**卷积神经网络CNN原理——结合实例matlab实现**

卷积神经网络CNN是深度学习的一个重要组成部分，由于其优异的学习性能（尤其是对图片的识别）。近年来研究异常火爆，出现了很多模型LeNet、Alex net、ZF net等等。由于大多高校在校生使用matlab比较多，而网上的教程代码基本都基于caffe框架或者python，对于新入门的同学来说甚是煎熬，所以本文采用matlab结合MNIst手写数据库完成对手写数字的识别。本人水平有限，如有纰漏，还望各路大神，帮忙指正。   
一、卷积网络原理

1、动机

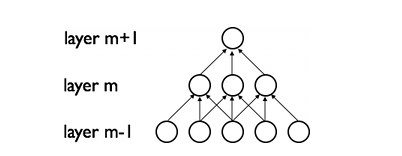
卷积神经网络（CNN）是多层感知机（MLP）的一个变种模型，它是从生物学概念中演化而来的。从Hubel和Wiesel早期对猫的视觉皮层的研究工作，我们知道在视觉皮层存在一种细胞的复杂分布，，这些细胞对于外界的输入局部是很敏感的，它们被称为“感受野”（细胞），它们以某种方法来覆盖整个视觉域。这些细胞就像一些滤波器一样，它们对输入的图像是局部敏感的，因此能够更好地挖掘出自然图像中的目标的空间关系信息。

此外，视觉皮层存在两类相关的细胞，S细胞（Simple Cell）和C（Complex Cell）细胞。S细胞在自身的感受野内最大限度地对图像中类似边缘模式的刺激做出响应，而C细胞具有更大的感受野，它可以对图像中产生刺激的模式的空间位置进行精准地定位。

视觉皮层作为目前已知的最为强大的视觉系统，广受关注。学术领域出现了很多基于它的神经启发式模型。比如：NeoCognitron [Fukushima], HMAX [Serre07] 以及本教程要讨论的重点 LeNet-5 [LeCun98]。

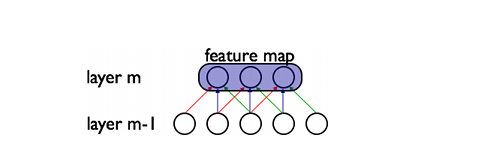
2、稀疏连接

CNNs通过加强神经网络中相邻层之间节点的局部连接模式（Local Connectivity Pattern）来挖掘自然图像（中的兴趣目标）的空间局部关联信息。第m层隐层的节点与第m-1层的节点的局部子集，并具有空间连续视觉感受野的节点（就是m-1层节点中的一部分，这部分节点在m-1层都是相邻的）相连。可以用下面的图来表示这种连接。

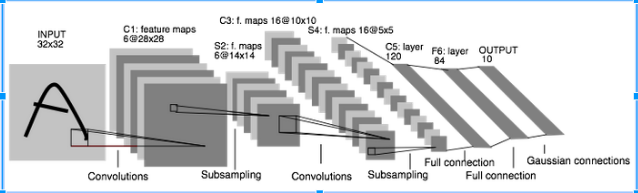
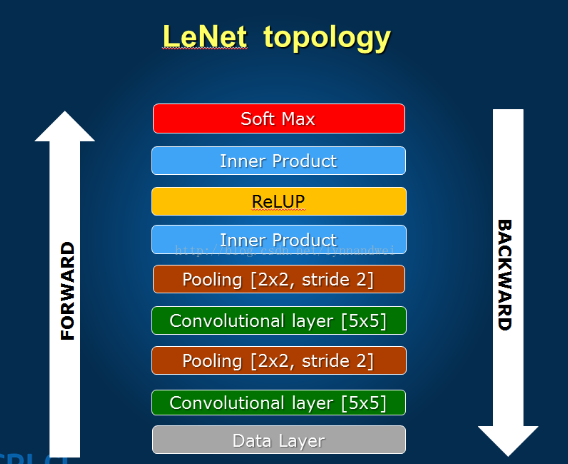
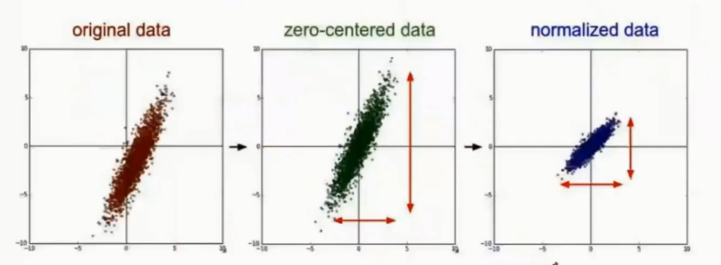
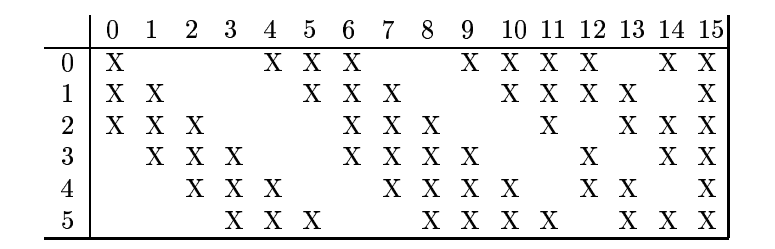


假设，m-1层为视网膜输入层（接受自然图像）。根据上图的描述，在m-1层上面的m层的神经元节点都具有宽度为3的感受野，m层每一个节点连接下面的视网膜层的3个相邻的节点。m+1层的节点与它下面一层的节点有着相似的连接属性，所以m+1层的节点仍与m层中3个相邻的节点相连，但是对于输入层（视网膜层）连接数就变多了，在本图中是5。这种结构把训练好的滤波器（corresponding to the input producing the strongest response）构建成了一种空间局部模式（因为每个上层节点都只对感受野中的，连接的局部的下层节点有响应）。根据上面图，多层堆积形成了滤波器（不再是线性的了），它也变得更具有全局性了（如包含了一大片的像素空间）。比如，在上图中，第m+1层能够对宽度为5的非线性特征进行编码（就像素空间而言）。

3、权值共享

在CNNs中，每一个稀疏滤波器hi在整个感受野中是重复叠加的，这些重复的节点形式了一种特征图（feature map）,这个特种图可以共享相同的参数，比如相同的权值矩阵和偏置向量。   


在上图中，属于同一个特征图的三个隐层节点，因为需要共享相同颜色的权重, 他们的被限制成相同的。在这里， 梯度下降算法仍然可以用来训练这些共享的参数，只需要在原算法的基础上稍作改动即可。共享权重的梯度可以对共享参数的梯度进行简单的求和得到。

二、网络的分析   
上面这些内容，基本就是CNN的精髓所在了，下面结合LeNet做具体的分析。   
结构图：   
   
   
LeNet算上输入输出一共为八层，下面逐层分析。   
第一层：数据输入层   
CNN的强项在于图片的处理，lenet的输入为32\*32的矩阵图片。这里需要注意的点：   
1、数据的归一化，这里的归一化是广义的，不一定要归到0-1，但要是相同的一个区间范围，一般我们的灰度图为0-255。   
2、数据的去均值，如果样本有非零的均值，而且与测试部分的非零均值不一致，可能就会导致识别率的下降。当然这不一定发生，我们这么做是为了增加系统的鲁棒性。   
   
第二层：卷积层c1   
卷积层是卷积神经网络的**核心**，通过不同的卷积核，来获取图片的特征。卷积核相当于一个滤波器，不同的滤波器提取不同特征。打个比方，对于手写数字识别，某一个卷积核提取‘一’，另一个卷积核提取‘|’，所以这个数字很有可能就判定为‘7’。当然实际要比这复杂度得多，但原理大概就是这个样子。   
第三层：pooling层   
基本每个卷积层后边都会接一个pooling层，目的是为了降维。一般都将原来的卷积层的输出矩阵大小变为原来的一半，方便后边的运算。另外，pooling层增加了系统的鲁棒性，把原来的准确描述变为了概略描述（原来矩阵大小为28\*28，现在为14\*14，必然有一部分信息丢失，一定程度上防止了过拟合）。   
第四层：卷积层   
与之前类似，在之前的特征中进一步提取特征，对原样本进行更深层次的表达。注意：这里不是全连接。这里不是全连接。这里不是全连接。X代表连接，空白代表不连。   
   
第五层：pooling层   
与之前类似。   
第六层：卷积层（全连接）   
这里有120个卷积核，**这里是全连接的**。将矩阵卷积成一个数，方便后边网络进行判定。   
第七层：全连接层   
和MLP中的隐层一样，获得高维空间数据的表达。   
第八层：输出层   
这里一般采用RBF网络，每个RBF的中心为每个类别的标志，网络输出越大，代表越不相似，输出的最小值即为网络的判别结果。   
三、卷积网络的BP训练   
前面的都很好理解，卷积神经网络的难度在于BP过程。网上zouxy09的博文写的很好，可以看一下，自己搞明白。传送门：[CNN的BP推导](http://blog.csdn.net/zouxy09/article/details/9993371)   
四、代码部分   
关于MNIST数据集，网上有很多现成的代码对其进行提取，但提取出来的都是乱序的很不利于使用。这里有提取好的分类后的，[详情传送门](http://blog.sina.com.cn/s/blog_6981b2bf0100pxxm.html)   
简单起见，我们的代码选用一层卷积层。

%%% matlab实现LeNet-5

%%% 作者：xd.wp

%%% 时间：2016.10.22 14:29

%% 程序说明

% 1、池化（pooling）采用平均2\*2

% 2、网络结点数说明：

% 输入层：28\*28

% 第一层：24\*24（卷积）\*20

% tanh

% 第二层：12\*12（pooling）\*20

% 第三层：100(全连接)

% 第四层：10(softmax)

% 3、网络训练部分采用800个样本，检验部分采用100个样本

clear all;clc;

%% 网络初始化

layer\_c1\_num=20;

layer\_s1\_num=20;

layer\_f1\_num=100;

layer\_output\_num=10;

%权值调整步进

yita=0.01;

%bias初始化

bias\_c1=(2\*rand(1,20)-ones(1,20))/sqrt(20);

bias\_f1=(2\*rand(1,100)-ones(1,100))/sqrt(20);

%卷积核初始化

[kernel\_c1,kernel\_f1]=init\_kernel(layer\_c1\_num,layer\_f1\_num);

%pooling核初始化

pooling\_a=ones(2,2)/4;

%全连接层的权值

weight\_f1=(2\*rand(20,100)-ones(20,100))/sqrt(20);

weight\_output=(2\*rand(100,10)-ones(100,10))/sqrt(100);

disp('网络初始化完成......');

%% 开始网络训练

disp('开始网络训练......');

for iter=1:20

for n=1:20

for m=0:9

%读取样本

train\_data=imread(strcat(num2str(m),'\_',num2str(n),'.bmp'));

train\_data=double(train\_data);

% 去均值

% train\_data=wipe\_off\_average(train\_data);

%前向传递,进入卷积层1

for k=1:layer\_c1\_num

state\_c1(:,:,k)=convolution(train\_data,kernel\_c1(:,:,k));

%进入激励函数

state\_c1(:,:,k)=tanh(state\_c1(:,:,k)+bias\_c1(1,k));

%进入pooling1

state\_s1(:,:,k)=pooling(state\_c1(:,:,k),pooling\_a);

end

%进入f1层

[state\_f1\_pre,state\_f1\_temp]=convolution\_f1(state\_s1,kernel\_f1,weight\_f1);

%进入激励函数

for nn=1:layer\_f1\_num

state\_f1(1,nn)=tanh(state\_f1\_pre(:,:,nn)+bias\_f1(1,nn));

end

%进入softmax层

for nn=1:layer\_output\_num

output(1,nn)=exp(state\_f1\*weight\_output(:,nn))/sum(exp(state\_f1\*weight\_output));

end

%% 误差计算部分

Error\_cost=-output(1,m+1);

% if (Error\_cost<-0.98)

% break;

% end

%% 参数调整部分

[kernel\_c1,kernel\_f1,weight\_f1,weight\_output,bias\_c1,bias\_f1]=CNN\_upweight(yita,Error\_cost,m,train\_data,...

state\_c1,state\_s1,...

state\_f1,state\_f1\_temp,...

output,...

kernel\_c1,kernel\_f1,weight\_f1,weight\_output,bias\_c1,bias\_f1);

end

end

end

disp('网络训练完成，开始检验......');

count=0;

for n=1:20

for m=0:9

%读取样本

train\_data=imread(strcat(num2str(m),'\_',num2str(n),'.bmp'));

train\_data=double(train\_data);

% 去均值

% train\_data=wipe\_off\_average(train\_data);

%前向传递,进入卷积层1

for k=1:layer\_c1\_num

state\_c1(:,:,k)=convolution(train\_data,kernel\_c1(:,:,k));

%进入激励函数

state\_c1(:,:,k)=tanh(state\_c1(:,:,k)+bias\_c1(1,k));

%进入pooling1

state\_s1(:,:,k)=pooling(state\_c1(:,:,k),pooling\_a);

end

%进入f1层

[state\_f1\_pre,state\_f1\_temp]=convolution\_f1(state\_s1,kernel\_f1,weight\_f1);

%进入激励函数

for nn=1:layer\_f1\_num

state\_f1(1,nn)=tanh(state\_f1\_pre(:,:,nn)+bias\_f1(1,nn));

end

%进入softmax层

for nn=1:layer\_output\_num

output(1,nn)=exp(state\_f1\*weight\_output(:,nn))/sum(exp(state\_f1\*weight\_output));

end

[p,classify]=max(output);

if (classify==m+1)

count=count+1;

end

fprintf('真实数字为%d 网络标记为%d 概率值为%d \n',m,classify-1,p);

end

end

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80
* 81
* 82
* 83
* 84
* 85
* 86
* 87
* 88
* 89
* 90
* 91
* 92
* 93
* 94
* 95
* 96
* 97
* 98
* 99
* 100
* 101
* 102
* 103
* 104
* 105
* 106
* 107
* 108
* 109
* 110

init\_kernel.m

function [kernel\_c1,kernel\_f1]=init\_kernel(layer\_c1\_num,layer\_f1\_num)

%% 卷积核初始化

for n=1:layer\_c1\_num

kernel\_c1(:,:,n)=(2\*rand(5,5)-ones(5,5))/12;

end

for n=1:layer\_f1\_num

kernel\_f1(:,:,n)=(2\*rand(12,12)-ones(12,12));

end

end

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9

convolution.m

function [state]=convolution(data,kernel)

%实现卷积层操作

[data\_row,data\_col]=size(data);

[kernel\_row,kernel\_col]=size(kernel);

for m=1:data\_col-kernel\_col+1

for n=1:data\_row-kernel\_row+1

state(m,n)=sum(sum(data(m:m+kernel\_row-1,n:n+kernel\_col-1).\*kernel));

end

end

end

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

pooling.m

function state=pooling(data,pooling\_a)

%% 实现取样层pooling操作

[data\_row,data\_col]=size(data);

[pooling\_row,pooling\_col]=size(pooling\_a);

for m=1:data\_col/pooling\_col

for n=1:data\_row/pooling\_row

state(m,n)=sum(sum(data(2\*m-1:2\*m,2\*n-1:2\*n).\*pooling\_a));

end

end

end

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

convolution\_f1.m

function [state\_f1,state\_f1\_temp]=convolution\_f1(state\_s1,kernel\_f1,weight\_f1)

%% 完成卷积层2操作

layer\_f1\_num=size(weight\_f1,2);

layer\_s1\_num=size(weight\_f1,1);

%%

for n=1:layer\_f1\_num

count=0;

for m=1:layer\_s1\_num

temp=state\_s1(:,:,m)\*weight\_f1(m,n);

count=count+temp;

end

state\_f1\_temp(:,:,n)=count;

state\_f1(:,:,n)=convolution(state\_f1\_temp(:,:,n),kernel\_f1(:,:,n));

end

end

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

CNN\_upweight.m

function [kernel\_c1,kernel\_f1,weight\_f1,weight\_output,bias\_c1,bias\_f1]=CNN\_upweight(yita,Error\_cost,classify,train\_data,state\_c1,state\_s1,state\_f1,state\_f1\_temp,...

output,kernel\_c1,kernel\_f1,weight\_f1,weight\_output,bias\_c1,bias\_f1)

%%% 完成参数更新，权值和卷积核

%% 结点数目

layer\_c1\_num=size(state\_c1,3);

layer\_s1\_num=size(state\_s1,3);

layer\_f1\_num=size(state\_f1,2);

layer\_output\_num=size(output,2);

[c1\_row,c1\_col,~]=size(state\_c1);

[s1\_row,s1\_col,~]=size(state\_s1);

[kernel\_c1\_row,kernel\_c1\_col]=size(kernel\_c1(:,:,1));

[kernel\_f1\_row,kernel\_f1\_col]=size(kernel\_f1(:,:,1));

%% 保存网络权值

kernel\_c1\_temp=kernel\_c1;

kernel\_f1\_temp=kernel\_f1;

weight\_f1\_temp=weight\_f1;

weight\_output\_temp=weight\_output;

%% Error计算

label=zeros(1,layer\_output\_num);

label(1,classify+1)=1;

delta\_layer\_output=output-label;

%% 更新weight\_output

for n=1:layer\_output\_num

delta\_weight\_output\_temp(:,n)=delta\_layer\_output(1,n)\*state\_f1';

end

weight\_output\_temp=weight\_output\_temp-yita\*delta\_weight\_output\_temp;

%% 更新bias\_f1以及kernel\_f1

for n=1:layer\_f1\_num

count=0;

for m=1:layer\_output\_num

count=count+delta\_layer\_output(1,m)\*weight\_output(n,m);

end

%bias\_f1

delta\_layer\_f1(1,n)=count\*(1-tanh(state\_f1(1,n)).^2);

delta\_bias\_f1(1,n)=delta\_layer\_f1(1,n);

%kernel\_f1

delta\_kernel\_f1\_temp(:,:,n)=delta\_layer\_f1(1,n)\*state\_f1\_temp(:,:,n);

end

bias\_f1=bias\_f1-yita\*delta\_bias\_f1;

kernel\_f1\_temp=kernel\_f1\_temp-yita\*delta\_kernel\_f1\_temp;

%% 更新weight\_f1

for n=1:layer\_f1\_num

delta\_layer\_f1\_temp(:,:,n)=delta\_layer\_f1(1,n)\*kernel\_f1(:,:,n);

end

for n=1:layer\_s1\_num

for m=1:layer\_f1\_num

delta\_weight\_f1\_temp(n,m)=sum(sum(delta\_layer\_f1\_temp(:,:,m).\*state\_s1(:,:,n)));

end

end

weight\_f1\_temp=weight\_f1\_temp-yita\*delta\_weight\_f1\_temp;

%% 更新 bias\_c1

for n=1:layer\_s1\_num

count=0;

for m=1:layer\_f1\_num

count=count+delta\_layer\_f1\_temp(:,:,m)\*weight\_f1(n,m);

end

delta\_layer\_s1(:,:,n)=count;

delta\_layer\_c1(:,:,n)=kron(delta\_layer\_s1(:,:,n),ones(2,2)/4).\*(1-tanh(state\_c1(:,:,n)).^2);

delta\_bias\_c1(1,n)=sum(sum(delta\_layer\_c1(:,:,n)));

end

bias\_c1=bias\_c1-yita\*delta\_bias\_c1;

%% 更新 kernel\_c1

for n=1:layer\_c1\_num

delta\_kernel\_c1\_temp(:,:,n)=rot90(conv2(train\_data,rot90(delta\_layer\_c1(:,:,n),2),'valid'),2);

end

kernel\_c1\_temp=kernel\_c1\_temp-yita\*delta\_kernel\_c1\_temp;

%% 网络权值更新

kernel\_c1=kernel\_c1\_temp;

kernel\_f1=kernel\_f1\_temp;

weight\_f1=weight\_f1\_temp;

weight\_output=weight\_output\_temp;

end

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 63
* 64
* 65
* 66
* 67
* 68
* 69
* 70
* 71
* 72
* 73
* 74
* 75
* 76
* 77
* 78
* 79
* 80

程序运行结果：   
