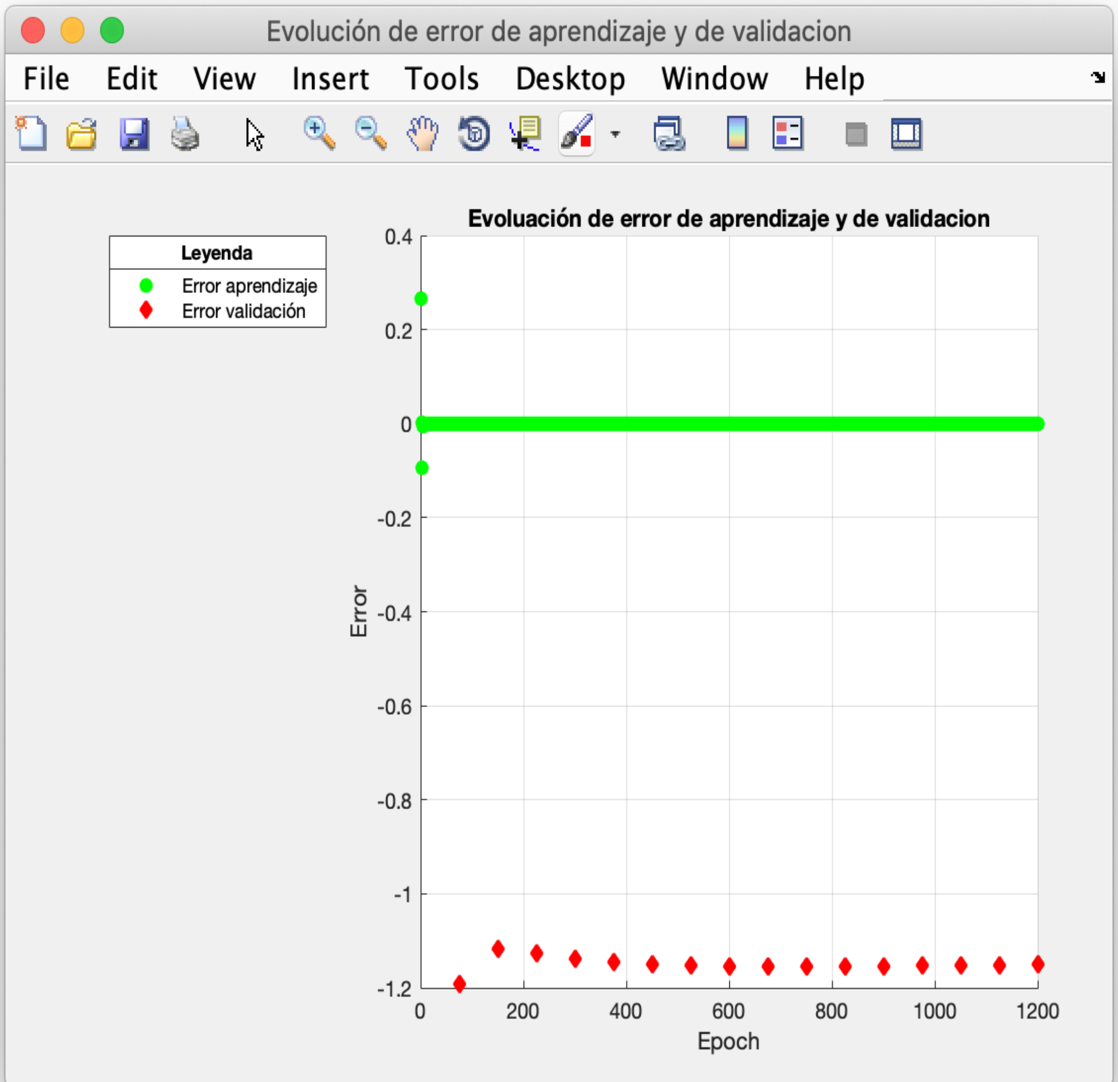


Figura 14: Gráfica 1.2

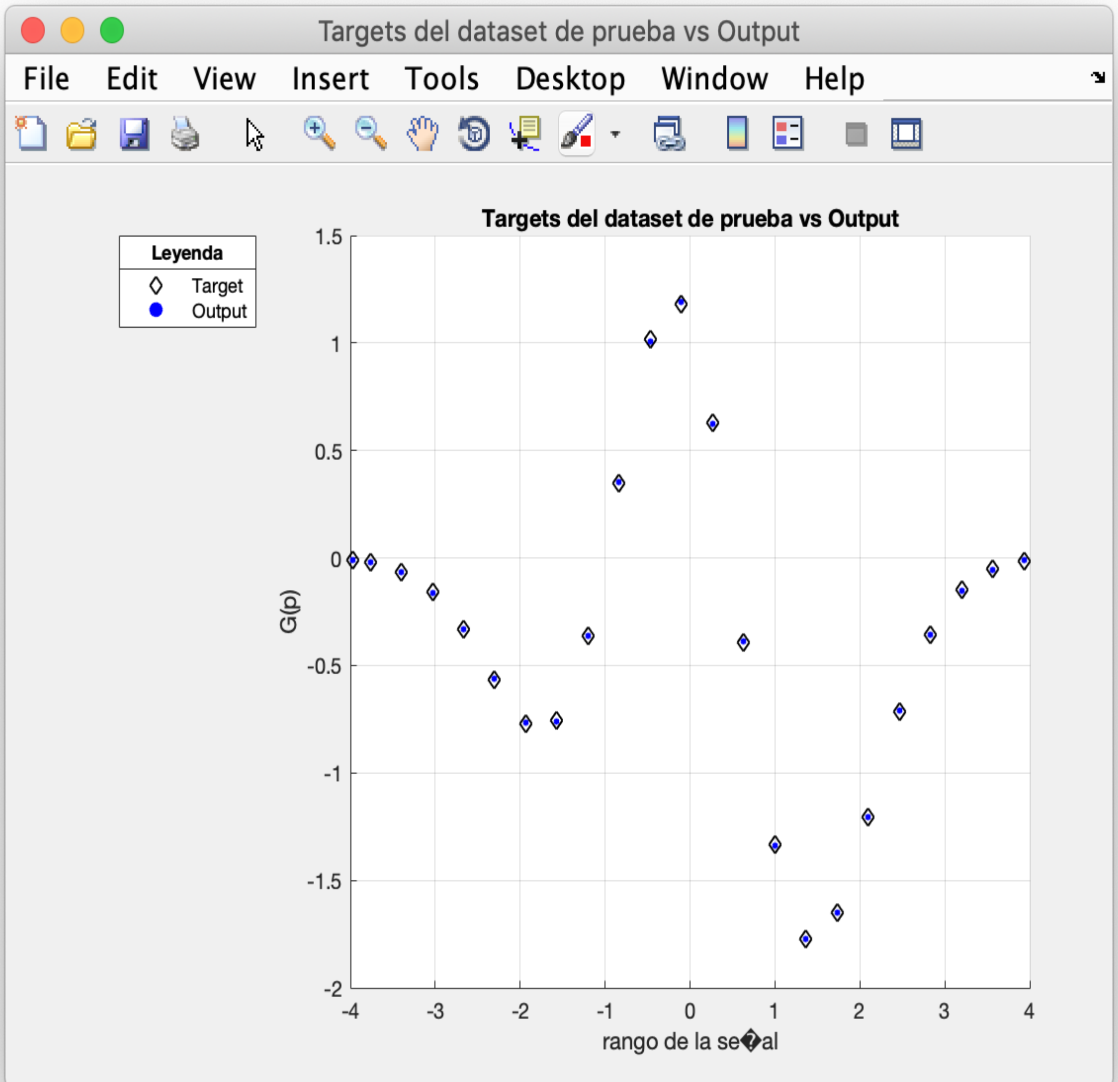


Figura 15: Gráfica 1.3

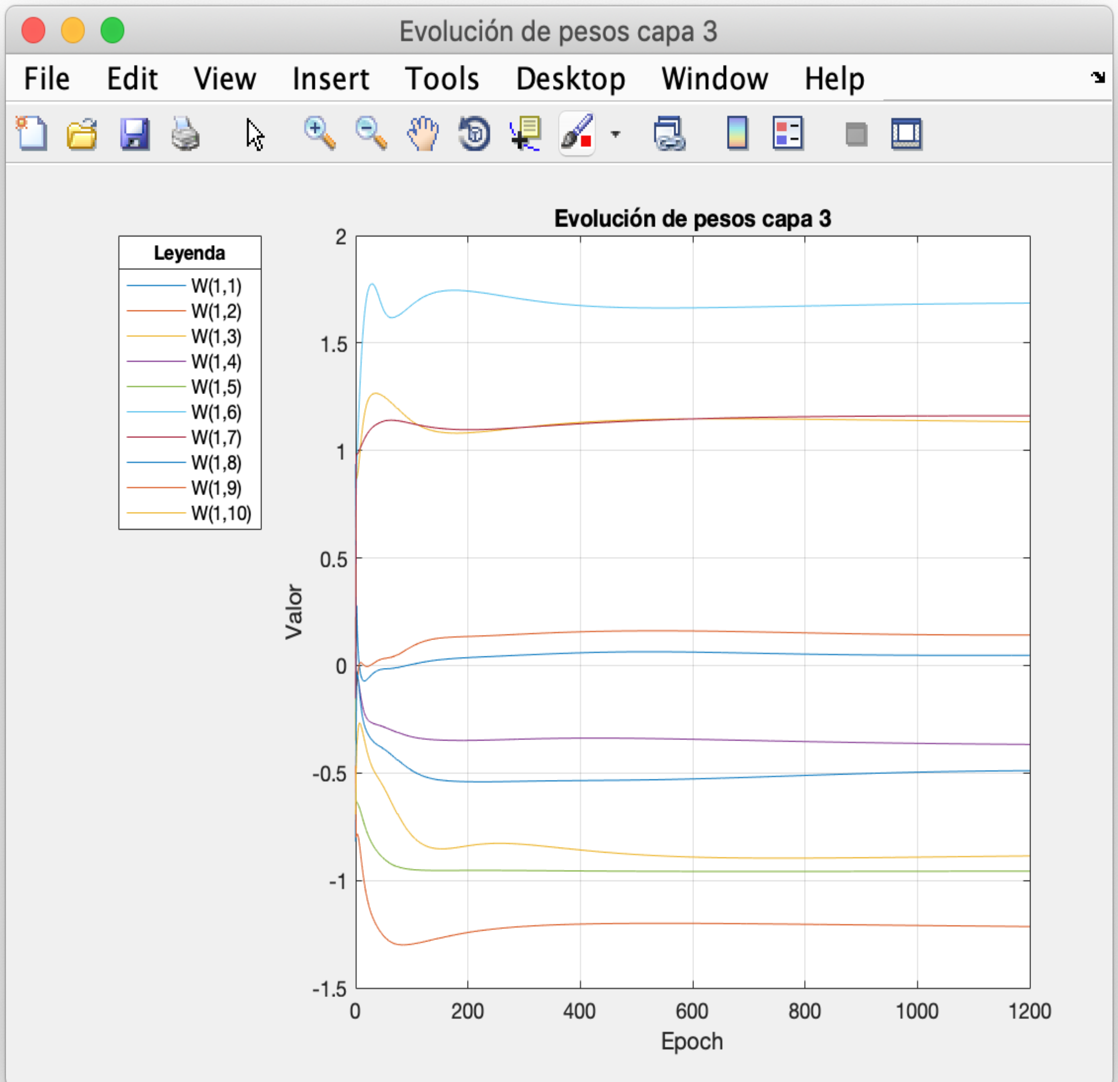


Figura 16: Gráfica 1.4

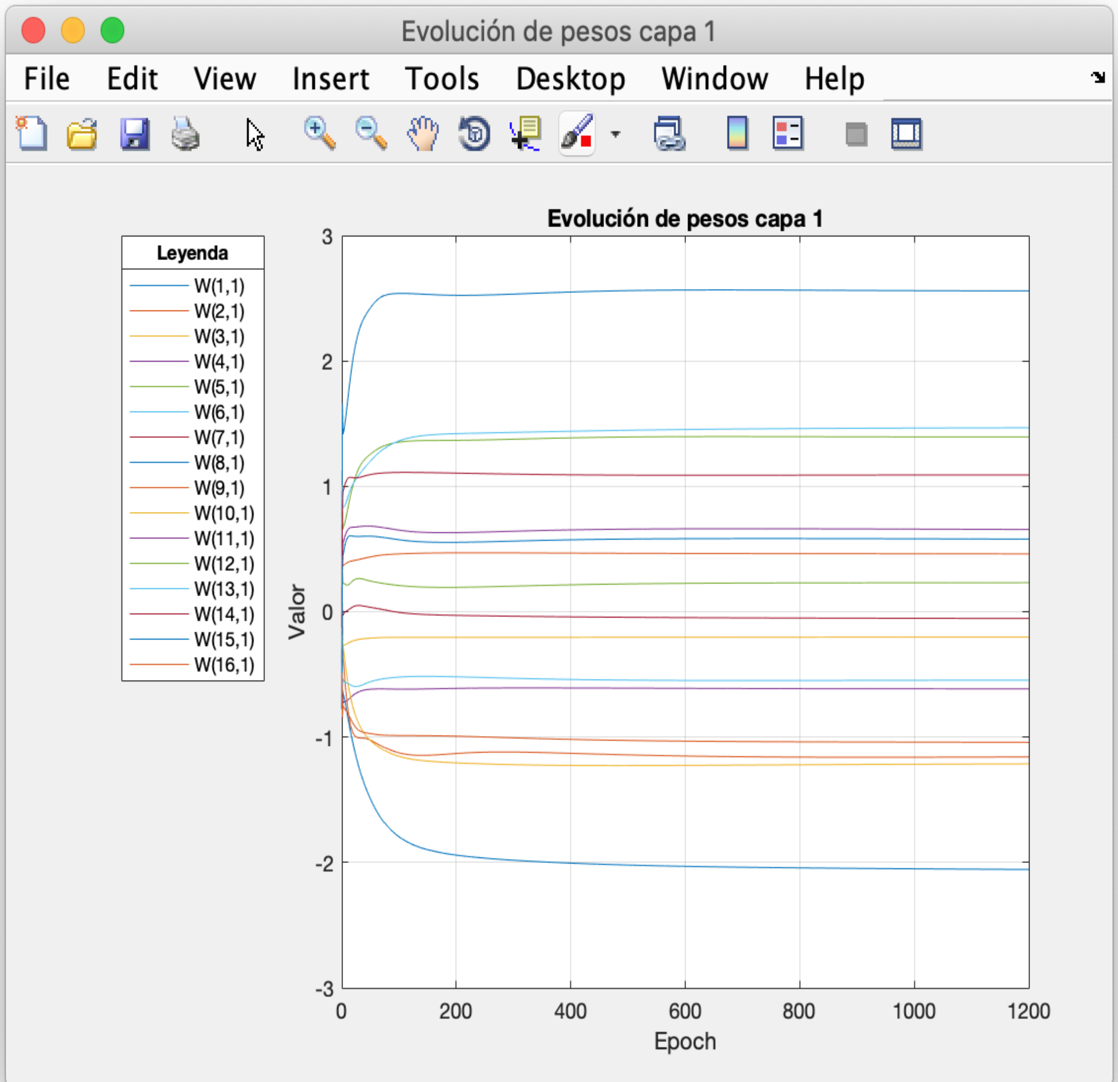


Figura 17: Gráfica 1.5

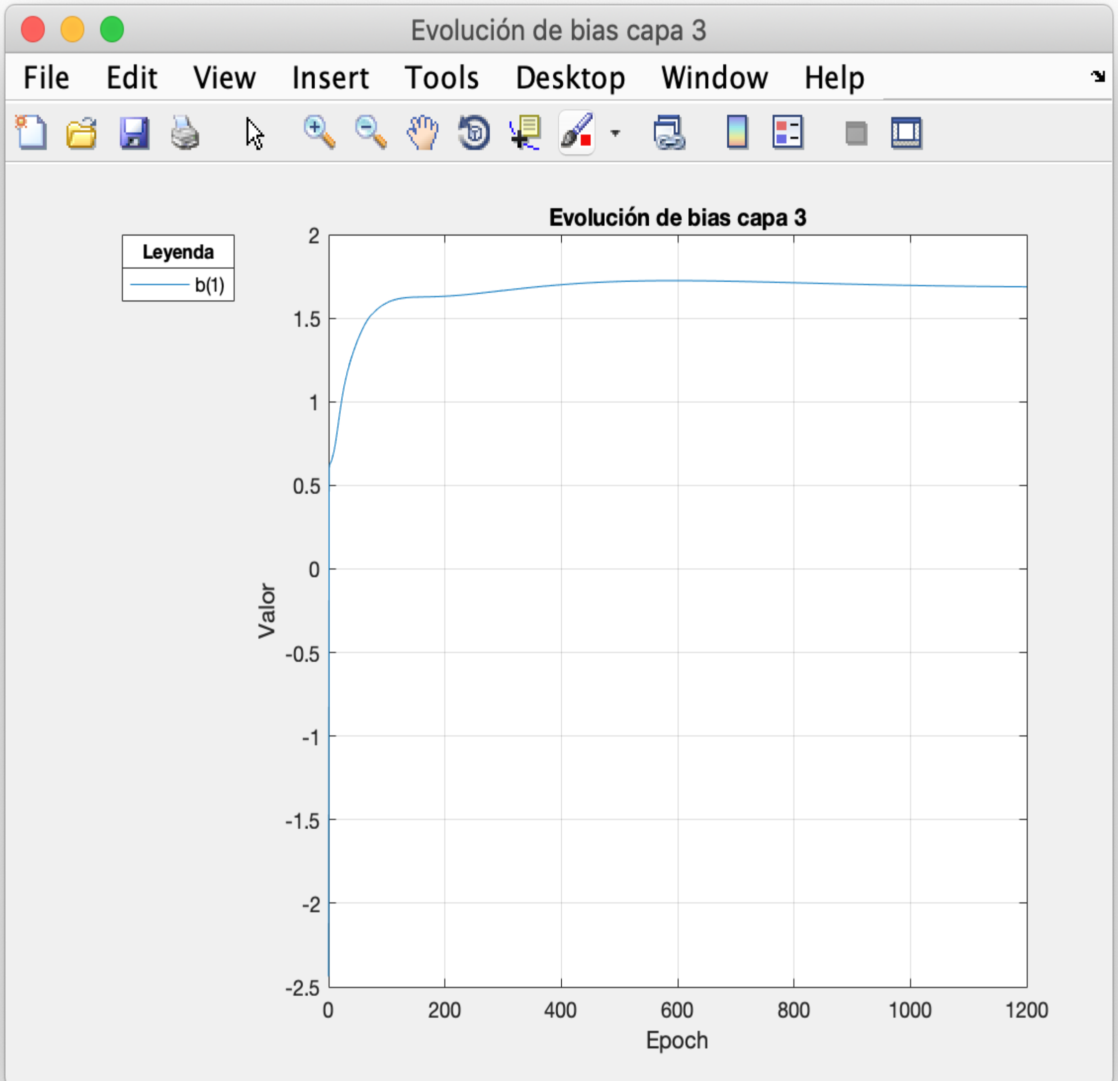


Figura 18: Gráfica 1.6

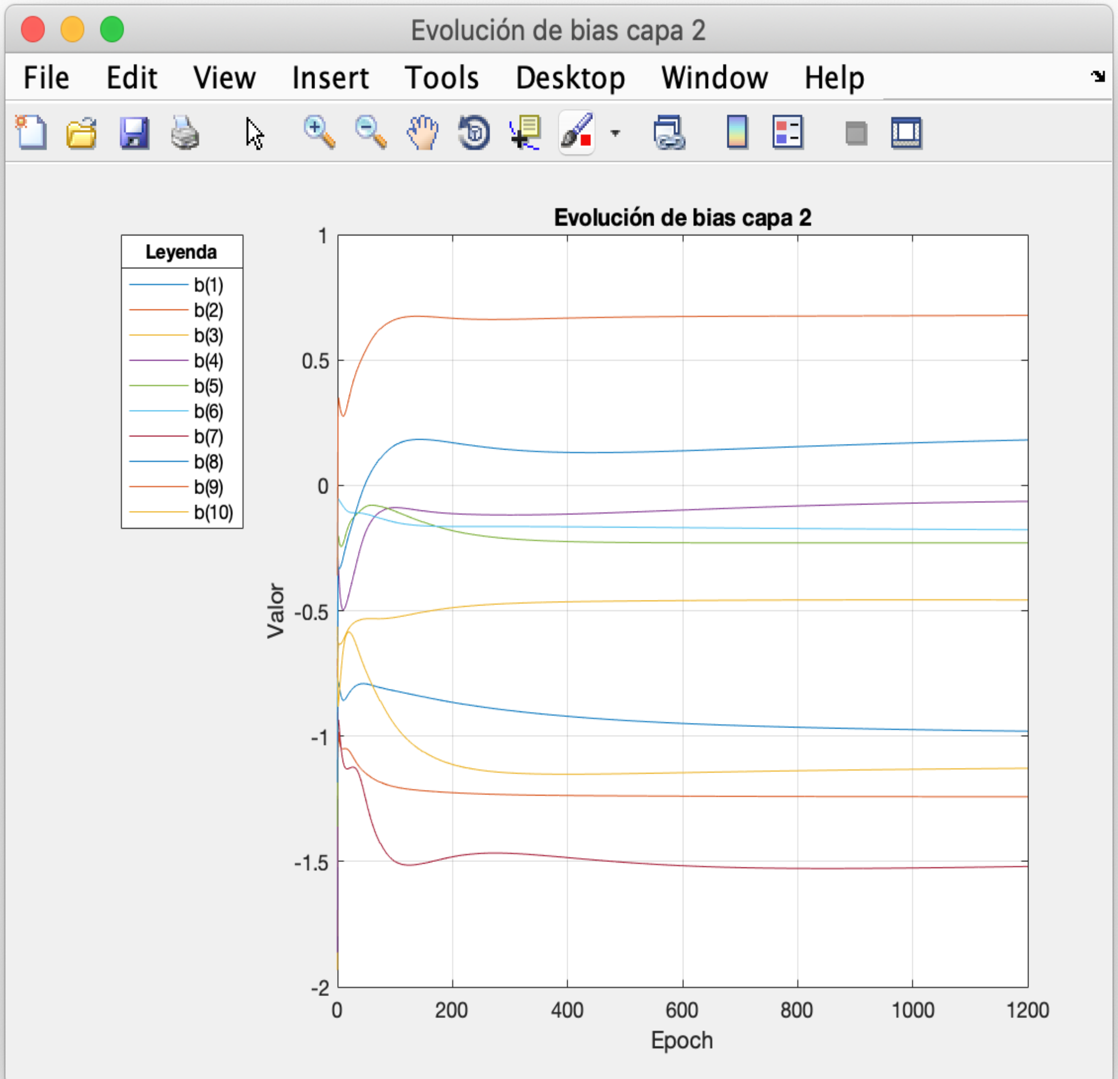


Figura 19: Gráfica 1.7

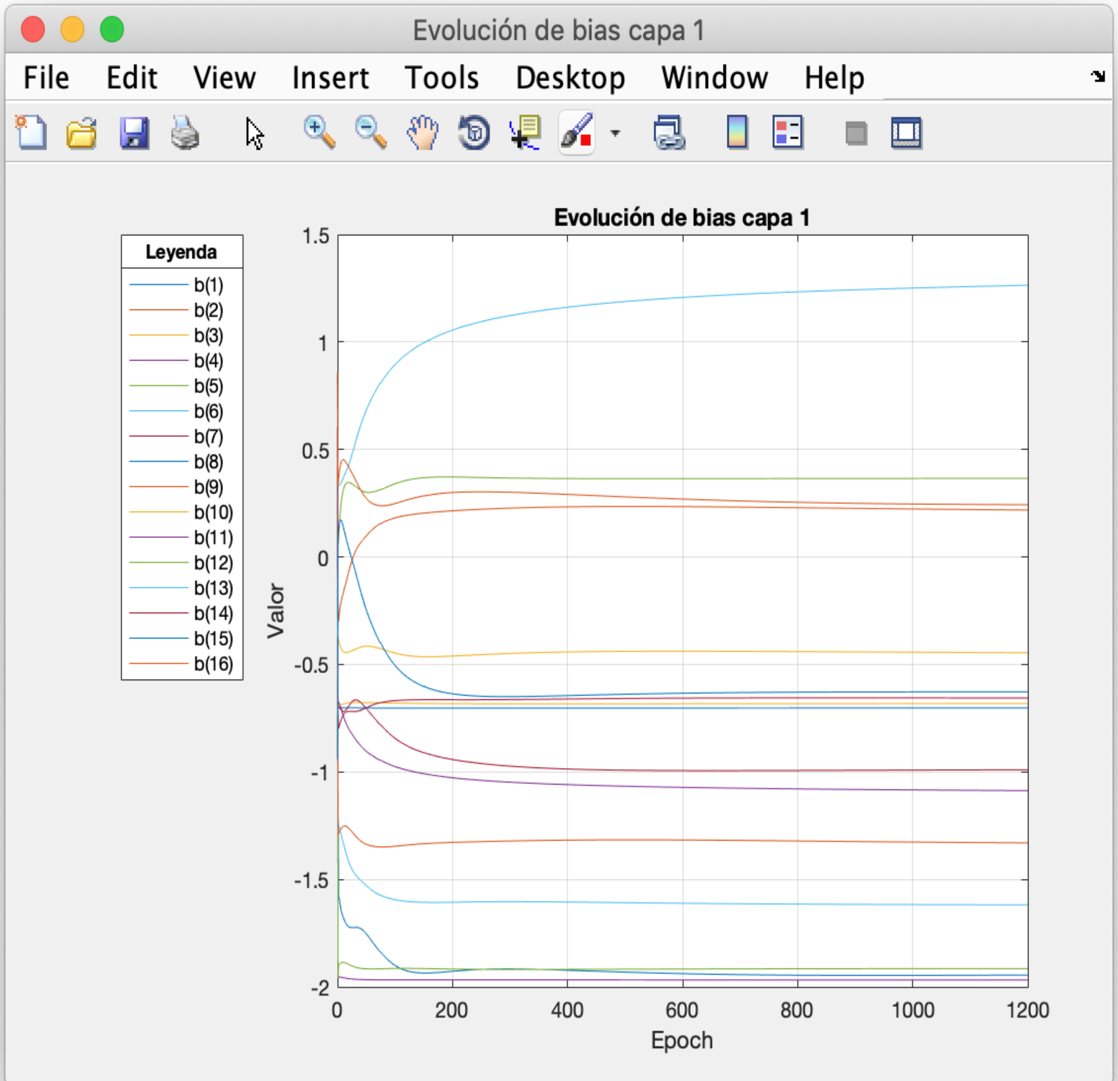


Figura 20: Gráfica 1.8

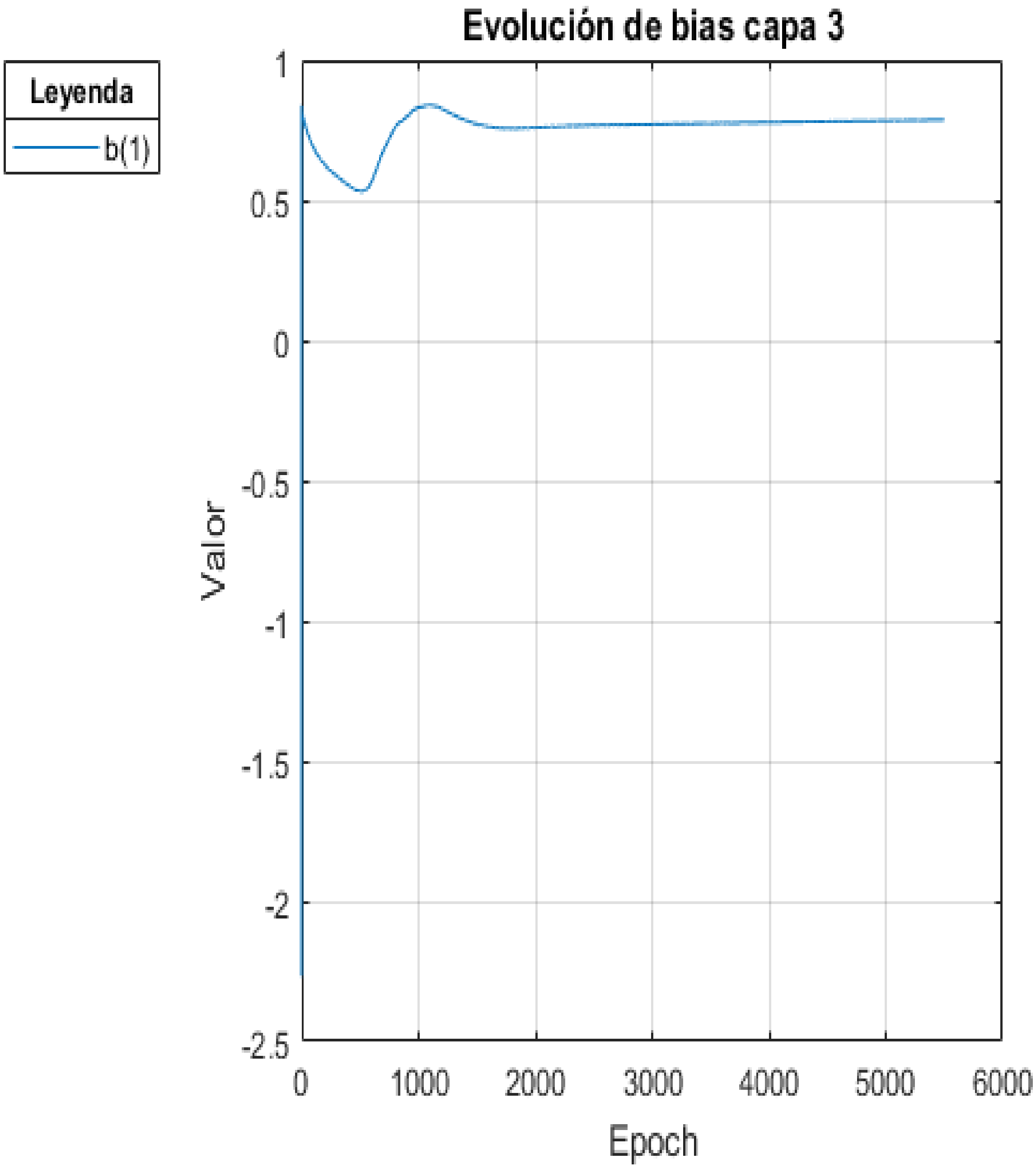


Figura 21: Gráfica 1.9

4. Discusión de Resultados

Para cada uno de los resultados se muestra:

1. Los datos con los cuales fue realizado el ejemplo.
2. Los pesos y bias iniciales.
3. La gráfica del “historial” de la evolución de los parámetros del perceptrón
4. La gráfica de los vectores de entrada con su target y la frontera de desición final.

5. Conclusiones

El perceptrón es la unidad básico de las redes neuronales que se usan hoy en día, su creador, Frank Rosenblatt, hizp una muy importante aportación para el campo de las redes neuronales artificiales. La práctica estuvo mucho más corta de lo que esperaba, la regla de aprendizaje es “magia”.

6. Referencias

Martin T Hagan. Machine Learning, Neural Network Design (2nd Edition), 2014.
https://medium.com/@thomascountz/calculate-the-decision-boundary-of-a-single-perceptron-visualizing

7. Apéndice

Listing 1: Código

```
clc
clear

% Read the inputs file
inputs_path = strcat(input('Ingrese el nombre del archivo de inputs sin la extensión: ','s'), '.txt');
%inputs_path = 'inputs.txt';
inputs = importdata(inputs_path);

% Read the targets
targets_file = strcat(input('Ingrese el nombre del archivo de targets sin la extensión: ','s'), '.txt');
%targets_path = 'targets.txt';
targets = importdata(targets_path);

data_size = size(inputs, 1);

% Enter MLP architecture
architecture = str2num(input('Ingrese el vector de la arquitectura: ','s'));
% Calculate layer parameters
%architecture = str2num('1 16 10 1');
num_layers = length(architecture) - 1;
```

```
R = architecture(1);
functions_vector = str2num(input('Ingrese el vector de las funciones de activación: 1)
     purelin()\n2) logsig()\n3) tansig()\n\n: ','s'));
%functions_vector = str2num('3 2 1');

% Enter the learning factor
alpha = input('Ingresa el valor del factor de aprendizaje(alpha): ');
%alpha = .01;

epochmax = input('Ingresa el número máximo de épocas: ');
% epochmax = 10000;
%validation_iter = 500;
%numval = 7;
%error_epoch_validation = .0000000000000001;
numval = input('Numero maximo de incrementos consecutivos del error de validacion (
    numval): ');
error_epoch_validation = input('Ingrese el valor minimo del error de epoca (
    error_epoch_validation): ');
validation_iter = input('Ingrese el múltiplo de épocas para realizar una época de
    validación  (validation_iter): ');

% Dataset Slicing
config_option = input('Elija una configuración de distribución de datasets: \n1:
    80-10-10\n2: 70-15-15\n');
%config_option = 2;
[training_ds, test_ds, validation_ds] = dataset_slices(config_option, inputs, targets)
    ;
validation_ds_size = size(validation_ds, 1);
test_ds_size = size(test_ds, 1);
training_ds_size = size(training_ds, 1);

disp('Dataset de entrenamiento:');
disp(training_ds);
disp('Dataset de validacion:');
disp(validation_ds);
disp('Dataset de prueba:');
disp(test_ds);

% Open the files for weights and bias
total_weight_files = 0;
total_bias_files = 0;
for i=1:num_layers
% For neurons
for j=1:architecture(i + 1)
% For weights
for l=1:architecture(i)
total_weight_files = total_weight_files + 1;
end
end
```

```
total_bias_files = total_bias_files + 1;
end

W_files = zeros(total_weight_files, 1);
b_files = zeros(total_bias_files, 1);

current_file = 1;
for i=1:num_layers
path = strcat(pwd, '/historico/capa_', num2str(i), '/pesos/');
if ~exist(path, 'dir')
mkdir(path);
end
% For layers
for j=1:architecture(i + 1)
% For neurons
for k=1:architecture(i)
archivo_pesos = strcat(path, '/pesos', num2str(j), '_', num2str(k),'.txt');
W_files(current_file) = fopen(archivo_pesos,'w');
current_file = current_file +1;
end
end
end

current_file = 1;
for i=1:num_layers
path = strcat(pwd,'/historico/capa_', num2str(i), '/bias/');
if ~exist(path, 'dir')
mkdir(path);
end
for j=1:architecture(i+1)
archivo_bias = strcat(path,'/bias',num2str(j),'.txt');
b_files(current_file) = fopen(archivo_bias,'w');
current_file = current_file +1;
end
end

% Initialize MLP parameters and Print them

num_w_files = 1;
num_b_files = 1;
W = cell(num_layers,1);
b = cell(num_layers,1);
% Output of each layer
a = cell(num_layers + 1, 1);
% Sentitivities
S = cell(num_layers, 1);
% Derivatives of each layer
F_m = cell(num_layers, 1);

```

```
% For each layer
for i=1:num_layers
% Random value
W_r_value = 2 * rand(architecture(i + 1), architecture(i)) - 1;
b_r_value = 2* rand(architecture(i + 1), 1) - i;
W{i} = W_r_value
b{i} = b_r_value
% For each neuron
for j=1:architecture(i + 1)
%For each weight
for k=1:architecture(i)
% Print wights value
fprintf(W_files(num_w_files), '%f\r\n', W_r_value(j, k));
num_w_files = num_w_files + 1;
end
end
% For each neuron
for j=1:architecture(i + 1)
% print bias value
fprintf(b_files(num_b_files), '%f\r\n', b_r_value(j));
num_b_files = num_b_files + 1;
end
end

% Learning algorithm
num_validation_epoch = 0;
early_stopping_increment = 0;
validation_error = 0;
learning_error = 0;
early_s_counter = 0;

% initialize vectors for printing errors
learning_err_values = zeros(epochmax, 1);
evaluation_err_values = zeros(ceil(epochmax / validation_iter), 1);
for epoch=1:epochmax
l_error = 0;
% Reset the values
num_w_files = 1;
num_b_files = 1;
% if isn't a validation epoch
if(mod(epoch ,validation_iter) ~= 0)
for t_data=1:training_ds_size
% initial condition
a{1} = training_ds(t_data, 1);
% Foward propagation
for t_p=1:num_layers
W_aux = cell2mat(W(t_p));
b_aux = cell2mat(b(t_p));
a_aux = cell2mat(a(t_p));
```

```
n_f = W_aux * a_aux + b_aux;
a{t_p + 1} = get_activation_function(n_f, functions_vector(t_p));
end
a_aux = cell2mat(a(num_layers + 1));
t_error = training_ds(t_data, 2) - a_aux;
l_error = l_error + (t_error / data_size);
% Sensitivities calculation
F_m{num_layers} = get_F_matrix(functions_vector(num_layers), architecture(num_layers +
    1), a_aux);
F_m_temp = cell2mat(F_m(num_layers));
S{num_layers} = F_m_temp * (t_error)*(-2);
% Backpropagation
for m = num_layers-1:-1:1
W_aux = cell2mat(W(m+1));
s_aux = cell2mat(S(m+1));
a_aux = cell2mat(a(m+1));
F_m{m} = get_F_matrix(functions_vector(m),architecture(m+1),a_aux);
F_m_temp = cell2mat(F_m(m));
S{m} = F_m_temp * (W_aux')*s_aux;
end
% Learning Rules
for k = num_layers:-1:1
W_aux = cell2mat(W(k));
b_aux = cell2mat(b(k));
s_aux = cell2mat(S(k));
a_aux = cell2mat(a(k));
W{k} = W_aux - (alpha * s_aux * a_aux');
b{k} = b_aux - (alpha * s_aux);
W_aux = cell2mat(W(k));
b_aux = cell2mat(b(k));
end
end
learning_error = l_error;
learning_err_values(epoch) = l_error;
% This epoch is a validation one
else
val_error = 0;
num_validation_epoch = num_validation_epoch + 1;
for t_data = 1:validation_ds_size
% Initial Condition
a{1} = validation_ds(t_data, 1);
% Foward propagation
for k=1:num_layers
W_aux = cell2mat(W(k));
a_aux = cell2mat(a(k));
b_aux = cell2mat(b(k));
n_f = W_aux * a_aux + b_aux;
a{k + 1} = get_activation_function(n_f, functions_vector(k));
end
```

```
a_aux = cell2mat(a(num_layers+1));
val_error = validation_ds(t_data,2)-a_aux;
val_error = val_error+(val_error/validation_ds_size);
end
evaluation_err_values(epoch) = val_error;
if early_stopping_increment == 0
validation_error = val_error;
early_stopping_increment = early_stopping_increment+1;
fprintf('Incremento actual para early stopping = %d\n', early_stopping_increment);
else
if val_error > validation_error
validation_error = val_error;
early_stopping_increment = early_stopping_increment+1;
fprintf('Incremento actual para early stopping = %d\n', early_stopping_increment);
if early_stopping_increment == numval
% Reset the counter
early_s_counter = 1;
fprintf('Early stopping en la época:  %d\n', epoch);
break;
end
else
validation_error = 0;
early_stopping_increment = 0;
fprintf('Incremento actual para early stopping = %d\n', early_stopping_increment);
end
end
end

% Print the values on console
num_w_files = 1;
num_b_files = 1;
for k = num_layers:-1:1
W_aux = cell2mat(W(k));
b_aux = cell2mat(b(k));
for j=1:architecture(k+1)
for l=1:architecture(k)
fprintf(W_files(num_w_files), '%f\r\n', W_aux(j,l));
num_w_files = num_w_files +1;
end
end
for j=1:architecture(k + 1)
fprintf(b_files(num_b_files), '%f\r\n', b_aux(j));
num_b_files = num_b_files + 1;
end
end

% Check stopping calculations
if mod(epoch,validation_iter) ~= 0 && l_error <= error_epoch_validation && l_error >=
    0
```

```
learning_error = l_error;
fprintf('Aprendizaje exitoso en la época %d\n', epoch);
break;
end
end

if epoch == epochmax
disp('Se llego a epochmax');
end

% Print the las final values
if early_s_counter == 1
num_w_files = 1;
num_b_files = 1;
for k = num_layers:-1:1
W_aux = cell2mat(W(k));
b_aux = cell2mat(b(k));
for j = 1:architecture(k + 1)
for l=1:architecture(k)
fprintf(W_files(num_w_files), '%f\r\n', W_aux(j, l));
num_w_files = num_w_files + 1;
end
end
for j=1:architecture(k + 1)
fprintf(b_files(num_b_files), '%f\r\n', b_aux(j));
num_b_files = num_b_files + 1;
end
end
end

% Close all files
for i=1:total_weight_files
fclose(W_files(i));
end
for i=1:total_bias_files
fclose(b_files(i));
end

% Propagate the test dataset
test_error = 0;
output = zeros(test_ds_size,1);
for i=1:test_ds_size
% Initial condition
a{1} = test_ds(i,1);
for k=1:num_layers
W_aux = cell2mat(W(k));
a_aux = cell2mat(a(k));
b_aux = cell2mat(b(k));
n_f = W_aux*a_aux+b_aux;
```

```
a{k+1} = get_activation_function(n_f, functions_vector(k));
end
test_data = cell2mat(a(1));
a_aux = cell2mat(a(num_layers + 1));
test_error = test_error + (1 / test_ds_size) * (test_ds(i,2) - a_aux);
output(i) = a_aux;
end

% Print last errors
fprintf('Error de aprendizaje = %f\n', learning_error);
fprintf('Error de validación = %f\n', validation_error);
fprintf('Error de prueba = %f\n', test_error);


% Output vs test
scatter_output_vs_test(test_ds, output);
% Propagate the training size for ploting

output = zeros(training_ds_size,1);
for i=1:training_ds_size
% Initial Condition
a{1} = training_ds(i, 1);
for k=1:num_layers
W_aux = cell2mat(W(k));
a_aux = cell2mat(a(k));
b_aux = cell2mat(b(k));
a{k+1} = get_activation_function(W_aux*a_aux+b_aux,functions_vector(k));
end
a_aux = cell2mat(a(num_layers + 1));
test_error = test_error + (1 / training_ds_size) * (training_ds(i,2) - a_aux);
output(i) = a_aux;
end

scatter_output_vs_training(training_ds, output);

% Plot the error evolution
error_plot(validation_iter, num_validation_epoch, learning_err_values, epoch,
    evaluation_err_values);
% Plot weight evolution
weight_evolution_plot(architecture, num_layers, epoch);
% Plot bias evolution
bias_evolution_plot(architecture, num_layers, epoch);

% Write final values
for i=1:num_layers
path = strcat(pwd, '/Valores_finales/capa_', num2str(i), '/');
if ~exist(path, 'dir')
mkdir(path);
end
```

```
W_aux = cell2mat(W(i));
res_pesos = strcat(path, '/pesos.txt');
dlmwrite(res_pesos, W_aux, ';');
end

for i=1:num_layers
path = strcat(pwd,'/Valores_finales/capa_', num2str(i), '/');
if ~exist(path, 'dir')
mkdir(path);
end
b_aux = cell2mat(b(i));
res_bias = strcat(path, '/bias.txt');
dlmwrite(res_bias, b_aux, ';');
end
```