http://www.pudn.com/Download/item/id/2667412.html

代码下载地址：http://download.csdn.net/download/love\_ljq/10189737

　　最近在做一件觉得挺有意思的事情，而这件事情的基础就是要先用Matlab底层代码实现深度学习。之前自己没有写过，所以就找了LeNet这个最简单的网络。这里要着重说明的是，我用Matlab底层代码实现的是测试部分，并没有实现参数训练部分，这里的网络模型和网络参数都是已经训练好的。

　　深度学习的网络操作，主要来讲就那么几种，卷积、全连接、激活(ReLU)、Softmax，大的网络差不多也是由这么一些基础部件搭起来的，而LeNet这个小麻雀完全包含了这些五脏六腑，相对于Caffe一句话的实现，用底层代码实现还是要复杂的多。

　　首先我们来看一下LeNet这个网络的结构。

　　上面的这个网络结构图是pyCaffe通过draw\_net.py文件，根据lenet.prototxt进行绘制的，从图中可以很直观地看出LeNet的网络结构。Convolution层，MaxPooling层，Convolution层，MaxPooling层，然后是InnerProduct，ReLU，InnerProduct，Softmax。

　　除了网络模型，我们还需要知道网络每一层的参数，这些参数也是通过pyCaffe来读取的并且保存为.mat文件以供Matlab进行读取。LeNet的参数都是浮点型的，第一层的卷积层的参数量为(20\*5\*5+20)=520个，其中20个是偏置，其第二层卷积层的参数量为(50\*20\*5\*5+50)=520个，第三层内积层的参数量为(500\*800+500)=40500个，第四层内积层的参数量为(10\*500+10)=5010个，总计为71080个。

　　咱们之间来看一下代码吧。

% 首先进行参数的读取，一共是8个文件，每一层需要参数的操作对应一个w和一个b

load('.\parameters\conv1\_w.mat');

load('.\parameters\conv1\_b.mat');

load('.\parameters\conv2\_w.mat');

load('.\parameters\conv2\_b.mat');

load('.\parameters\ip1\_w.mat');

load('.\parameters\ip1\_b.mat');

load('.\parameters\ip2\_w.mat');

load('.\parameters\ip2\_b.mat');

% 读取图像并将图像像素值除以255以归一化到(0,1)，图像的尺寸为28x28，单通道

image = double(imread('1.jpg'))/255;

% 第一层卷积层操作，这里的convolution是我自己写的函数

% 因为感觉Matlab自带的函数不好用就自己写了一个，这个子函数会提供源代码

% 关于卷积操作的网络应该有很多资料的

% 输入为我们的图像，大小为28x28，经过20个5x5卷积核的卷积，输出为24x24x20

A = zeros(24,24,20);

for i = 1:20

A(:,:,i) = convolution(image,conv1\_w(i,1,:,:)) + conv1\_b(i);

end

% MaxPooling层，就是相邻四个像素取极大值，输出为12x12x20

B = zeros(12,12,20);

for i = 1:20

for row = 1:12

for col = 1:12

B(row,col,i) = max(max(A(2\*row-1:2\*row,2\*col-1:2\*col,i)));

end

end

end

% 第三层convolution层，输出为8x8x50

C = zeros(8,8,50);

for i = 1:50

for j = 1:20

C(:,:,i) = C(:,:,i) + convolution(B(:,:,j),conv2\_w(i,j,:,:));

end

C(:,:,i) = C(:,:,i) + conv2\_b(i);

end

% 第四层MaxPooling层

D = zeros(4,4,50);

for i = 1:50

for row = 1:4

for col = 1:4

D(row,col,i) = max(max(C(2\*row-1:2\*row,2\*col-1:2\*col,i)));

end

end

end

% 首先将4x4x500的图像展成1x800的图像与参数进行匹配，再进行矩阵乘法并加上偏置

DRow = zeros(1,800);

for index = 1:50

DRow(index\*16-15:index\*16) = [D(1,1:4,index),D(2,1:4,index),D(3,1:4,index),D(4,1:4,index)];

end

E = ip1\_w \* DRow' + ip1\_b';

% ReLU层，选择正数，抛弃负数部分

F = (E>0).\*E;

% 内积点乘

G = ip2\_w\*F + ip2\_b';

% 最后的SoftMax层，这里的softMax也是我自己写的函数

[~,result] = max(softMax(G'));

% 结果显示

disp(['The classification Result is ',num2str(result-1)]);

】

　　我们将softmax之后的结果与python的计算结果进行对比，发现两者基本相同，但是因为python和matlab在中间结果等精度不一致，导致两者有约万分之一的差。

　　最后是实际实验，我们测试了几张手写识别图片，都得到了正确的结果，毕竟LeNet的精度大于99%。

附

% convolution function source code

% 因为比较简单，也没有去优化，直接用了多重for循环

function result = convolution(image,core)

[imageSize,~] = size(image);

coreSize = 5;

result = zeros(imageSize-(coreSize-1));

for rowIndex = 1:size(result,1)

for colIndex = 1:size(result,1)

for x = 1:5

for y = 1:5

result(rowIndex,colIndex) = result(rowIndex,colIndex) + image(rowIndex + x - 1, colIndex + y - 1)\*double(core(1,1,x,y));

end

end

end

end

end

% softmax function source code

function softMaxResult = softMax(Array)

[~,cols] = size(Array);

ArrayCal = zeros(3,cols);

ArrayCal(1,:) = exp(Array);

sumOfArray = sum(ArrayCal(1,:));

ArrayCal(2,:) = ArrayCal(1,:)/sumOfArray;

softMaxResult = ArrayCal(2,:);

end