Francisco Antonio Palos Angulo 132759

Implementación: Red neuronal multicapa para el reconocimiento de dígitos manuscritos.

Para este proyecto de redes neuronales tiene el objetivo de reconocer dígitos escritos a mano (de 0 a 9). El reconocimiento automático de dígitos escritos a mano se usa ampliamente en la actualidad, desde el reconocimiento de códigos postales (códigos postales) en sobres de correo hasta el reconocimiento de montos escritos en cheques bancarios. En esta parte del proyecto, usaremos extender la implementación anterior de regresión logística y la aplicaremos a la clasificación de uno contra todos.

El conjunto de datos DigitosProcesados.mat contiene 10000 ejemplos de entrenamiento de dígitos escritos a mano 1000 ejemplos por digito. El formato .mat significa que los datos se han guardado en un formato de matriz nativo Matlab, en lugar de un texto (ASCII) formal como un archivo csv. Estas matrices se pueden leer directamente en su programa usando el comando de load. Después de la carga, las matrices de las dimensiones y valores correctos aparecerán en la memoria de su programa. La matriz ya tendrá un nombre, por lo que no es necesario que les asigne nombres. Cada ejemplo de entrenamiento es una imagen en escala de grises de 20 píxeles por 20 píxeles del dígito. Cada píxel está representado por un número de punto flotante que indica la intensidad de la escala de grises en esa ubicación. La cuadrícula de píxeles de 20 por 20 se "desenrolla" en un vector de 400 dimensiones. Cada uno de estos ejemplos de entrenamiento se convierte en una sola fila en nuestra matriz de datos X. Esto nos da una matriz X de 10000 por 400 donde cada fila es un ejemplo de entrenamiento para una imagen de dígitos escrita a mano.

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

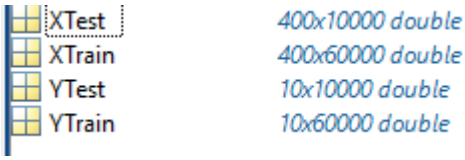
Primera parte del código:

```
%% Configuracion de parametros
entradas = 20*20;%resolucion de las imagenes
capadesalida = 10;
Neuronas = [entradas, 25, capadesalida];
L=numel(Neuronas); % Capas incluyendo la entrada y salida(Layers)
n=1; % Inicializaci´on del primer instante
%num_muestras = 60000; % N´umero de muestras de entrenamiento
num_muestras = 60000; % N´umero de muestras de entrenamiento
eta=0.0005; % Factor de aprendizaje
alpha=1; % Factor de olvido
a = 1; % Factor para funci´on de activaci´on
bias = 0.001;
%% Inicializacion de las entradas
load('DigitosProcesados.mat'); d = YTrain;
```

Ejemplo de dígitos en imagen.

Cargamos los datos generalizados donde se puede apreciar que tenemos 400 datos de entrada anterior mente mencionados debido a la resolución de las imágenes, una capa de salida 10 a raíz de los dígitos que queremos reconocer. Mediante el código se iba probando, se experimentó con una taza de aprendizaje del 0.005 para jugar con el error obtenido, ya que, al dar valores más grandes, se mantenía oscilando en bucle.

En esta parte también se carga la base de datos la cual trae diferentes tablas dentro del mismo archivo: YTest, XTRain, YTest, YTrain.

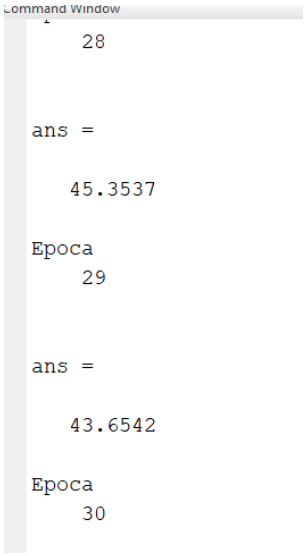


Antes de comenzar con el algoritmo, se establecieron presos aleatorios iniciales en la variable w.

```
%% Comienza el algoritmo
e{1,1}=1;
 for yy=1:50 %Probamos con 50 epocas
    orden = randperm(num_muestras);
%% =============== Inicia el corrido hacia adelante
    for i=1:num_muestras
        n = orden(i);
        ye{1} = XTrain(:, n);
        for l=2:L
            temp1=w{l}; temp2=ye{l-1};
            v{l} = temp1*temp2+bias*ones(Neuronas(l), 1);
            ye{l} = func_activ(a, v{l},l);
        end
        e{:, i} = d(:, n)-ye{L};
%% ==========  Inicia la retropropagacion
        for l=L:-1:1
            if l==L
                delta{l}=a*ye{l}.*(1.-ye{l}).*e{:,i};
            else
                delta{l} = a*ye{l}.*(1.-ye{l}).*(w{l+1}'*delta{l+1});
            end
        end
%% ============= Se retroalimentan los pesos
```

Estas dos partes dan inicio a las dos etapas principales de aprendizaje como los son el “feedfoward” y “backpropagation” la retropropagacion se basa en el parámetro delta.

El código retro alimenta al usuario visualizando la época del entrenamiento, dicho parámetro queda para ajustar.




```
%% Test de prueba
for n=1:num_pruebas
    yfinal{1} = XTest(:,n);
    for l=2:L
        vfinal{l,n} = w{l}*yfinal{l-1}+bias*ones(Neuronas(l), 1);
        yfinal{l} = func_activ(a, vfinal{l,n},l);
    end
    %% Calculo del error y salida final
    yn{n} = yfinal{L};
    dfinal{n} = round(yn{n});


end
```

En esta parte se trabaja con la tabla XTest de la base de datos para tener operar sobre las muestras de prueba.

```
%% Comprobacion de Salidas obtenidas contra esperadas
cont = 0;
for i=1:num_pruebas
    if(dfinal{i} == dt(:, i))
        r = ['Digito ', num2str(i), ' Bien'];
        cont = cont+1;
    else
        r = ['Digito ', num2str(i), ' Mal'];
        disp(r);
         q = round(4999*rand + 1);
         imshow(vec2mat(XTest(:,q),20)','InitialMagnification',300)
    end
end
r = ['Muestras correctas: ', num2str(cont)];
disp(r);
```

En esta parte final es donde podemos obtener el resultado de cuantas salidas exitosas y cuantas erróneas presenta el aprendizaje y entrenamiento.

Con los parámetros: `eta=0.0005, num_muestras = 60000, época=50`

Se obtuvieron los siguientes resultados:

Desde la primera época un error del:

```
ans =

   76.6710
```

%

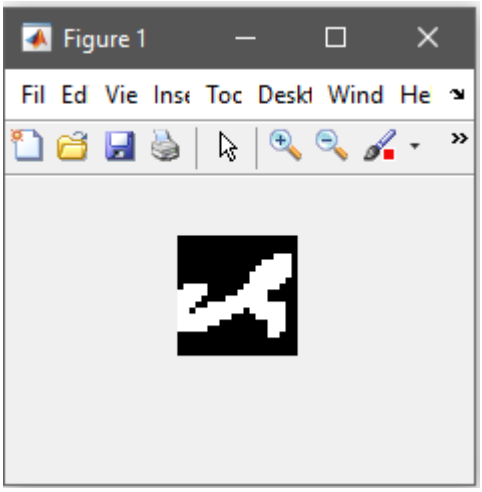
Hasta un error del:

```
ans =

   42.1827

Epoca
    50
```

```
Muestras correctas: 8656
```



Ejemplo de un resultado erróneo

- Con 1000 muestras se obtuvieron

```
50

Muestras correctas: 232
```

- Con 2000 muestras y 200 épocas

```
Muestras correctas: 5876
```

Conclusión: En esta implementación

, al tratar de jugar con el número de muestras se convierte un tanto ineficiente, dado que dependiendo de esta variable realmente es el tiempo a dar un resultado, también se desestabiliza el error ya que se toma de referencia el numero de muestras total. Se necesitó estar alterando la taza de aprendizaje para poder disminuir el error y no entrar en un bucle.