Práctica 4: Extreme Learning Machine
Modelos de la Computación

Emilio Gomez Esteban

1. Enunciado de la prática

Descarga el fichero ELM.zip del campus virtual. Para descomprimirlo, ejecuta en la línea de comandos de Matlab lo siguiente (unzip ELM.zip):

Aspectos a tener en cuenta:

1. El fichero comprimido contiene dos ficheros: la base de datos 'handwriting.mat' y el fichero de código 'P4_ELMStudents.m'.

2. La base de datos asociada a la práctica se conoce como MNIST de Dígitos reducida. A diferencia del MNIST original, que contiene 60000 imágenes de entrenamiento y 10000 imágenes de prueba de dígitos escritos a mano, esta versión reducida contiene 500 imágenes para cada dígito del 0 al 9, lo que suma un total de 5000 imágenes. Estas imágenes son una colección de dígitos escritos a mano.

3. Por la estructura de la base de datos, los 500 primeros números son el 0, los 500 siguientes el 1, y así sucesivamente. Al no incluirse la etiqueta de clase en la base de datos original, lo primero que hace el script es generar esa etiqueta de clase.

4. Después de generar las etiquetas de clase para nuestros datos, el siguiente paso en nuestro proceso es preparar los datos para entrenar y evaluar nuestros modelos de aprendizaje automático. En nuestro caso, en esta fase nos limitaremos a escalar los datos para dejarlos todos con el mismo rango.

5. Una vez que hemos escalado nuestros datos, creamos tres conjuntos de datos diferentes: el conjunto de entrenamiento (llamado 'TrainVal'), el conjunto de validación (llamado 'Val') y el conjunto de prueba (llamado 'Test'). Además de estos tres conjuntos, tenemos un cuarto conjunto llamado 'Train' que se crea incorporando parte de los datos del conjunto de validación al conjunto de entrenamiento (TrainVal).

El objetivo de esta práctica es implementar el modelo ELM (Extreme Learning Machine) neuronal en dos fases diferentes del código. La primera fase en la que implementamos el modelo ELM ocurre dentro de lo que llamamos 'validación anidada'. La validación anidada es una técnica que nos permite encontrar los mejores hiperparámetros para nuestro modelo. En este caso, estamos interesados en determinar los valores óptimos para dos hiperparámetros: C y L. Cada uno de estos hiperparámetros afecta cómo funciona el modelo ELM y es esencial encontrar los valores que maximizan su rendimiento.

En la segunda fase, utilizamos el modelo ELM con el conjunto completo de entrenamiento. En este punto, ya hemos encontrado los valores ideales para C y L gracias a la validación anidada. Ahora, el objetivo es entrenar el modelo con todo el conjunto de datos de entrenamiento. Una vez entrenado, podemos calcular dos métricas importantes: el Error Cuadrático Medio (MSE, por sus siglas en inglés) y la Tasa de Clasificación Correcta (CCR, por sus siglas en inglés) en el conjunto de prueba.

Parte optativa: Si logran implementar el modelo kernel propuesto en el artículo 'A multi-class classification model with parametrized target outputs for randomized-based feedforward neural networks', podrán subir un punto en su calificación del próximo parcial. Esta tarea es opcional y no es necesaria para aprobar la parte práctica, pero es una buena forma de mejorar su nota.

2. Resolución de la práctica

Hemos implementado las dos fases con el siguiente código:

```
clear all;

D = load('handwriting.mat');
X = D.X;

[N, K] = size(X);
J = 10;

Y = zeros(N,J);

% Generate the Y Label
for i =1:10
    Y(1+(500*(i-1)):i*500,i) =1;
end

% Scale the data
Xscaled = (X-min(X))./(max(X)-min(X));

% Remove the NaN elements
Xscaled = Xscaled(:,any(~isnan(Xscaled)));

% Compute again the number of total elements and attributes
[N, K] = size(Xscaled);

CVHO = cvpartition(N,'HoldOut',0.25);

XscaledTrain = Xscaled(CVHO.training(1),:);
XscaledTest = Xscaled(CVHO.test(1),:);
YTrain = Y(CVHO.training(1),:);
```

```
YTest = Y(CVHO.test(1),:);


% Create the validation set
[NTrain, K] = size(XscaledTrain);
CVHOV = cvpartition(NTrain,'HoldOut',0.25);

% Generate the validation sets
XscaledTrainVal = XscaledTrain(CVHOV.training(1),:);
XscaledVal = XscaledTrain(CVHOV.test(1),:);
YTrainVal = YTrain(CVHOV.training(1),:);
YVal = YTrain(CVHOV.test(1),:);

% Performance Matrix
Performance = zeros(7,6);

i = 0;
j = 0;
% Estimate the hyper-parameters values
for C = [10^(-3) 10^(-2) 10^(-1) 1 10 100 1000]
    i = i+1;
    for L = [50 100 500 1000 1500 2000]
        j = j+1;

        % Inicializar pesos aleatorios
        t = rand(L,K)*2 - 1;    % Pesos de entrada de tamanno
           (L x K)
        u = XscaledTrainVal*t';  % u de tamanno (NTrainVal x
           L)
        H = 1./(1 + exp(-u));  % Aplicacion de la funcion
           sigmoide, tamanno (L x NTrainVal)

        % Calcular los pesos de salida usando la ecuacion de
           ELM
        w = inv(eye(L)/C + H'*H)*H'*YTrainVal;

        % Salida para el conjunto de validacion
        Hval = 1./(1 + exp(-t * XscaledVal'));
        Yvale = Hval'*w;

        %Calcular el error cuadratico medio
        err = mean(mean((Yvale - YVal).^2));

        %Almacenar el rendimiento
        epsilon = 1.0e-9;
        Performance(i,j) = 1 /(err + epsilon);
```

```
72
73      end
74      j=0;
75  end
76
77  C = [10^(-3) 10^(-2) 10^(-1) 1 10 100 1000];
78  L = [50 100 500 1000 1500 2000];
79
80  [maxValue, linearIndexesOfMaxes] = max(Performance(:));
81  [rowsOfMaxes, colsOfMaxes] = find(Performance == maxValue);
82
83  Copt = C(rowsOfMaxes(1));
84  Lopt = L(colsOfMaxes(1));
85
86  % Calcular con el conjunto de entrenamiento el ELM neuronal y
87  % reportar el error cometido en test
88
89  t = rand(Lopt, K)*2 - 1;
90  u = XscaledTrain*t';
91  H = 1./(1 + exp(-u)); %Funcion sigmoide
92  w = inv(eye(Lopt)/Copt + H'*H)*H'*YTrain;
93
94  Htest = 1./(1 + exp(-t * XscaledTest'));
95  Yteste = Htest' * w;
96
97  % Calcular el error en el conjunto de test
98  errorTest = mean(mean((Yteste - YTest).^2));
99  disp(['Error en el conjunto de test (MSE): ', num2str(
       errorTest)]);
100
101 % Convertir predicciones a etiquetas de clase
102 [~, predictedLabels] = max(Yteste, [], 2);  % Obtener el
       indice del valor maximo en cada fila
103
104 % Convertir YTest a etiquetas de clase
105 [~, trueLabels] = max(YTest, [], 2);  % Obtener el indice del
        valor maximo en cada fila
106
107 % Calcular la Tasa de Clasificacion Correcta (CCR)
108 correctPredictions = sum(predictedLabels == trueLabels);
109 totalPredictions = length(trueLabels);
110 CCR = correctPredictions / totalPredictions;
111
112 disp(['Tasa de Clasificacion Correcta (CCR): ', num2str(CCR)
       ]);
```

En cuanto al calculo del error se utilizó el error cuadrático medio para evaluar el rendimiento en los conjuntos de validación y prueba. Un menor ECM indica que las predicciones del modelo están más cerca de los valores reales, lo que significa que el modelo tiene un mejor rendimiento. Esto significa que queremos encontrar los valores de los hiperparámetros que resulten en el menor error cuadrático medio en el conjunto de validación. Por tanto, lo que hemos hecho es inverter el ECM, para obtener el *Performance* y calcular para que valores de C y L obtenemos el máximo rendimiento. Sin embargo, hay que tener en cuenta que este enfoque puede tener problemas si el ECM se aproxima a cero (puedes encontrar divisiones por cero). Para ello, sumamos el error y un valor *epsilon* positivo muy pequeño. Finalmente, se almacenan los valores de rendimiento en la matriz *Performance* y luego se utiliza *max* para encontrar los hiperparámetros óptimos.

Obtenemos, por ejemplo, que $Copt = 0{,}1$ y que $Lopt = 2000$. Además, en la segunda fase, una vez entrenado el modelo con todo el conjunto de datos de entrenamiento, obtenemos (en el conjunto de prueba) que el Error en el conjunto de test (MSE): 0.01979 y la Tasa de Clasificación Correcta (CCR): 0.9488.

Para calcular el mejor rendimiento se pueden utilizar diferentes técnicas. Podemos seguir usando como métrica el error cuadrático medio, pero en vez de invertirlo, podemos almacenar los valores de rendimiento en la matriz *Performance* y luego utilizar *min* para encontrar los hiperparámetros óptimos. Podríamos implementar el siguiente código:

```
% Calcular el error cuadratico medio
    % Dentro de la validacion anidada
    err = mean(mean((Yvale - YVal).^2));

    % Almacenar el rendimiento
    Performance(i, j) = err;

% Fuera de la validacion anidada
[minValue, linearIndexesOfMins] = min(Performance(:));
[rowsOfMins, colsOfMins] = find(Performance == minValue);

Copt = C(rowsOfMins(1));  % Mejor C
Lopt = L(colsOfMins(1));  % Mejor L
```

Cambiando de métrica, podemos calcular la CCR, que se define como la proporción de predicciones correctas sobre el total de predicciones realizadas. La CCR se puede maximizar, y es una métrica intuitiva en problemas de clasificación.Se define como

$$CCR = \frac{PrediccionesCorrectas}{PrediccionesTotales}$$

En nuestro caso, después de hacer predicciones sobre el conjunto de validación, se puede calcular la CCR. En este caso, se podría haber procedido de la forma (en la validación

anidada):

```
% Calcular la Tasa de Clasificacion Correcta
[~, predictedClasses] = max(Yvale, [], 2);
[~, trueClasses] = max(YVal, [], 2);

correctPredictions = sum(predictedClasses == trueClasses);
totalPredictions = length(trueClasses);
ccr = correctPredictions / totalPredictions;  % Tasa de
   Clasificacion Correcta

% Almacenar el rendimiento en la matriz de performance
Performance(i, j) = ccr;  % Usar la CCR
```

Otra forma sería definir una nueva métrica que combine el ECM y otras métricas de rendimiento, como la CCR. Por ejemplo, podrías usar una combinación ponderada:

$$Performance = w_1 * CCR - w_2 * ECM,$$

donde w_1 y w_2 son pesos que se ajustan según la importancia de cada métrica.