19 de Noviembre del 2018

Segundo avance del MLP

Redes neuronales

Josue Ruiz Hernández
Juan Damian Osornio Gutierrez
ESCUELA SUPERIOR DE CÓMPUTO

Explicación del programa

Se crearon 4 inputs y 4 targets con la ecuación:

$$g(p) = 1 + sen\left(\frac{i\pi p}{4}\right)$$

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:

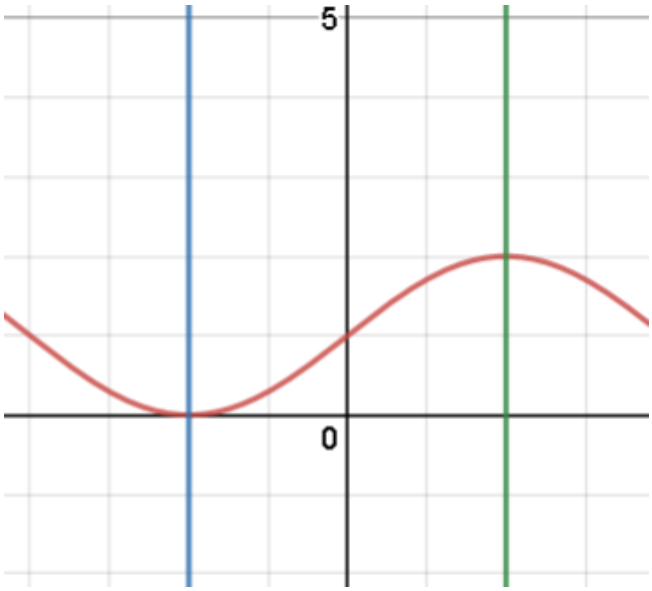

Figure 1 grafica de g(p) con i=1

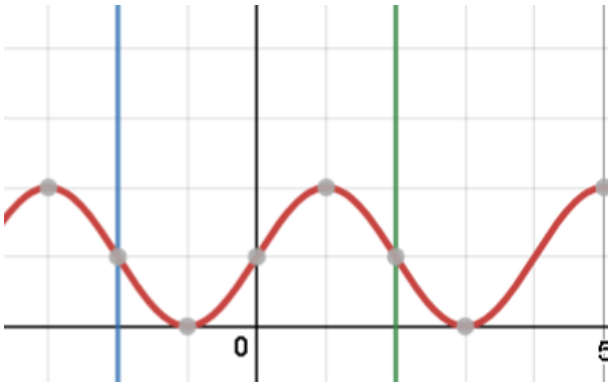

Figure 2 grafica de g(p) con i=2

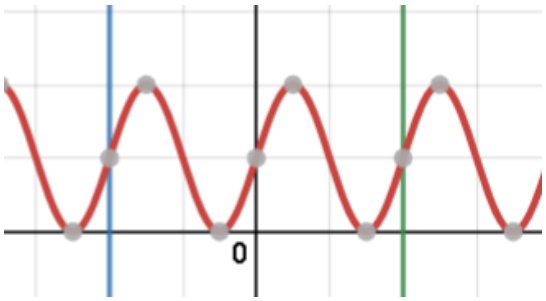

Figure 3 grafica de g(p) con i=4

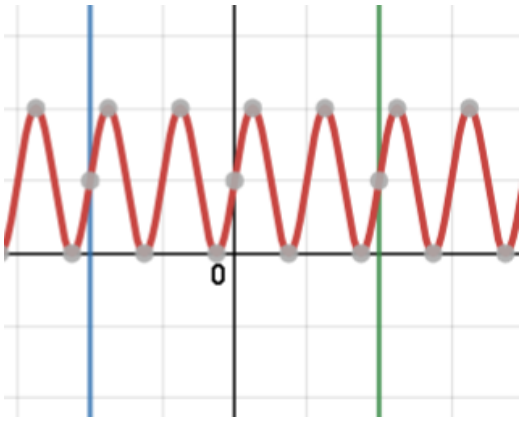

Figure 4 grafica de g(p) con i=8

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
8.
9.  ip=fopen('target_#4.txt' ,'w');
10. for d=1:101
11. fprintf(ip,'%.4f\n',data(d));
12. end
13. fclose(ip);
14.
15.
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:

```
function [C_E,C_V,C_P,T_E,T_V,T_P]=separar_datos(entradas,target,opcion,numero_datos)
size(entradas)
if(opcion==1)
    indices_validacion=unique(randi([1 numero_datos],round(numero_datos*0.10),1));
    for i=1:size(indices_validacion,1)
        C_V(i,1)=entradas(indices_validacion(i,1),1);
        T_V(i)=target(indices_validacion(i,1));
    end

    for i=size(indices_validacion,1):1
        entradas(indices_validacion(i,1))=[];
        target(indices_validacion(i,1))=[];
    end

    numero_datos=numero_datos-round(numero_datos*0.1);
    indices_validacion=unique(randi([1 numero_datos],round(numero_datos*0.10),1));
    size(indices_validacion)
    for i=1:size(indices_validacion,1)
        C_P(i,1)=entradas(indices_validacion(i,1));
        T_P(i,1)=target(indices_validacion(i,1));
    end

    for i=size(indices_validacion,1):1
        entradas(indices_validacion(i,1))=[];
        target(indices_validacion(i,1))=[];
    end

    numero_datos=numero_datos-round(numero_datos*0.1);
    for i=1:numero_datos
        C_E(i,1)=entradas(i,1);
        T_E(i,1)=target(i,1);
    end
end

if(opcion==2)
    indices_validacion=unique(randi([1 numero_datos],round(numero_datos*0.15),1));
    for i=1:size(indices_validacion,1)
        C_V(i,1)=entradas(indices_validacion(i,1),1);
        T_V(i)=target(indices_validacion(i,1));
    end

    for i=size(indices_validacion,1):1
        entradas(indices_validacion(i,1))=[];
        target(indices_validacion(i,1))=[];
    end

    numero_datos=numero_datos-round(numero_datos*0.15);
    indices_validacion=unique(randi([1 numero_datos],round(numero_datos*0.15),1));
    for i=1:size(indices_validacion,1)
        C_P(i,1)=entradas(indices_validacion(i,1));
        T_P(i,1)=target(indices_validacion(i,1));
```

```
    end

    for i=size(indices_validacion,1):1
        entradas(indices_validacion(i,1))=[];
        target(indices_validacion(i,1))=[];
    end

    numero_datos=numero_datos-round(numero_datos*0.15);
    for i=1:numero_datos
        C_E(i,1)=entradas(i,1);
        T_E(i,1)=target(i,1);
    end
end

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:

```
function Salida=feedFordward(pesos,entrada,bias,num_capas,funciones)
    a=cell(num_capas+1,1);
    a(1,1)=entrada;
    for i=2:num_capas+1
        a(i,1)=funcion_activacion(pesos{i-1},bias{i-1},a{i-1},funciones(1,i-1));
    end
    Salida=a;
end
```

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

```
function Salida=funcion_activacion(pesos,bias,entrada,opcion)
    if(opcion==1)
        Salida = pesos*entrada+bias;
    end
    if(opcion==2)
        Salida=logsig(pesos*entrada+bias);
    end
    if(opcion==3)
        Salida=tansig(pesos*entrada+bias);
    end
end

function Salida=logsig(n)
    for i=1:size(n,1)
        for j=1:size(n,2)
            Salida(i,j)=1/(1+exp(-n(i,j)));
        end
    end
end

function Salida=tansig(n)
    for i=1:size(n,1)
        for j=1:size(n,2)
            Salida(i,j)=(exp(n(i,j))-exp(-n(i,j)))/(exp(n(i,j))+exp(-n(i,j)));
        end
    end
end
```

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

```
function [Pesos,Bias]=backpropagation(pesos,bias,error_i,arquitectura,funciones,salida,factor_a)

    %Sensitividades %
    total_capas=size(funciones,2);
    S=cell(total_capas,1);
    derivada=derivada_funcion_activacion(funciones(1,total_capas),arquitectura(size(arquitectura,2)),salida{total_capas+1});
    S{total_capas}=-2*derivada*error_i;

    for capa=total_capas:2
        derivada=derivada_funcion_activacion(funciones(1,capa-1),arquitectura(1,capa),salida{capa});
        S{capa-1}=derivada*transpose(pesos{capa})*S{capa};
    end
    %Final de sensitividades%

    % Actualizando pesos y bias%
    for i=total_capas:2
        pesos{i}=pesos{i}-factor_a*S{i}*transpose(salida{i-1});
        bias{i}=bias{i}-factor_a*S{i};
    end
    %Finaliza actualizacion%
    Pesos=pesos;
    Bias=bias;
end
```

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

```
function S=derivada_funcion_activacion(opcion,n_neuronas,salida)
    Matriz=zeros(n_neuronas);

    if opcion==1
        for i=1:size(Matriz,1)
            for j=1:size(Matriz,2)
                Matriz(i,j)=1;
            end
        end
    end

    if opcion==2
        for i=1:size(Matriz,1)
            for j=1:size(Matriz,2)
                if i==j
                    Matriz(i,j)=salida(i)*(1-salida(i));
                end
            end
        end
    end

    if opcion==3
        for i=1:size(Matriz,1)
            for j=1:size(Matriz,2)
                if i==j
                    Matriz(i,j)=1-salida(i)^2;
```

```
                end
            end
        end
    end

    S=Matriz;
end
```

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

```
addpath("./Funciones/")

% 1a y 1b Pedir archivos%
archivo_entrada=input('Ingrese el nombre del archivo con los datos de entrada: ','s');
ruta='Archivos/';
archivo_entrada=strcat(ruta,archivo_entrada);

datos_entrada=importdata(archivo_entrada);
datos_grafica=datos_entrada;
[filas_entrada,columnas_entrada]=size(datos_entrada);

archivo_entrada=input('Ingrese el nombre del archivo con los targets: ','s');
ruta='Archivos/';
archivo_entrada=strcat(ruta,archivo_entrada);

targets=importdata(archivo_entrada);
[filas_target,columnas_target]=size(targets);
targets_grafica=targets;
%Fin de la peticion de entradas%

% 1c Se pide el rango de la señal%
limite_inferior=input('Limite inferior de la senal: ');
limite_superior=input('Limite superior de la senal: ');
if(limite_inferior<=limite_superior)
    rango=limite_inferior:1/(filas_entrada):limite_superior;
else
    rango=limite_superior:1/(filas_entrada):limite_inferior;
end
%Fin del modulo%

% 1d Pedir vectores de arquitectura%
vector_arquitectura=input('Vector de arquitectura: ');
[uno,tam_vector_arquitectura]=size(vector_arquitectura);
vector_funciones=input('Vector de funciones: ');
[uno,numero_capas]=size(vector_funciones);
%Fin del modulo 1d%

% 1e Pedir valor del factor de aprendizaje%
factor_aprendizaje=input('Ingrese el factor de aprendizaje: ');
%Fin del modulo 1e%

%1f valores de las condiciones de finalizacion%
epocas_totales=input('Numero de epocas: ');
error_epoch_max=input('Senal de error: ');
epoch_val=input('Cada cuantas iteraciones se llevara a cabo una epoca de validacion: ');
num_val=input('Numero maximo de incrementos consecutivos del error de epoca de validacion: ');
```

```
%Fin del modulo 1f%

%1g Division del dataset en 3%
disp('Escoja la opcion que guste para la division del dataset: ');
disp('1. 80%-10%-10%');
disp('2. 70%-15%-15%');
opcion=input(' ');
[conjunto_entrenamiento,conjunto_validacion,conjunto_prueba,target_entrenamiento,target_validacion,target_prueba]=separar_datos(datos_entrada,targets,opcion,filas_entrada);
[filas_validacion,columnas_validacion]=size(conjunto_validacion);
%Fin del modulo 1g%


% 2. Se inician los valores aleatorios entre -1 y 1%
pesos=cell(numero_capas,1);
bias=cell(numero_capas,1);
for i=1:tam_vector_arquitectura-1
    pesos(i,1)=-1+2*rand(vector_arquitectura(1,i+1),vector_arquitectura(1,i));
    bias(i,1)=-1+2*rand(vector_arquitectura(i+1),1);
end
%Final del modulo 2%

% Comenzando el entrenamiento %
error_epoca_val=zeros(100,1);
contador_validacion=2;
contador_val=0;
Salida_iteracion=0;
band=0;
eleccion=1;
while(eleccion==1)
    inicio=1;
    for epoca=inicio:epocas_totales

        if mod(epoca,num_val)==0
            suma_errores=0;
            for iteracion=1:size(conjunto_validacion,1)
                Salida_iteracion=feedFordward(pesos,conjunto_validacion(iteracion),bias,numero_capas,vector_funciones);
                suma_error=suma_error+(target_validacion(iteracion)-Salida_iteracion{numero_capas+1})^2;
            end
            error_epoca_val(contador_validacion)=suma_error/filas_entrada;

            if epoca>2
                if error_epoca_val(contador_validacion-1)<error_epoca_val(contador_validacion)
                    contador_val=contador_val+1;
                else
                    contador_val=0;
                end
            end

            if contador_val==num_val
                fprintf(1, '\nSalida por aumento de errores en validacion\n');
                band=1;
                break;
            end
                   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_ar
    quitectura,vector_funciones,Salida_iteracion,factor_aprendizaje);
112.                    end
113.                    fprintf(1,'----------------------');
114.                    Error=(suma_error/size(conjunto_entrenamiento,1))
115.
116.                    if Error<error_epoch_max
117.                        band=1;
118.                        fprintf(1,'Salida exitosa con el error %d\n',Error);
119.                        break;
120.                    end
121.
122.                    if epoca==epocas_totales
123.                        eleccion=input('Desea hacer mas iteraciones? 1. si 2 no');

124.                    end
125.                end
126.            end
127.
128.            if band==1
129.                break;
130.            end
131.        end
132.        % Fin del entrenamiento %
133.
134.        % Comprobacion %
135.        suma_error=0;
136.        for iteracion=1:size(conjunto_prueba,1)
137.            Salida_iteracion=feedFordward(pesos,conjunto_prueba(iteracion),bias,nu
    mero_capas,vector_funciones);
138.            error_it=(target_prueba(iteracion)-
    Salida_iteracion{numero_capas+1});
139.            suma_error=suma_error+error_it;
140.
141.        end
142.
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%
148.
149.
150.
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,nume
    ro_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

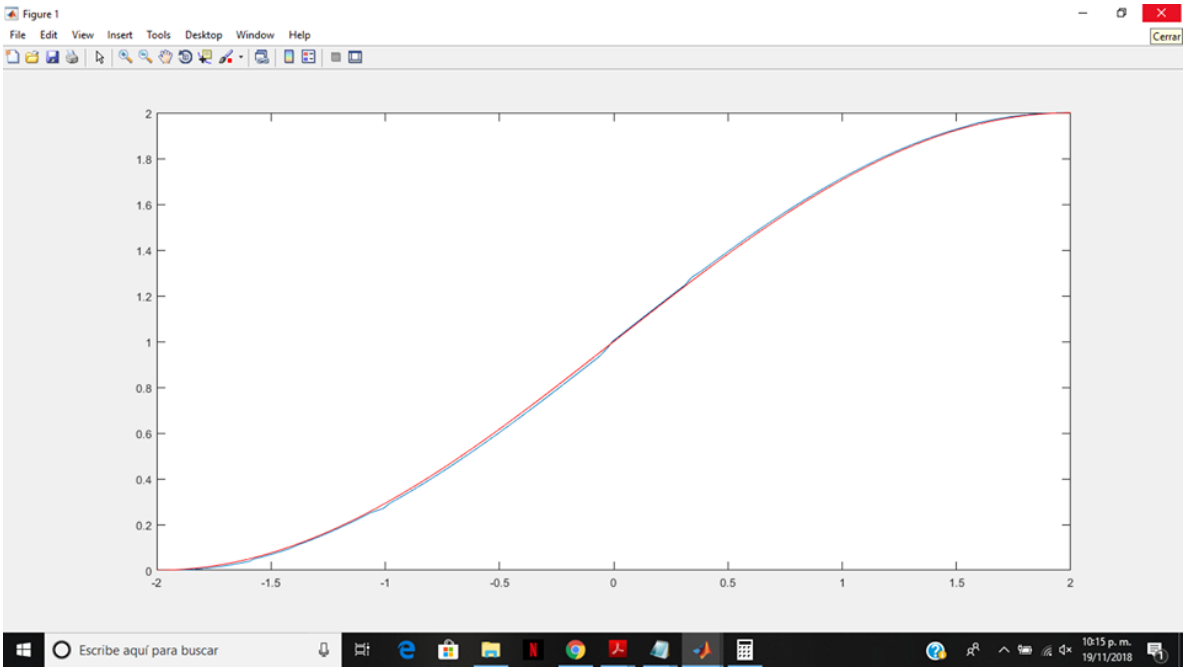

Figure 2 Gráfica de salida