19 de Noviembre del 2018

Josue Ruiz Hernández

Juan Damian Osornio Gutierrez

Escuela superior de cómputo

Segundo avance del MLP

Redes neuronales

# Explicación del programa

Se crearon 4 inputs y 4 targets con la ecuación:

Para p 🡪 -2<p<2

Tomando i=1,2,4 y 8

Para esto generamos un vector p con una frecuencia de muestreo de 0.04 lo que en Matlab se podría traducir a p=-2:0.04:2

Teóricamente la función g(p) se graficaría de la siguiente manera:

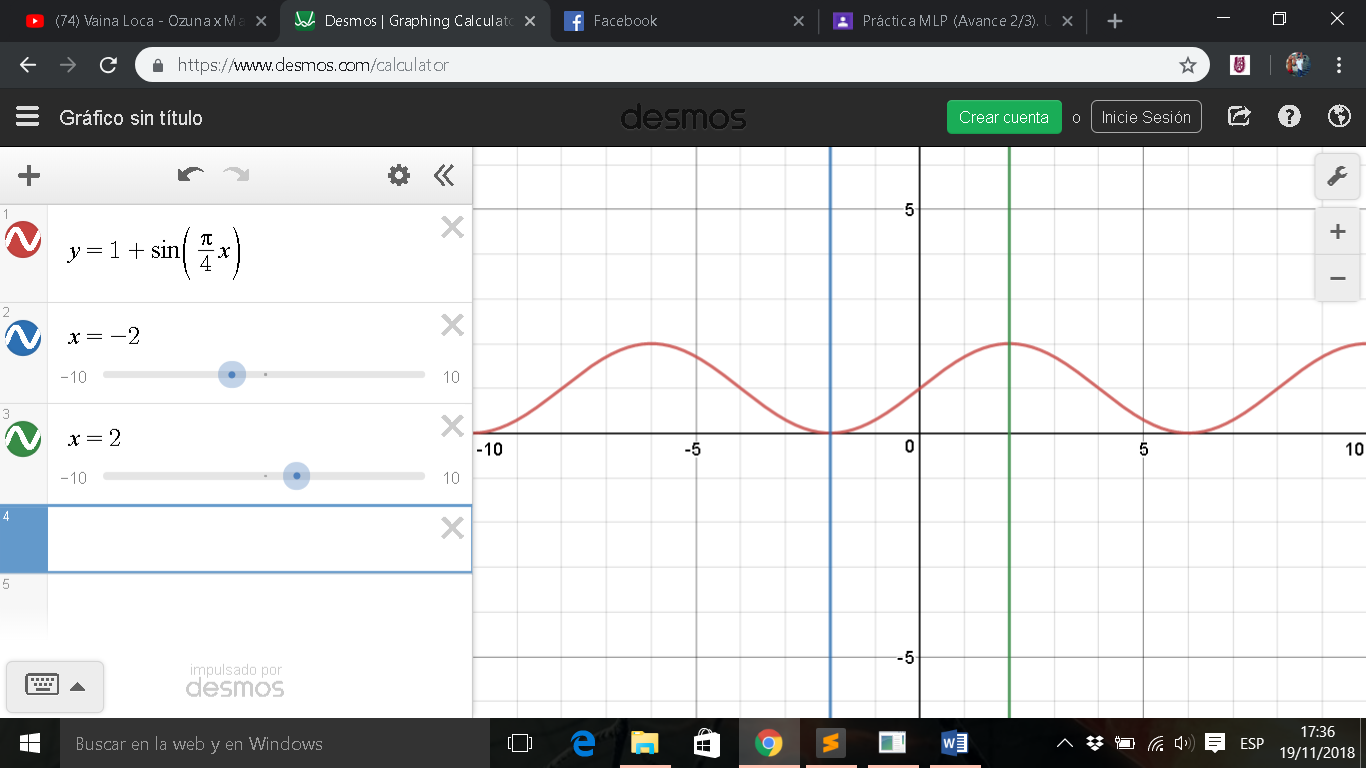


Figure 1 grafica de g(p) con i=1

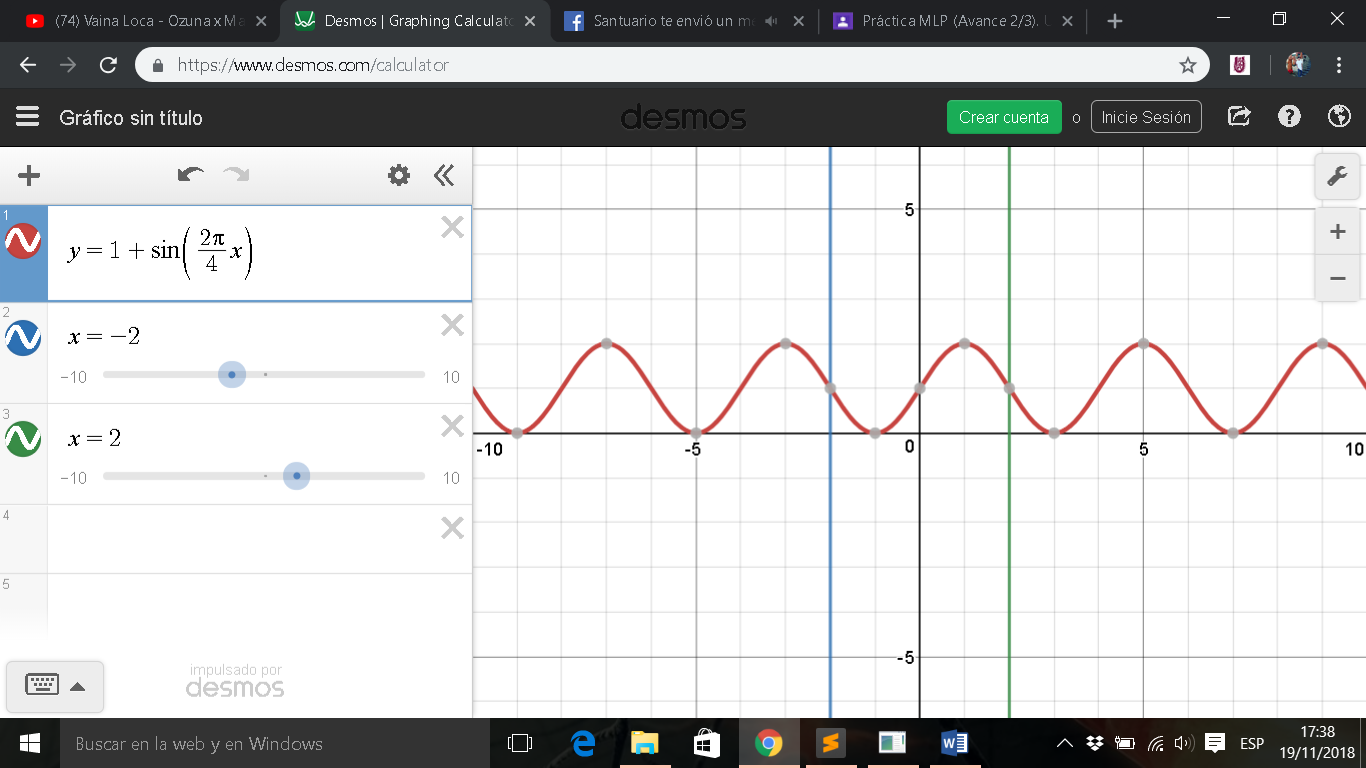


Figure 2 grafica de g(p) con i=2

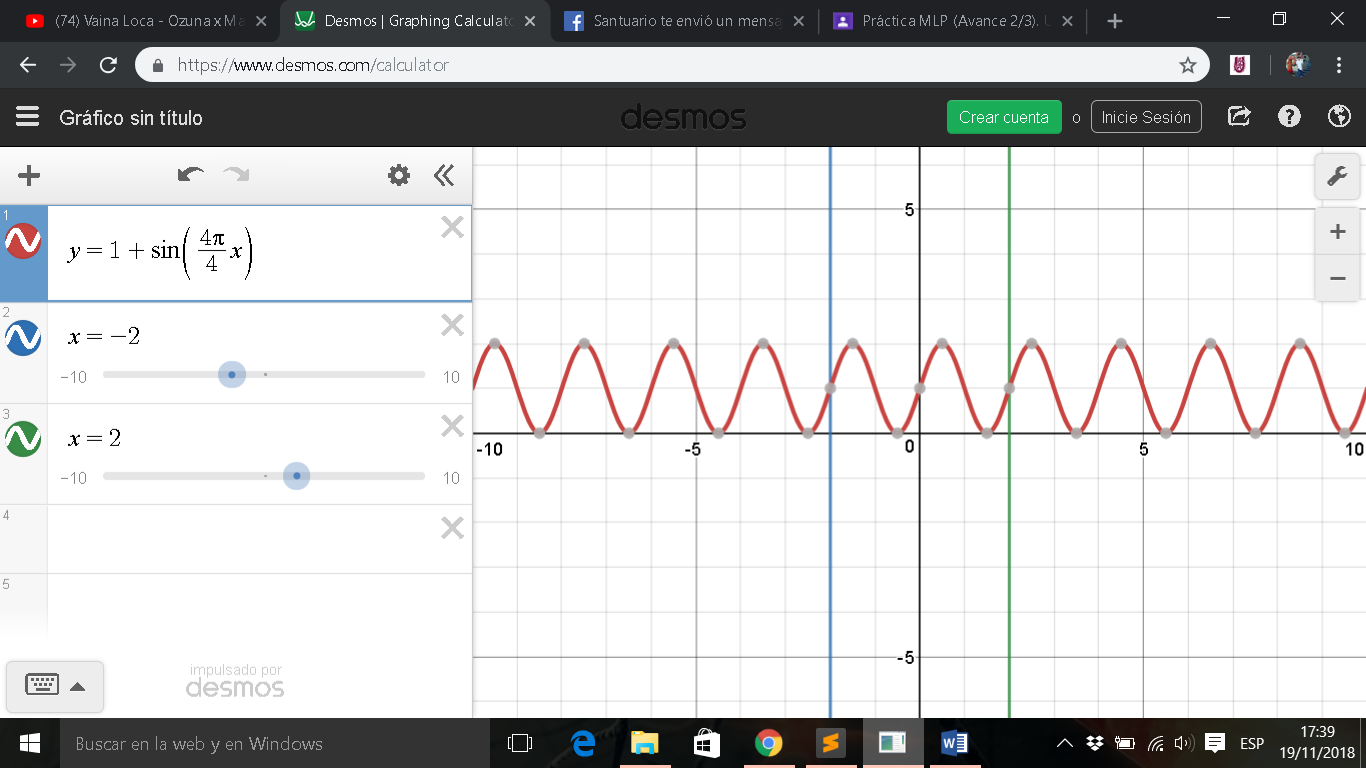


Figure 3 grafica de g(p) con i=4

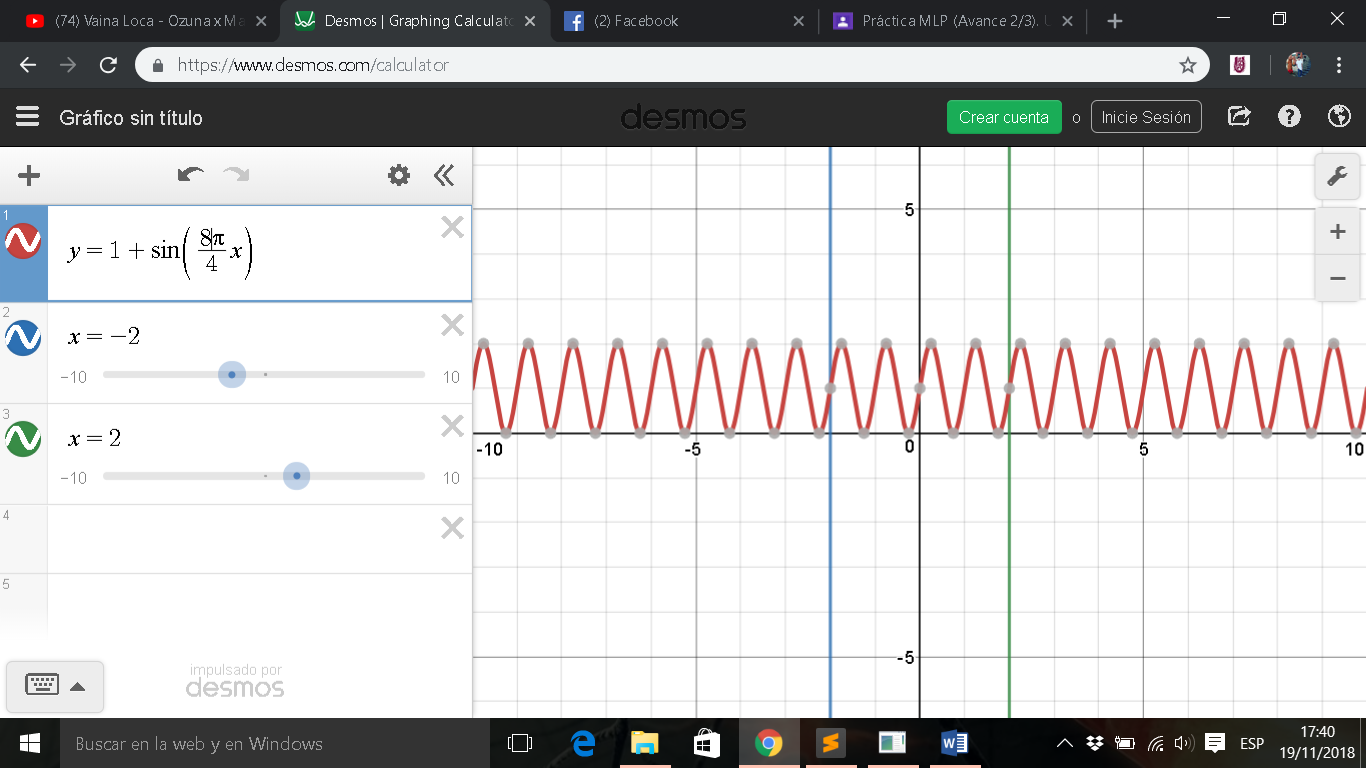


Figure 4 grafica de g(p) con i=8

Para crear los datasets hicimos un programa muy sencillo, cuyo código es el siguiente:

1. data=zeros(100,1);
2. pos=1;
3. a=-2:0.04:2;
4. **for** p = -2:0.04:2
5. data(pos)=1+sin((pi\*p\*8)/4);
6. pos=pos+1;
7. end
9. ip=fopen('target\_#4.txt' ,'w');
10. **for** d=1:101
11. fprintf(ip,'%.4f\n',data(d));
12. end
13. fclose(ip);

16. ipp=fopen('input\_#4.txt' ,'w');
17. **for** d=1:101
18. fprintf(ipp,'%.4f\n',a(d));
19. end
20. fclose(ipp);

Lo único que hace es crear el input y el target con sus respectivas i, en el código se ve el 8 en la línea 5 pero ese es el que cambiamos por 1, 2, 4 u 8. Con esto obtenemos 4 datasets con 100 muestras de entre -2 y 2.

Para el programa decidimos hacer funciones que hagan una cosa en específico como sería el separar datos de manera uniforme en los 3 conjuntos (entrenamiento,validación y prueba). El algoritmo es el siguiente:

1. function [C\_E,C\_V,C\_P,T\_E,T\_V,T\_P]=separar\_datos(entradas,target,opcion,numero\_datos)
2. size(entradas)
3. **if**(opcion==1)
4. indices\_validacion=unique(randi([1 numero\_datos],round(numero\_datos\*0.10),1));
5. **for** i=1:size(indices\_validacion,1)
6. C\_V(i,1)=entradas(indices\_validacion(i,1),1);
7. T\_V(i)=target(indices\_validacion(i,1));
8. end
10. **for** i=size(indices\_validacion,1):1
11. entradas(indices\_validacion(i,1))=[];
12. target(indices\_validacion(i,1))=[];
13. end
15. numero\_datos=numero\_datos-round(numero\_datos\*0.1);
16. indices\_validacion=unique(randi([1 numero\_datos],round(numero\_datos\*0.10),1));
17. size(indices\_validacion)
18. **for** i=1:size(indices\_validacion,1)
19. C\_P(i,1)=entradas(indices\_validacion(i,1));
20. T\_P(i,1)=target(indices\_validacion(i,1));
21. end
23. **for** i=size(indices\_validacion,1):1
24. entradas(indices\_validacion(i,1))=[];
25. target(indices\_validacion(i,1))=[];
26. end
28. numero\_datos=numero\_datos-round(numero\_datos\*0.1);
29. **for** i=1:numero\_datos
30. C\_E(i,1)=entradas(i,1);
31. T\_E(i,1)=target(i,1);
32. end
33. end
35. **if**(opcion==2)
36. indices\_validacion=unique(randi([1 numero\_datos],round(numero\_datos\*0.15),1));
37. **for** i=1:size(indices\_validacion,1)
38. C\_V(i,1)=entradas(indices\_validacion(i,1),1);
39. T\_V(i)=target(indices\_validacion(i,1));
40. end
42. **for** i=size(indices\_validacion,1):1
43. entradas(indices\_validacion(i,1))=[];
44. target(indices\_validacion(i,1))=[];
45. end
47. numero\_datos=numero\_datos-round(numero\_datos\*0.15);
48. indices\_validacion=unique(randi([1 numero\_datos],round(numero\_datos\*0.15),1));
49. **for** i=1:size(indices\_validacion,1)
50. C\_P(i,1)=entradas(indices\_validacion(i,1));
51. T\_P(i,1)=target(indices\_validacion(i,1));
52. end
54. **for** i=size(indices\_validacion,1):1
55. entradas(indices\_validacion(i,1))=[];
56. target(indices\_validacion(i,1))=[];
57. end
59. numero\_datos=numero\_datos-round(numero\_datos\*0.15);
60. **for** i=1:numero\_datos
61. C\_E(i,1)=entradas(i,1);
62. T\_E(i,1)=target(i,1);
63. end
64. end
66. end

Como vamos a usar feedforward 3 veces, una en entrenamiento, otra en validación y la ultima en prueba, entonces creamos la función feedforward que es la siguiente:

1. function Salida=feedFordward(pesos,entrada,bias,num\_capas,funciones)
2. a=cell(num\_capas+1,1);
3. a(1,1)=entrada;
4. **for** i=2:num\_capas+1
5. a(i,1)=funcion\_activacion(pesos{i-1},bias{i-1},a{i-1},funciones(1,i-1));
6. end
7. Salida=a;
8. end

la cual hace uso de la función de activación que se muestra a continuación:

1. function Salida=funcion\_activacion(pesos,bias,entrada,opcion)
2. **if**(opcion==1)
3. Salida = pesos\*entrada+bias;
4. end
5. **if**(opcion==2)
6. Salida=logsig(pesos\*entrada+bias);
7. end
8. **if**(opcion==3)
9. Salida=tansig(pesos\*entrada+bias);
10. end
11. end
13. function Salida=logsig(n)
14. **for** i=1:size(n,1)
15. **for** j=1:size(n,2)
16. Salida(i,j)=1/(1+exp(-n(i,j)));
17. end
18. end
19. end
21. function Salida=tansig(n)
22. **for** i=1:size(n,1)
23. **for** j=1:size(n,2)
24. Salida(i,j)=(exp(n(i,j))-exp(-n(i,j)))/(exp(n(i,j))+exp(-n(i,j)));
25. end
26. end
27. end

En la época de entrenamiento hacemos la actualización de pesos y bias, para eso hacemos uso de la función backpropagation:

1. function [Pesos,Bias]=backpropagation(pesos,bias,error\_i,arquitectura,funciones,salida,factor\_a)
3. %Sensitividades %
4. total\_capas=size(funciones,2);
5. S=cell(total\_capas,1);
6. derivada=derivada\_funcion\_activacion(funciones(1,total\_capas),arquitectura(size(arquitectura,2)),salida{total\_capas+1});
7. S{total\_capas}=-2\*derivada\*error\_i;
9. **for** capa=total\_capas:2
10. derivada=derivada\_funcion\_activacion(funciones(1,capa-1),arquitectura(1,capa),salida{capa});
11. S{capa-1}=derivada\*transpose(pesos{capa})\*S{capa};
12. end
13. %Final de sensitividades%
15. % Actualizando pesos y bias%
16. **for** i=total\_capas:2
17. pesos{i}=pesos{i}-factor\_a\*S{i}\*transpose(salida{i-1});
18. bias{i}=bias{i}-factor\_a\*S{i};
19. end
20. %Finaliza actualizacion%
21. Pesos=pesos;
22. Bias=bias;
23. end

Pero claro para las sentividades hacemos uso de las derivadas de la función de transferencia y para ello otra función:

1. function S=derivada\_funcion\_activacion(opcion,n\_neuronas,salida)
2. Matriz=zeros(n\_neuronas);
4. **if** opcion==1
5. **for** i=1:size(Matriz,1)
6. **for** j=1:size(Matriz,2)
7. Matriz(i,j)=1;
8. end
9. end
10. end
12. **if** opcion==2
13. **for** i=1:size(Matriz,1)
14. **for** j=1:size(Matriz,2)
15. **if** i==j
16. Matriz(i,j)=salida(i)\*(1-salida(i));
17. end
18. end
19. end
20. end
22. **if** opcion==3
23. **for** i=1:size(Matriz,1)
24. **for** j=1:size(Matriz,2)
25. **if** i==j
26. Matriz(i,j)=1-salida(i)^2;
27. end
28. end
29. end
30. end
32. S=Matriz;
33. end

Ya sabiendo eso el código del main es el siguiente donde se ven usadas las funciones que explicamos:

1. addpath("./Funciones/")
3. % 1a y 1b Pedir archivos%
4. archivo\_entrada=input('Ingrese el nombre del archivo con los datos de entrada: ','s');
5. ruta='Archivos/';
6. archivo\_entrada=strcat(ruta,archivo\_entrada);
8. datos\_entrada=importdata(archivo\_entrada);
9. datos\_grafica=datos\_entrada;
10. [filas\_entrada,columnas\_entrada]=size(datos\_entrada);
12. archivo\_entrada=input('Ingrese el nombre del archivo con los targets: ','s');
13. ruta='Archivos/';
14. archivo\_entrada=strcat(ruta,archivo\_entrada);
16. targets=importdata(archivo\_entrada);
17. [filas\_target,columnas\_target]=size(targets);
18. targets\_grafica=targets;
19. %Fin de la peticion de entradas%
21. % 1c Se pide el rango de la señal%
22. limite\_inferior=input('Limite inferior de la senal: ');
23. limite\_superior=input('Limite superior de la senal: ');
24. **if**(limite\_inferior<=limite\_superior)
25. rango=limite\_inferior:1/(filas\_entrada):limite\_superior;
26. **else**
27. rango=limite\_superior:1/(filas\_entrada):limite\_inferior;
28. end
29. %Fin del modulo%
31. % 1d Pedir vectores de arquitectura%
32. vector\_arquitectura=input('Vector de arquitectura: ');
33. [uno,tam\_vector\_arquitectura]=size(vector\_arquitectura);
34. vector\_funciones=input('Vector de funciones: ');
35. [uno,numero\_capas]=size(vector\_funciones);
36. %Fin del modulo 1d%
38. % 1e Pedir valor del factor de aprendizaje%
39. factor\_aprendizaje=input('Ingrese el factor de aprendizaje: ');
40. %Fin del modulo 1e%
42. %1f valores de las condiciones de finalizacion%
43. epocas\_totales=input('Numero de epocas: ');
44. error\_epoch\_max=input('Senal de error: ');
45. epoch\_val=input('Cada cuantas iteraciones se llevara a cabo una epoca de validacion: ');
46. num\_val=input('Numero maximo de incrementos consecutivos del error de epoca de validacion: ');
47. %Fin del modulo 1f%
49. %1g Division del dataset en 3%
50. disp('Escoja la opcion que guste para la division del dataset: ');
51. disp('1. 80%-10%-10%');
52. disp('2. 70%-15%-15%');
53. opcion=input(' ');
54. [conjunto\_entrenamiento,conjunto\_validacion,conjunto\_prueba,target\_entrenamiento,target\_validacion,target\_prueba]=separar\_datos(datos\_entrada,targets,opcion,filas\_entrada);
55. [filas\_validacion,columnas\_validacion]=size(conjunto\_validacion);
56. %Fin del modulo 1g%

59. % 2. Se inician los valores aleatorios entre -1 y 1%
60. pesos=cell(numero\_capas,1);
61. bias=cell(numero\_capas,1);
62. **for** i=1:tam\_vector\_arquitectura-1
63. pesos(i,1)=-1+2\*rand(vector\_arquitectura(1,i+1),vector\_arquitectura(1,i));
64. bias(i,1)=-1+2\*rand(vector\_arquitectura(i+1),1);
65. end
66. %Final del modulo 2%
68. % Comenzando el entrenamiento %
69. error\_epoca\_val=zeros(100,1);
70. contador\_validacion=2;
71. contador\_val=0;
72. Salida\_iteracion=0;
73. band=0;
74. eleccion=1;
75. **while**(eleccion==1)
76. inicio=1;
77. **for** epoca=inicio:epocas\_totales
79. **if** mod(epoca,num\_val)==0
80. suma\_errores=0;
81. **for** iteracion=1:size(conjunto\_validacion,1)
82. Salida\_iteracion=feedFordward(pesos,conjunto\_validacion(iteracion),bias,numero\_capas,vector\_funciones);
83. suma\_error=suma\_error+(target\_validacion(iteracion)-Salida\_iteracion{numero\_capas+1})^2;
84. end
85. error\_epoca\_val(contador\_validacion)=suma\_error/filas\_entrada;
87. **if** epoca>2
88. **if** error\_epoca\_val(contador\_validacion-1)<error\_epoca\_val(contador\_validacion)
89. contador\_val=contador\_val+1;
90. **else**
91. contador\_val=0;
92. end
93. end
95. **if** contador\_val==num\_val
96. fprintf(1, '\nSalida por aumento de errores en validacion\n');
97. band=1;
98. **break**;
99. end
100. contador\_validacion=contador\_validacion+1;

103. **else**
104. suma\_error=0;
105. fprintf(1,'---------------------');
106. **for** iteracion=1:size(conjunto\_entrenamiento,1)
107. Salida\_iteracion=feedFordward(pesos,conjunto\_entrenamiento(iteracion),bias,numero\_capas,vector\_funciones);
108. error\_it=(target\_entrenamiento(iteracion)-Salida\_iteracion{numero\_capas+1})^2;
109. suma\_error=suma\_error+error\_it;
110. %Salida\_iteracion{numero\_capas+1}
111. [pesos,bias]=backpropagation(pesos,bias,error\_it,vector\_arquitectura,vector\_funciones,Salida\_iteracion,factor\_aprendizaje);
112. end
113. fprintf(1,'---------------------');
114. Error=(suma\_error/size(conjunto\_entrenamiento,1))
116. **if** Error<error\_epoch\_max
117. band=1;
118. fprintf(1,'Salida exitosa con el error %d\n',Error);
119. **break**;
120. end
122. **if** epoca==epocas\_totales
123. eleccion=input('Desea hacer mas iteraciones? 1. si 2 no');
124. end
125. end
126. end
128. **if** band==1
129. **break**;
130. end
131. end
132. % Fin del entrenamiento %
134. % Comprobacion %
135. suma\_error=0;
136. **for** iteracion=1:size(conjunto\_prueba,1)
137. Salida\_iteracion=feedFordward(pesos,conjunto\_prueba(iteracion),bias,numero\_capas,vector\_funciones);
138. error\_it=(target\_prueba(iteracion)-Salida\_iteracion{numero\_capas+1});
139. suma\_error=suma\_error+error\_it;
141. end
143. Error=suma\_error/size(conjunto\_prueba,1);
144. **if**(abs(Error)<0.00001)
145. fprintf(1,"Aprendido exitosamente\n");
146. end
147. % Fin de la comprobacion%


151. % Graficando resultados %
152. plot(transpose(datos\_grafica),transpose(targets\_grafica));
153. hold on;
154. **for** iteracion=1:size(datos\_grafica,1)
155. Salida\_iteracion=feedFordward(pesos,datos\_entrada(iteracion),bias,numero\_capas,vector\_funciones);
156. salida\_red(iteracion)=Salida\_iteracion{numero\_capas+1};
157. end
158. plot(transpose(datos\_grafica),transpose(salida\_red));
159. % Fin de la impresion %

# Resultados

Para este resultado escojimos la funcion 1 y los conjuntos de entrenamiento fueron los siguientes:

Conjunto de validacion

-1.80000

-1.56000

-0.80000

-0.64000

0.16000

0.40000

0.60000

0.92000

1.60000

1.80000

Conjunto de prueba

-1.88000

-0.72000

-0.64000

-0.44000

-0.40000

0.68000

0.72000

1.08000

1.60000

Conjunto de entrenamiento

-2.00000

-1.96000

-1.92000

-1.88000

-1.84000

-1.80000

-1.76000

-1.72000

-1.68000

-1.64000

-1.60000

-1.56000

-1.52000

-1.48000

-1.44000

-1.40000

-1.36000

-1.32000

-1.28000

-1.24000

-1.20000

-1.16000

-1.12000

-1.08000

-1.04000

-1.00000

-0.96000

-0.92000

-0.88000

-0.84000

-0.80000

-0.76000

-0.72000

-0.68000

-0.64000

-0.60000

-0.56000

-0.52000

-0.48000

-0.44000

-0.40000

-0.36000

-0.32000

-0.28000

-0.24000

-0.20000

-0.16000

-0.12000

-0.08000

-0.04000

0.00000

0.04000

0.08000

0.12000

0.16000

0.20000

0.24000

0.28000

0.32000

0.36000

0.40000

0.44000

0.48000

0.52000

0.56000

0.60000

0.64000

0.68000

0.72000

0.76000

0.80000

0.84000

0.88000

0.92000

0.96000

1.00000

1.04000

1.08000

1.12000

1.16000

1.20000

1.24000

Los target son:

|  |
| --- |
| 0.0000 |
| 0.0005 |
| 0.0020 |
| 0.0044 |
| 0.0079 |
| 0.0123 |
| 0.0177 |
| 0.0241 |
| 0.0314 |
| 0.0397 |
| 0.0489 |
| 0.0591 |
| 0.0702 |
| 0.0822 |
| 0.0952 |
| 0.1090 |
| 0.1237 |
| 0.1393 |
| 0.1557 |
| 0.1729 |
| 0.1910 |
| 0.2098 |
| 0.2295 |
| 0.2499 |
| 0.2710 |
| 0.2929 |
| 0.3155 |
| 0.3387 |
| 0.3626 |
| 0.3871 |
| 0.4122 |
| 0.4379 |
| 0.4642 |
| 0.4910 |
| 0.5182 |
| 0.5460 |
| 0.5742 |
| 0.6029 |
| 0.6319 |
| 0.6613 |
| 0.6910 |
| 0.7210 |
| 0.7513 |
| 0.7819 |
| 0.8126 |
| 0.8436 |
| 0.8747 |
| 0.9059 |
| 0.9372 |
| 0.9686 |
| 1.0000 |
| 1.0314 |
| 1.0628 |
| 1.0941 |
| 1.1253 |
| 1.1564 |
| 1.1874 |
| 1.2181 |
| 1.2487 |
| 1.2790 |
| 1.3090 |
| 1.3387 |
| 1.3681 |
| 1.3971 |
| 1.4258 |
| 1.4540 |
| 1.4818 |
| 1.5090 |
| 1.5358 |
| 1.5621 |
| 1.5878 |
| 1.6129 |
| 1.6374 |
| 1.6613 |
| 1.6845 |
| 1.7071 |
| 1.7290 |
| 1.7501 |
| 1.7705 |
| 1.7902 |
| 1.8090 |
| 1.8271 |
| 1.8443 |
| 1.8607 |
| 1.8763 |
| 1.8910 |
| 1.9048 |
| 1.9178 |
| 1.9298 |
| 1.9409 |
| 1.9511 |
| 1.9603 |
| 1.9686 |
| 1.9759 |
| 1.9823 |
| 1.9877 |
| 1.9921 |
| 1.9956 |
| 1.9980 |
| 1.9995 |
| 2.0000 |

La salida de la red es la siguiente:

|  |
| --- |
| -0.0490 |
| -0.0386 |
| -0.0279 |
| -0.0167 |
| -0.0051 |
| 0.0070 |
| 0.0195 |
| 0.0325 |
| 0.0459 |
| 0.0597 |
| 0.0740 |
| 0.0888 |
| 0.1040 |
| 0.1196 |
| 0.1357 |
| 0.1523 |
| 0.1693 |
| 0.1868 |
| 0.2047 |
| 0.2230 |
| 0.2418 |
| 0.2611 |
| 0.2808 |
| 0.3009 |
| 0.3214 |
| 0.3424 |
| 0.3637 |
| 0.3855 |
| 0.4077 |
| 0.4303 |
| 0.4533 |
| 0.4766 |
| 0.5003 |
| 0.5244 |
| 0.5489 |
| 0.5737 |
| 0.5988 |
| 0.6242 |
| 0.6500 |
| 0.6761 |
| 0.7025 |
| 0.7291 |
| 0.7560 |
| 0.7832 |
| 0.8107 |
| 0.8383 |
| 0.8662 |
| 0.8944 |
| 0.9227 |
| 0.9512 |
| 0.9799 |
| 1.0087 |
| 1.0377 |
| 1.0669 |
| 1.0962 |
| 1.1255 |
| 1.1550 |
| 1.1846 |
| 1.2142 |
| 1.2439 |
| 1.2737 |
| 1.3035 |
| 1.3333 |
| 1.3631 |
| 1.3929 |
| 1.4227 |
| 1.4524 |
| 1.4822 |
| 1.5118 |
| 1.5414 |
| 1.5709 |
| 1.6003 |
| 1.6296 |
| 1.6588 |
| 1.6878 |
| 1.7167 |
| 1.7455 |
| 1.7740 |
| 1.8024 |
| 1.8306 |
| 1.8586 |
| 1.8864 |
| 1.9139 |
| 1.9413 |
| 1.9683 |
| 1.9951 |
| 2.0217 |
| 2.0479 |
| 2.0739 |
| 2.0996 |
| 2.1249 |
| 2.1500 |
| 2.1747 |
| 2.1992 |
| 2.2232 |
| 2.2470 |
| 2.2704 |
| 2.2934 |
| 2.3161 |
| 2.3384 |
| 2.3603 |

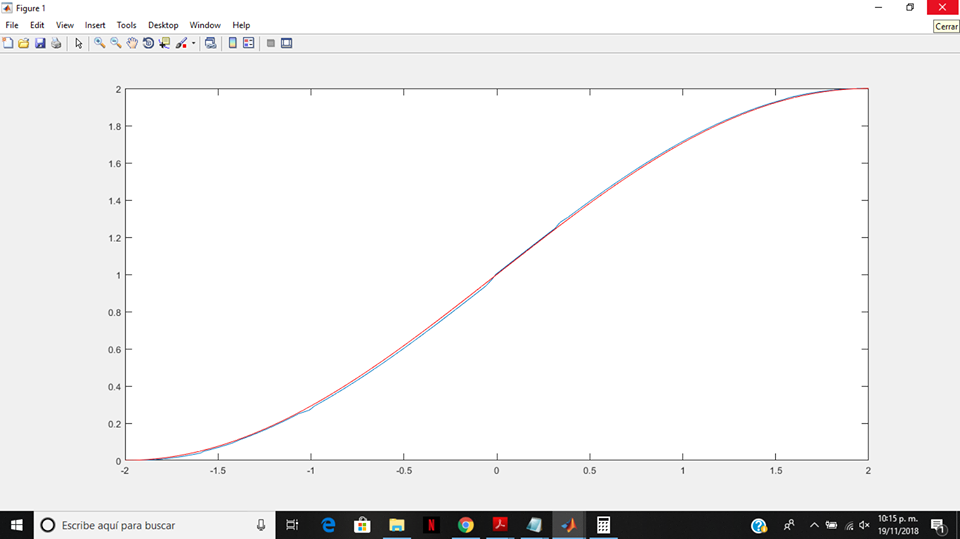


Figure 2 Gráfica de salida