MATLAB程序：ELM极速学习机

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/

来源：Extreme Learning Machines
http://www.ntu.edu.sg/home/egbhuang/elm_codes.html

更新：只做科普，别做学术。

2004年南洋理工大学黄广斌提出了ELM算法。极限学习机（ELM Extreme Learning Machine)是一种快速的的单隐层神经网络（SLFN）训练算法。**该算法的特点是在网络参数的确定过程中,隐层节点参数随机选取,在训练过程中无需调节，只需要设置隐含层神经元的个数，便可以获得唯一的最优解;而网络的外权（即输出权值）是通过最小化平方损失函数得到的最小二乘解（最终化归成求解一个矩阵的 Moore-Penrose 广义逆问题）.这样网络参数的确定过程中无需任何迭代步骤,从而大大降低了网络参数的调节时间。与传统的训练方法相比，该方法具有学习速度快、泛化性能好等优点。**

参考： [Extreme Learning Machine]

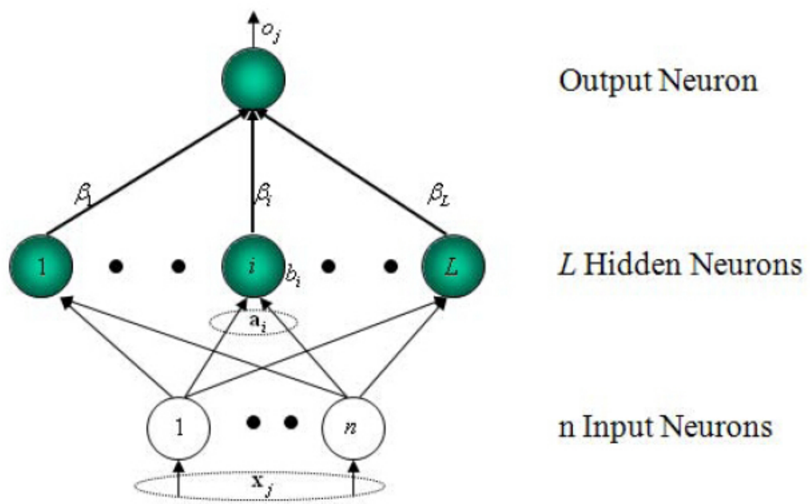

Figure 1: SLFN: additive hidden nodes

表 2.1　ELM 算法

输入	样本集 $\aleph=\left\{(\mathbf{x}_i,\mathbf{y}_i)\middle\|\mathbf{x}_i\in\Re^n, \mathbf{y}_i\in\Re^m, i=1,\ldots,N\right\}$，激励函数 $G(\mathbf{x})$，隐藏层节点数为 $\hat{N}$
输出	输出权值 $\boldsymbol{\beta}$
步骤 1	随机设计输入权值 $\boldsymbol{a}_i$ 和偏置 b_i。
步骤 2	计算隐藏层输出矩阵 $\mathbf{H}$
步骤 3	计算输出权值 $\boldsymbol{\beta}$，$\boldsymbol{\beta}=\mathbf{H}^{+}\mathbf{Y}$ ，$\mathbf{Y}=[\mathbf{y}_1,\ldots,\mathbf{y}_N]^T$

ELM的MATLAB算法：

```
function [TrainingTime, TestingTime, TrainingAccuracy, TestingAccuracy] = elm(TrainingData_File, TestingData_File, Elm_Type, NumberofHiddenNeurons, ActivationFunction)

% Usage: elm(TrainingData_File, TestingData_File, Elm_Type, NumberofHiddenNeurons, ActivationFunction)
% OR:    [TrainingTime, TestingTime, TrainingAccuracy, TestingAccuracy] = elm(TrainingData_File, TestingData_File, Elm_Type, NumberofHiddenNeurons, ActivationFunction)
%
% Input:
% TrainingData_File     - Filename of training data set
% TestingData_File      - Filename of testing data set
% Elm_Type              - 0 for regression; 1 for (both binary and multi-classes) classification
% NumberofHiddenNeurons - Number of hidden neurons assigned to the ELM
% ActivationFunction    - Type of activation function:
%                           'sig' for Sigmoidal function
%                           'sin' for Sine function
%                           'hardlim' for Hardlim function
%                           'tribas' for Triangular basis function
%                           'radbas' for Radial basis function (for additive type of SLFNs instead of RBF type of SLFNs)
%
% Output:
% TrainingTime          - Time (seconds) spent on training ELM
% TestingTime           - Time (seconds) spent on predicting ALL testing data
% TrainingAccuracy      - Training accuracy:
%                           RMSE for regression or correct classification rate for classification
% TestingAccuracy       - Testing accuracy:
%                           RMSE for regression or correct classification rate for classification
%
% MULTI-CLASSE CLASSIFICATION: NUMBER OF OUTPUT NEURONS WILL BE AUTOMATICALLY SET EQUAL TO NUMBER OF CLASSES
% FOR EXAMPLE, if there are 7 classes in all, there will have 7 output
% neurons; neuron 5 has the highest output means input belongs to 5-th class
%
% Sample1 regression: [TrainingTime, TestingTime, TrainingAccuracy, TestingAccuracy] = elm('sinc_train', 'sinc_test', 0, 20, 'sig')
% Sample2 classification: elm('diabetes_train', 'diabetes_test', 1, 20, 'sig')
%
    %%%%    Authors:    MR QIN-YU ZHU AND DR GUANG-BIN HUANG
    %%%%    NANYANG TECHNOLOGICAL UNIVERSITY, SINGAPORE
    %%%%    EMAIL:      EGBHUANG@NTU.EDU.SG; GBHUANG@IEEE.ORG
    %%%%    WEBSITE:    http://www.ntu.edu.sg/eee/icis/cv/egbhuang.htm
    %%%%    DATE:       APRIL 2004

%%%%%%%%%%% Macro definition
REGRESSION=0;
CLASSIFIER=1;

%%%%%%%%%%% Load training dataset
train_data=load(TrainingData_File);
T=train_data(:,1)';
P=train_data(:,2:size(train_data,2))';
clear train_data;                                   %   Release raw training data array

%%%%%%%%%%% Load testing dataset
test_data=load(TestingData_File);
TV.T=test_data(:,1)';
TV.P=test_data(:,2:size(test_data,2))';
clear test_data;                                    %   Release raw testing data array
```

```

NumberofTrainingData=size(P,2);
NumberofTestingData=size(TV.P,2);
NumberofInputNeurons=size(P,1);

if Elm_Type~=REGRESSION
    %%%%%%%%%%%% Preprocessing the data of classification
    sorted_target=sort(cat(2,T,TV.T),2);
    label=zeros(1,1);                               %   Find and save in 'label' class label from training and testing data sets
    label(1,1)=sorted_target(1,1);
    j=1;
    for i = 2:(NumberofTrainingData+NumberofTestingData)
        if sorted_target(1,i) ~= label(1,j)
            j=j+1;
            label(1,j) = sorted_target(1,i);
        end
    end
    number_class=j;
    NumberofOutputNeurons=number_class;

    %%%%%%%%%% Processing the targets of training
    temp_T=zeros(NumberofOutputNeurons, NumberofTrainingData);
    for i = 1:NumberofTrainingData
        for j = 1:number_class
            if label(1,j) == T(1,i)
                break;
            end
        end
        temp_T(j,i)=1;
    end
    T=temp_T*2-1;

    %%%%%%%%%% Processing the targets of testing
    temp_TV_T=zeros(NumberofOutputNeurons, NumberofTestingData);
    for i = 1:NumberofTestingData
        for j = 1:number_class
            if label(1,j) == TV.T(1,i)
                break;
            end
        end
        temp_TV_T(j,i)=1;
    end
    TV.T=temp_TV_T*2-1;

end                                                 %   end if of Elm_Type

%%%%%%%%%%% Calculate weights & biases
start_time_train=cputime;

%%%%%%%%%%% Random generate input weights InputWeight (w_i) and biases BiasofHiddenNeurons (b_i) of hidden neurons
InputWeight=rand(NumberofHiddenNeurons,NumberofInputNeurons)*2-1;
BiasofHiddenNeurons=rand(NumberofHiddenNeurons,1);
tempH=InputWeight*P;
clear P;                                            %   Release input of training data
ind=ones(1,NumberofTrainingData);
BiasMatrix=BiasofHiddenNeurons(:,ind);              %   Extend the bias matrix BiasofHiddenNeurons to match the demention of H
tempH=tempH+BiasMatrix;

%%%%%%%%%%% Calculate hidden neuron output matrix H
switch lower(ActivationFunction)
    case {'sig','sigmoid'}
        %%%%%%%% Sigmoid
        H = 1 ./ (1 + exp(-tempH));
    case {'sin','sine'}
        %%%%%%%% Sine
        H = sin(tempH);
    case {'hardlim'}
        %%%%%%%% Hard Limit
        H = double(hardlim(tempH));
    case {'tribas'}
        %%%%%%%% Triangular basis function
```

```
        H = tribas(tempH);
    case {'radbas'}
        %%%%%%%% Radial basis function
        H = radbas(tempH);
        %%%%%%%% More activation functions can be added here
end
clear tempH;                                        %   Release the temparary array for calculation of hidden neuron output matrix H

%%%%%%%%%%% Calculate output weights OutputWeight (beta_i)
OutputWeight=pinv(H') * T';                        % implementation without regularization factor //refer to 2006 Neurocomputing paper
%OutputWeight=inv(eye(size(H,1))/C+H * H') * H * T';   % faster method 1 //refer to 2012 IEEE TSMC-B paper
%implementation; one can set regularizaiton factor C properly in classification applications
%OutputWeight=(eye(size(H,1))/C+H * H') \ H * T';      % faster method 2 //refer to 2012 IEEE TSMC-B paper
%implementation; one can set regularizaiton factor C properly in classification applications

%If you use faster methods or kernel method, PLEASE CITE in your paper properly:

%Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang, "Extreme Learning Machine for Regression and Multi-Class Classification," submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, Octob

end_time_train=cputime;
TrainingTime=end_time_train-start_time_train        %   Calculate CPU time (seconds) spent for training ELM

%%%%%%%%%%% Calculate the training accuracy
Y=(H' * OutputWeight)';                             %   Y: the actual output of the training data
if Elm_Type == REGRESSION
    TrainingAccuracy=sqrt(mse(T - Y))               %   Calculate training accuracy (RMSE) for regression case
end
clear H;

%%%%%%%%%%% Calculate the output of testing input
start_time_test=cputime;
tempH_test=InputWeight*TV.P;
clear TV.P;             %   Release input of testing data
ind=ones(1,NumberofTestingData);
BiasMatrix=BiasofHiddenNeurons(:,ind);              %   Extend the bias matrix BiasofHiddenNeurons to match the demention of H
tempH_test=tempH_test + BiasMatrix;
switch lower(ActivationFunction)
    case {'sig','sigmoid'}
        %%%%%%%% Sigmoid
        H_test = 1 ./ (1 + exp(-tempH_test));
    case {'sin','sine'}
        %%%%%%%% Sine
        H_test = sin(tempH_test);
    case {'hardlim'}
        %%%%%%%% Hard Limit
        H_test = hardlim(tempH_test);
    case {'tribas'}
        %%%%%%%% Triangular basis function
        H_test = tribas(tempH_test);
    case {'radbas'}
        %%%%%%%% Radial basis function
        H_test = radbas(tempH_test);
        %%%%%%%% More activation functions can be added here
end
TY=(H_test' * OutputWeight)';                       %   TY: the actual output of the testing data
end_time_test=cputime;
TestingTime=end_time_test-start_time_test           %   Calculate CPU time (seconds) spent by ELM predicting the whole testing data

if Elm_Type == REGRESSION
    TestingAccuracy=sqrt(mse(TV.T - TY))            %   Calculate testing accuracy (RMSE) for regression case
end

if Elm_Type == CLASSIFIER
%%%%%%%%%% Calculate training & testing classification accuracy
    MissClassificationRate_Training=0;
    MissClassificationRate_Testing=0;

    for i = 1 : size(T, 2)
        [x, label_index_expected]=max(T(:,i));
        [x, label_index_actual]=max(Y(:,i));
        if label_index_actual~=label_index_expected
```

```
            MissClassificationRate_Training=MissClassificationRate_Training+1;
        end
    end
    TrainingAccuracy=1-MissClassificationRate_Training/size(T,2)
    for i = 1 : size(TV.T, 2)
        [x, label_index_expected]=max(TV.T(:,i));
        [x, label_index_actual]=max(TY(:,i));
        if label_index_actual~=label_index_expected
            MissClassificationRate_Testing=MissClassificationRate_Testing+1;
        end
    end
    TestingAccuracy=1-MissClassificationRate_Testing/size(TV.T,2)
end
```

ELM

我对ELM算法的解读

function [TrainingTime, TestingTime, TrainingAccuracy, TestingAccuracy] = elm (TrainingData_File, TestingData_File, Elm_Type, NumberofHiddenNeurons, ActivationFunction)

%输入：TrainingData_File（训练数据集的文件名），TestingData_File（测试数据集的文件名），Elm_Type有两个值：0为回归，1为（二进制和多类的）分类，NumberofHiddenNeurons（被分配到elm的隐层神经元数目），ActivationFunction（激活函数）
输出：训练时间（花费在训练elm上的时间（秒）），测试时间（用于预测所有测试数据），训练精度（回归或分类正确率的分类误差），测试精度（回归或分类正确率的分类误差）
例如：1回归：[TrainingTime, TestingTime, TrainingAccuracy, TestingAccuracy] = elm ('sinc_train '，'sinc_test’，0，20，“sig”）
2分类：[TrainingTime, TestingTime, TrainingAccuracy, TestingAccuracy] = elm（'diabetes_train '，'diabetes_test’，1，20，'sig'）

%宏定义
REGRESSION=0; **%回归为0**
CLASSIFIER=1; **%分类为1**

```
%导入训练数据集
train_data=load(TrainingData_File);
T=train_data(:,1)';   %将train_data第一列转置为行向量
P=train_data(:,2:size(train_data,2))';   %将train_data第2到最后一列
转置为行向量
clear train_data;    %释放原训练数据矩阵
%  导入测试数据集
test_data=load(TestingData_File);
TV.T=test_data(:,1)';
TV.P=test_data(:,2:size(test_data,2))';
clear test_data;                                   %释放原测试数据矩阵


NumberofTrainingData=size(P,2);         %P的列数
NumberofTestingData=size(TV.P,2);      %TV.P的列数
NumberofInputNeurons=size(P,1);        %P的行数
```

if Elm_Type~=REGRESSION （求训练数据集单隐层前向神经网络的输出函数）

```
    %分类数据的预处理（离散）
    sorted_target=sort(cat(2,T,TV.T),2);  %cat(2,T,TV.T)=[T,TV.T]，对[T,TV.T]总数据集中第一行进行排序
    label=zeros(1,1);  %为寻找和保存'label'类标签的训练和测试数据集
    label(1,1)=sorted_target(1,1);   j=1;
    for i = 2:(NumberofTrainingData+NumberofTestingData)
        if sorted_target(1,i) ~= label(1,j)
            j=j+1;   label(1,j) = sorted_target(1,i);
        end
    end
number_class=j;
NumberofOutputNeurons=number_class;
```

将第一行分类（相同的为一类）

```
  %处理训练目标
 temp_T=zeros(NumberofOutputNeurons, NumberofTrainingData);
    for i = 1:NumberofTrainingData
        for j = 1:number_class
            if label(1,j) == T(1,i)
                break;       %结束循环体
            end
        end
      temp_T(j,i)=1;
    end
    T=temp_T*2-1;
%T：训练数据的实际输入
```

```
 %处理测试目标
temp_TV_T=zeros(NumberofOutputNeurons, NumberofTestingData);
   for i = 1:NumberofTestingData
      for j = 1:number_class
         if label(1,j) == TV.T(1,i)
            break;
         end
      end
      temp_TV_T(j,i)=1;
   end
   TV.T=temp_TV_T*2-1;
end                                        %如果elm_type结束
%计算权值和偏差
start_time_train=cputime;
%随机生成隐层节点参数：网络输入与隐层之间的内权（输入权值InputWeight）和偏置BiasofHiddenNeurons
InputWeight=rand(NumberofHiddenNeurons,NumberofInputNeurons)*2-1;
%权值：-1到1
BiasofHiddenNeurons=rand(NumberofHiddenNeurons,1);
tempH=InputWeight*P; %输入权值与输入的训练集P的内积
clear P;  %释放输入的训练集
```

```
ind=ones(1,NumberofTrainingData);
BiasMatrix=BiasofHiddenNeurons(:,ind);%将隐层神经元的偏置匹配矩阵H的要求
tempH=tempH+BiasMatrix;    %针对的是加法型的隐层节点
%以下计算隐层神经元输出矩阵H
switch lower(ActivationFunction)
    case {'sig','sigmoid'}
        % Sigmoid 函数
        H = 1 ./ (1 + exp(-tempH));
    case {'sin','sine'}
        % 正弦函数
        H = sin(tempH);
    case {'hardlim'}
        % 硬限制传递函数
        H = double(hardlim(tempH));
    case {'tribas'}
        % 三角基函数
        H = tribas(tempH);
    case {'radbas'}
        % 径向量基函数
        H = radbas(tempH);
end
clear tempH;   %释放隐藏神经元的输出矩阵计算的阵列
%H：隐层节点的输出
```

将tempH代入指定的激活函数中

```
%计算连接隐层与网络输出之间的外权（输出权值OutputWeight ）
OutputWeight=pinv(H') * T';  %无正则化因子，pinv求伪逆
end_time_train=cputime;
TrainingTime=end_time_train-start_time_train
%计算用于训练的ELM时CPU的时间（秒）
%以下计算训练精度
Y=(H' * OutputWeight)';      % Y：训练数据的实际输出 T：训练数据的实际输入
 if Elm_Type == REGRESSION
   TrainingAccuracy=sqrt(mse(T - Y))    %计算回归案例的训练精度（RMSE）
%mse是均方误差函数（用于检验神经网络算法的误差分析）
end
clear H;   %H：隐层节点的输出
%计算测试的输入输出
start_time_test=cputime;
tempH_test=InputWeight*TV.P;
clear TV.P;             %释放测试数据的输入
ind=ones(1,NumberofTestingData);
BiasMatrix=BiasofHiddenNeurons(:,ind); %扩展隐层神经元的偏置匹配矩阵H的要求
tempH_test=tempH_test + BiasMatrix;     %针对的是加法型的隐层节点
```

```
switch lower(ActivationFunction)
    case {'sig','sigmoid'}
            H_test = 1 ./ (1 + exp(-tempH_test));
    case {'sin','sine'}
            H_test = sin(tempH_test);
    case {'hardlim'}
             H_test = hardlim(tempH_test);
    case {'tribas'}
             H_test = tribas(tempH_test);
    case {'radbas'}
             H_test = radbas(tempH_test);
end
TY=(H_test' * OutputWeight)';                    %   TY: 试验数据的实际输出
end_time_test=cputime;
TestingTime=end_time_test-start_time_test    %计算ELM预测整个测试数据消耗的CPU时间（秒）
if Elm_Type == REGRESSION
    TestingAccuracy=sqrt(mse(TV.T - TY))      %计算回归案例的测试精度（RMSE）
end
if Elm_Type == CLASSIFIER
%计算训练和测试分类精度
  MissClassificationRate_Training=0;  %初始化训练集误分类比率
  MissClassificationRate_Testing=0;   %初始化测试集误分类比率
```

```
 for i = 1 : size(T, 2)
        [x, label_index_expected]=max(T(:,i));
        [x, label_index_actual]=max(Y(:,i));
        if label_index_actual~=label_index_expected
            MissClassificationRate_Training=MissClassificationRate_Training+1;
        end
    end
    TrainingAccuracy=1-MissClassificationRate_Training/size(T,2)
    for i = 1 : size(TV.T, 2)
        [x, label_index_expected]=max(TV.T(:,i));
        [x, label_index_actual]=max(TY(:,i));
        if label_index_actual~=label_index_expected
            MissClassificationRate_Testing=MissClassificationRate_Testing+1;
        end
    end
    TestingAccuracy=1-MissClassificationRate_Testing/size(TV.T,2)
end
```

这是我很久之前写的神经网络ELM代码及解读，只有知道ELM的数学原理，再看程序，才会有所理解。如有不足之处，望指点。