目录

[1] Deep learning 简介

一、什么是 Deep Learning？

实际生活中，人们为了解决一个问题，如对象的分类（对象可是是文档、图像等），首先必须做的事情是如何来表达一个对象，即必须抽取一些特征来表示一个对象，如文本的处理中，常常用词集合来表示一个文档，或把文档表示在向量空间中（称为 VSM 模型），然后才能提出不同的分类算法来进行分类；又如在图像处理中，我们可以用像素集合来表示一个图像，后来人们提出了新的特征表示，如 SIFT，这种特征在很多图像处理的应用中表现非常良好，特征选取得好坏对最终结果的影响非常巨大。因此，选取什么特征对于解决一个实际问题非常的重要。

然而，手工地选取特征是一件非常费力、启发式的方法，能不能选取好很大程度上靠经验和运气；既然手工选取特征不太好，那么能不能自动地学习一些特征呢？答案是能！Deep Learning 就是用来干这个事情的，看它的一个别名 Unsupervised Feature Learning，就可以顾名思义了，Unsupervised 的意思就是不要人参与特征的选取过程。因此，自动地学习特征的方法，统称为 Deep Learning。

二、Deep Learning 的基本思想

假设我们有一个系统 S，它有 n 层（S1,...Sn），它的输入是 I，输出是 O，形象地表示为： I =>S1=>S2=>.....=>Sn => O，如果输出 O 等于输入 I，即输入 I 经过这个系统变化之后没有任何的信息损失（呵呵，大牛说，这是不可能的。信息论中有个“信息逐层丢失”的说法（信息处理不等式），设处理 a 信息得到 b，再对 b 处

理得到 c，那么可以证明：a 和 c 的**互信息**不会超过 a 和 b 的互信息。这表明信息处理不会增加信息，大部分处理会丢失信息。当然了，如果丢掉的是没用的信息那多好啊），保持了不变，这意味着输入 I 经过每一层 Si 都没有任何的信息损失，即在任何一层 Si，它都是原有信息（即输入 I）的另外一种表示。现在回到我们的主题 Deep Learning，我们需要自动地学习特征，假设我们有一堆输入 I（如一堆图像或者文本），假设我们设计了一个系统 S（有 n 层），我们通过调整系统中参数，使得它的输出仍然是输入 I，那么我们就可以自动地获取得到输入 I 的一系列层次特征，即 S1，…, Sn。

对于深度学习来说，其思想就是堆叠多个层，也就是说这一层的输出作为下一层的输入。通过这种方式，就可以实现对输入信息进行分级表达了。

另外，前面是假设输出严格地等于输入，这个限制太严格，我们可以略微地放松这个限制，例如我们只要使得输入与输出的差别尽可能地小即可，这个放松会导致另外一类不同的方法。上述就是 Deep Learning 的基本思想。

三、浅层学习（Shallow Learning）和深度学习（Deep Learning）

浅层学习是机器学习的第一次浪潮。

20 世纪 80 年代末期，用于人工神经网络的反向传播算法（也叫 Back Propagation 算法或者 BP 算法）的发明，给机器学习带来了希望，掀起了基于统计模型的机器学习热潮。这个热潮一直持续到今天。人们发现，利用 BP 算法可以让一个人工神经网络模型从大量训练样本中学习统计规律，从而对未知事件做预测。这种基于统计的机器学习方法比起过去基于人工规则的系统，在很多方面显出优越性。这个时候的人工神经网络，虽也被称作多层感知机（Multi-layer Perceptron），但实际是种只含有一层隐层节点的**浅层模型**。

20 世纪 90 年代，各种各样的浅层机器学习模型相继被提出，例如支撑向量机（SVM，Support Vector Machines）、 Boosting、最大熵方法（如 LR，Logistic Regression）等。这些模型的结构基本上可以看成带有一层隐层节点（如 SVM、Boosting），或没有隐层节点（如 LR）。这些模型无论是在理论分析还是应用都获得了巨大的成功。相对目前的深度学习，由于理论分析难度大，训练方法又需要很多经验和技巧，浅层人工神经网络后来相对沉寂。

深度学习是机器学习的第二次浪潮。

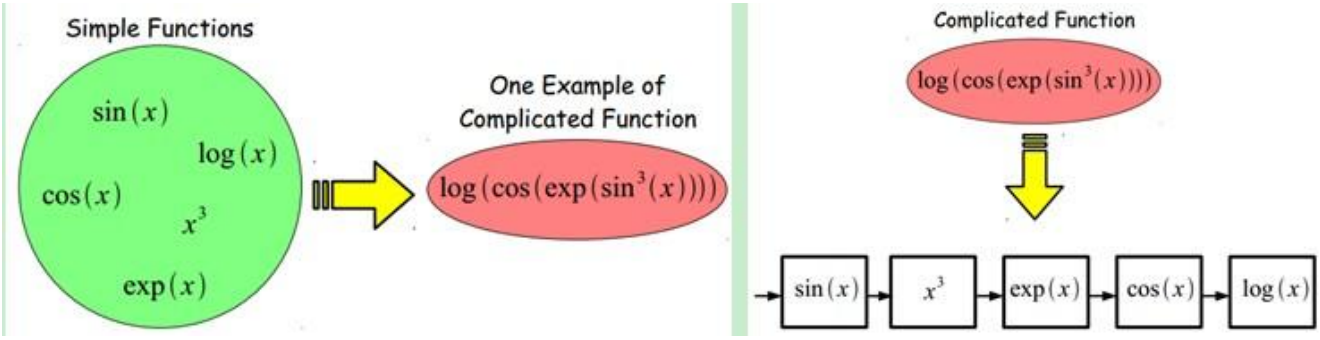
2006 年，加拿大多伦多大学教授、机器学习领域的泰斗 Geoffrey Hinton 和他的学生 RuslanSalakhutdinov 在《科学》上发表了一篇文章，开启了深度学习在学术界和工业界的浪潮。这篇文章有两个主要观点：

1）多隐层的人工神经网络具有优异的特征学习能力，学习得到的特征对数据有更本质的刻画，从而有利于可视化或分类；

2）深度神经网络在训练上的难度，可以通过“**逐层初始化**”（layer-wise pre-training）来有效克服。这篇文章中，逐层初始化是通过无监督学习实现的。

当前多数分类、回归等学习方法为浅层结构算法，其局限性在于有限样本和计算单元情况下对复杂函数的表示能力有限，针对复杂分类问题其泛化能力受到一定制约。深度学习可通过学习一种深层非线性网络结构，实现复杂函数逼近，表征输入数据的分布式表示，并展现了强大的从少数样本集中学习数据集本质特征的能力。

多层的好处是可以用较少的参数表示复杂的函数



深度学习的实质，是通过构建具有很多隐层的机器学习模型和海量的训练数据，来学习更有用的特征，从而提升分类或预测的准确性。因此，“深度模型”是手段，“特征学习”是目的。区别于传统的浅层学习，深度学习的不同在于：

1）强调了模型结构的深度，通常有 5 层、6 层，甚至 10 多层的隐层节点；

2）明确了特征学习的重要性。也就是说，通过逐层特征变换，将样本在原空间的特征表示变换到一个新特征空间，从而使分类或预测更容易。与人工规则构造特征的方法相比，利用大数据来学习特征，更能够刻画数据丰富的内在信息。

四、Deep learning 与 Neural Network

深度学习是机器学习研究中的一个新的领域，其动机在于建立模拟人脑进行分析和学习的神经网络，它模仿人脑的机制来解释数据，例如图像，声音和文本。**深度学习是无监督学习的一种。**

深度学习的概念源于人工神经网络的研究。含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层，来表示属性类别或特征，以发现数据的分布式特征表示。

Deep learning 本身算是 machine learning 的一个分支，简单可以理解为 neural network 的发展。大约二三十年前，neural network 曾经是 ML 领域特别火热的一个方向，但是后来确慢慢淡出了，原因包括以下几个方面：

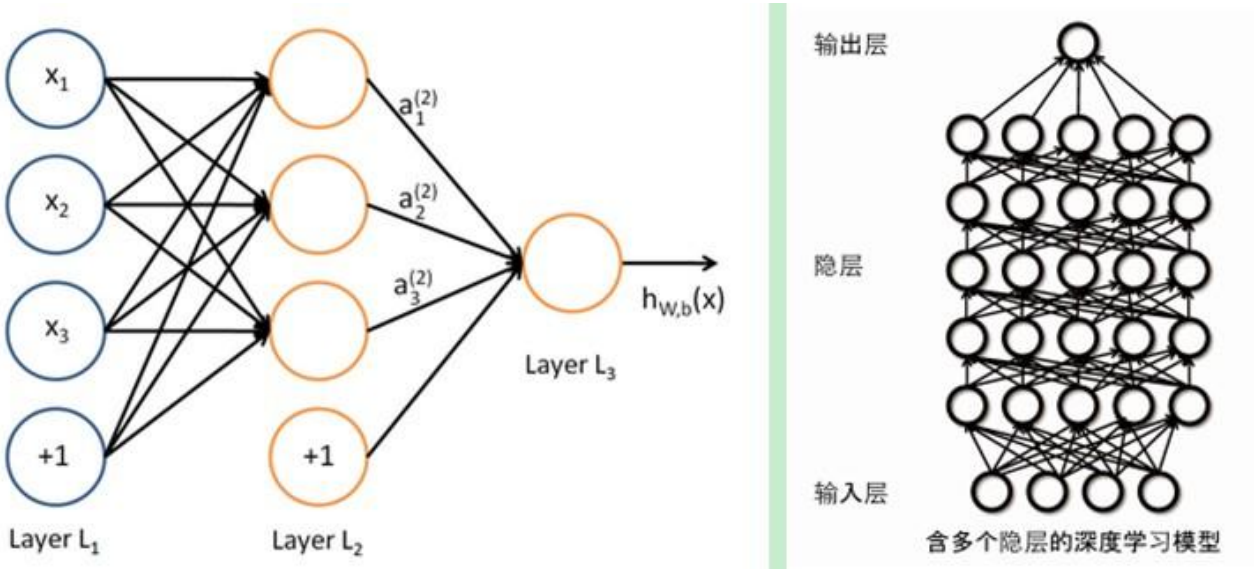
1）比较容易过拟合，参数比较难调整，而且需要不少技巧；

2）训练速度比较慢，在层次比较少（小于等于 3）的情况下效果并不比其它方法更优。

所以中间有大约 20 多年的时间，神经网络被关注很少，这段时间基本上是 SVM 和 boosting 算法的天下。但是，一个痴心的老先生 **Hinton**，他坚持了下来，并最终（和其它人一起 Bengio、Yann.lecun 等）提出了一个实际可行的 deep learning 框架。

Deep learning 与传统的神经网络之间有相同的地方也有很多不同。

二者的相同在于 deep learning 采用了与神经网络相似的分层结构，系统由包括输入层、隐层（多层）、输出层组成的多层网络，只有相邻层节点之间有连接，同一层以及跨层节点之间相互无连接，每一层可以看作是一个 logistic regression 模型；这种分层结构，是比较接近人类大脑的结构的。



而为了克服神经网络训练中的问题，DL 采用了与神经网络**不同**的训练机制。传统神经网络中，采用的是 back propagation 的方式进行，简单来讲就是采用迭代的算法来训练整个网络，随机设定初值，计算当前网络的输出，然后根据当前输出和 label 之间的差去改变前面各层的参数，直到收敛（整体是梯度下降方法）。而 deep learning 整体上是一个 **layer-wise** 的训练机制。这样做的原因是因为，如果采用 back propagation 的机制，对于一个 deep network（7 层以上），残差传播到最前面的层已经变得太小，出现所谓的 gradient diffusion（梯度扩散）。

参考：http://blog.csdn.net/zouxy09/article/details/8775518

http://blog.csdn.net/zouxy09/article/details/8775488

[2] Deep Learning 训练过程

1、传统神经网络的训练方法为什么不能用在深度神经网络

BP 算法作为传统的训练多层网络的典型算法，实际上对仅含几层的网络，该训练方法就已经很不理想。深度结构（涉及多个非线性处理单元层）**非凸目标代价函数**中普遍存在的**局部最小**是训练困难的主要来源。

BP 算法存在的问题：

（1）梯度越来越稀疏：从顶层越往下，误差校正信号越来越小；

（2）收敛到局部最小值：尤其是从远离最优区域开始的时候（随机值初始化会导致这种情况的发生）；

（3）一般，我们只能用有标签的数据来训练：但大部分的数据是没标签的，而大脑可以从没有标签的数据中学习。

2、deep learning 训练过程

如果对所有层同时训练，时间复杂度会太高；如果每次训练一层，偏差就会逐层传递。这会面临监督学习相反的问题，会严重欠拟合（因为深度网络的神经元和参数太多了）。

2006 年，Hinton 提出了在非监督数据上建立多层神经网络的一个有效方法。简单的说，分为两步，一是每次训练一层网络；二是调优，使原始表示 x 向上生成的高级表示 r 和该高级表示 r 向下生成的 x'尽可能一致。方法是：

1）首先逐层构建单层神经元，这样每次都是训练一个单层网络；

2）当所有层训练完后，Hinton 使用 **wake-sleep** 算法进行调优。

将除最顶层的其它层间的权重变为**双向的**，这样最顶层仍然是一个单层神经网络，而其它层则变为了**图模型**。向上的权重用于“认知”，向下的权重用于“生成”。然后使用 Wake-Sleep 算法调整所有的权重。**让认知和生成达成一致**，也就是保证生成的最顶层表示能够尽可能正确的**复原**底层的结点。比如顶层的一个结点表示人脸，那么所有人脸的图像应该激活这个结点，并且这个结果向下生成的图像应该能够表现为一个大概的人脸图像。Wake-Sleep 算法分为醒（wake）和睡（sleep）两个部分。

1）wake 阶段：认知过程，通过外界的特征和向上的权重（认知权重）产生每一层的抽象表示（结点状态），并且使用梯度下降修改层间的下行权重（生成权重）。也就是“如果现实跟我想象的不一样，改变我的权重使得我想象的东西就是这样的”。

2）sleep 阶段：生成过程，通过顶层表示（醒时学得的概念）和向下权重，生成底层的状态，同时修改层间向上的权重。也就是“如果梦中的景象不是我脑中的相应概念，改变我的认知权重使得这种景象在我看来就是这个概念”。

3、deep learning 训练过程具体如下：

1）使用自下上升非监督学习（就是从底层开始，一层一层的往顶层训练）：

采用无标定数据（有标定数据也可）分层训练各层参数，这一步可以看作是一个无监督训练过程，是和传统神经网络区别最大的部分（这个过程可以看作是 feature learning 过程）：

具体的，先用无标定数据训练第一层，训练时先学习第一层的参数（这一层可以看作是得到一个使得输出和输入差别最小的三层神经网络的隐层），由于模型 capacity 的限制以及稀疏性约束，使得得到的模型能够学习到数据本身的结构，从而

得到比输入更具有表示能力的特征；在学习得到第 n-1 层后，将 n-1 层的输出作为第 n 层的输入，训练第 n 层，由此分别得到各层的参数；

2）自顶向下的监督学习（就是通过带标签的数据去训练，误差自顶向下传输，对网络进行微调）：

基于第一步得到的各层参数进一步 fine-tune 整个多层模型的参数，这一步是一个有监督训练过程；第一步类似神经网络的随机初始化初值过程，由于 DL 的第一步不是随机初始化，而是通过学习输入数据的结构得到的，因而这个初值更接近全局最优，从而能够取得更好的效果；所以 deep learning 效果好很大程度上归功于第一步的 feature learning 过程。

参考： 反向传播 BP 算法

http://blog.csdn.net/zouxy09/article/details/8775518

[3] CNN 卷积神经网络推导和实现

本文来自论文：

Notes on Convolutional Neural Networks, Jake Bouvrie。

本篇主要是 CNN 的推导和实现的一些笔记，在看懂这个笔记之前，最好具有 CNN 的一些基础。这里列出一个资料供参考：

[1] LeNet-5, convolutional neural networks

[2]卷积神经网络

[3] Neural Network for Recognition of Handwritten Digits

[4] Deep learning：三十八(Stacked CNN 简单介绍)

[5] Gradient-based learning applied to document recognition.

[6]Imagenet classification with deep convolutional neural networks.

[7] UFLDL 中的“卷积特征提取”和“池化”。

[8] 反向传播 BP 算法

《Notes on Convolutional Neural Networks》

一、介绍

这个文档讨论的是 CNNs 的推导和实现。CNNs 的连接比权值要多很多，这种架构实际上隐含某种形式的规则化。这种特别的网络假定了我们希望通过数据驱动的方式学习到一些**滤波器**，作为提取输入的特征的一种方法。

本文中，我们先对训练全连接网络的经典 BP 算法做一个描述，然后推导 2D CNN 网络的卷积层和子采样层的 BP 权值更新方法。在推导过程中，我们更强调实现的效率，所以会给出一些 Matlab 代码。最后，我们讨论如何学习自动组合前一层的特征 maps，特别地，还学习特征 maps 的稀疏组合。

二、全连接的反向传播算法

典型的 CNN 中，开始几层都是卷积和**下采样**的交替，然后在最后一些层（靠近输出层的），都是全连接的一维网络。这时候我们已经将所有两维 2D 的特征 maps 转化为全连接的一维网络的输入。这样，当你准备好将最终的 2D 特征 maps 输入到 1D 网络中时，一个非常方便的方法就是把所有输出的特征 maps 连接成一个长的输入向量。然后我们回到 BP 算法的讨论。（更详细的基础推导可以参考 UFLDL 中“反向传导算法”）。

2.1、Feedforward Pass 前向传播

在下面的推导中，我们采用平方误差代价函数。我们讨论的是多类问题，共 c 类，共 N 个训练样本。

$$E^N = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(t_k^n - y_k^n)^2.$$

这里 t_k^n 表示第 n 个样本对应的标签的第 k 维。y_k^n 表示第 n 个样本对应的网络输出的第 k 个输出。对于多类问题，输出一般组织为“one-of-c”的形式，也就是只有该输入对应的类的输出节点输出为正，其他类的位或者节点为 0 或者负数，这个取决于你输出层的激活函数。sigmoid 就是 0，tanh 就是-1.

因为在全部训练集上的误差只是每个训练样本的误差的总和，所以这里我们先考虑对于一个样本的 BP。对于第 n 个样本的误差，表示为：

$$E^n = \frac{1}{2}\sum_{k=1}^{c}(t_k^n - y_k^n)^2 = \frac{1}{2}\|\mathbf{t}^n - \mathbf{y}^n\|_2^2.$$

传统的全连接神经网络中，我们需要根据 BP 规则计算代价函数 E 关于网络每一个权值的偏导数。我们用 l 来表示当前层，那么当前层的输出可以表示为：

$$\mathbf{x}^\ell = f(\mathbf{u}^\ell), \text{ with } \mathbf{u}^\ell = \mathbf{W}^\ell \mathbf{x}^{\ell-1} + \mathbf{b}^\ell$$

输出激活函数 f(.)可以有很多种，一般是 sigmoid 函数或者双曲线正切函数。sigmoid 将输出压缩到[0, 1]，所以最后的输出平均值一般趋于 0 。所以如果将我们的训练数据归一化为零均值和方差为 1，可以在梯度下降的过程中增加收敛性。对于归一化的数据集来说，双曲线正切函数也是不错的选择。

2.2、Backpropagation Pass 反向传播

反向传播回来的误差可以看做是每个神经元的基的灵敏度 sensitivities（灵敏度的意思就是我们的基 b 变化多少，误差会变化多少，也就是误差对基的变化率，也就是导数了），定义如下：（第二个等号是根据求导的链式法则得到的）

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial u}\frac{\partial u}{\partial b} = \delta$$

因为∂u/∂b=1，所以∂E/∂b=∂E/∂u=δ，也就是说 bias 基的灵敏度∂E/∂b=δ和误差 E 对一个节点全部输入 u 的导数∂E/∂u 是相等的。这个导数就是让高层误差反向传播到底层的神来之笔。反向传播就是用下面这条关系式：（下面这条式子表达的就是第 l 层的灵敏度，就是）

$$\boldsymbol{\delta}^\ell = (W^{\ell+1})^T \boldsymbol{\delta}^{\ell+1} \circ f'(\mathbf{u}^\ell)$$ 公式（1）

这里的“∘”表示每个元素相乘。输出层的神经元的灵敏度是不一样的：

$$\delta^L = f'(\mathbf{u}^L) \circ (\mathbf{y}^n - \mathbf{t}^n).$$

最后，对每个神经元运用 delta（即δ）规则进行权值更新。具体来说就是，对一个给定的神经元，得到它的输入，然后用这个神经元的 delta（即δ）来进行缩放。用向量的形式表述就是，对于第 l 层，误差对于该层每一个权值（组合为矩阵）的导数是该层的输入（等于上一层的输出）与该层的灵敏度（该层每个神经元的δ组合成一个向量的形式）的叉乘。然后得到的偏导数乘以一个负学习率就是该层的神经元的权值的更新了：

$$\frac{\partial E}{\partial \mathbf{W}^\ell} = \mathbf{x}^{\ell-1}(\boldsymbol{\delta}^\ell)^T$$
$$\Delta \mathbf{W}^\ell = -\eta \frac{\partial E}{\partial \mathbf{W}^\ell}$$

公式（2）

对于 bias 基的更新表达式差不多。实际上，对于每一个权值$(W)_{ij}$都有一个特定的学习率η_{ij}。

三、Convolutional Neural Networks 卷积神经网络

3.1、Convolution Layers 卷积层

我们现在关注网络中卷积层的 BP 更新。在一个卷积层，上一层的特征 maps 被一个可学习的卷积核进行卷积，然后通过一个激活函数，就可以得到输出特征 map。每一个输出 map 可能是组合卷积多个输入 maps 的值：

$$\mathbf{x}_j^\ell = f\left(\sum_{i \in M_j} \mathbf{x}_i^{\ell-1} * \mathbf{k}_{ij}^\ell + b_j^\ell\right)$$

这里 M_j 表示选择的输入 maps 的集合，那么到底选择哪些输入 maps 呢？有选择一对的或者三个的。但下面我们会讨论如何去自动选择需要组合的特征 maps。每一个输出 map 会给一个额外的偏置 b，但是对于一个特定的输出 map，卷积每个输入 maps 的卷积核是不一样的。也就是说，如果输出特征 map j 和输出特征 map k 都是从输入 map i 中卷积求和得到，那么对应的卷积核是不一样的。

3.1.1、Computing the Gradients 梯度计算

我们假定每个卷积层 l 都会接一个下采样层 l+1。对于 BP 来说，根据上文我们知道，要想求得层 l 的每个神经元对应的权值的权值更新，就需要先求层 l 的每一个神经节点的灵敏度δ（也就是权值更新的公式（2））。为了求这个灵敏度我们就需要先对下一层的节点（连接到当前层 l 的感兴趣节点的第 l+1 层的节点）的灵敏度求和（得到δ^{l+1}），然后乘以这些连接对应的权值（连接第 l 层感兴趣节点和第 l+1 层节点的权值）W。再乘以当前层 l 的该神经元节点的输入 u 的激活函数 f 的导数值（也就是那个灵敏度反向传播的公式（1）的δ^l的求解），这样就可以得到当前层 l 每个神经节点对应的灵敏度δ^l了。

然而，因为下采样的存在，采样层的一个像素（神经元节点）对应的灵敏度δ对应于卷积层（上一层）的输出 map 的一块像素（采样窗口大小）。因此，层 l 中的一个 map 的每个节点只与 l+1 层中相应 map 的一个节点连接。

为了有效计算层 l 的灵敏度，我们需要**上采样** upsample。这个下采样 downsample 层对应的灵敏度 map（特征 map 中每个像素对应一个灵敏度，所以也组成一个 map），这样才使得这个灵敏度 map 大小与卷积层的 map 大小一致，然后再将层 l 的 map 的激活值的偏导数与从第 l+1 层的上采样得到的灵敏度 map 逐元素相乘（也就是公式（1））。

在下采样层 map 的权值都取一个相同值β，而且是一个常数。所以我们只需要将上一个步骤得到的结果乘以一个β就可以完成第 l 层灵敏度δ的计算。

我们可以对卷积层中每一个特征 map j 重复相同的计算过程。但很明显需要匹配相应的子采样层的 map（参考公式（1））：

$$\delta_j^\ell = \beta_j^{\ell+1}\left(f'(\mathbf{u}_j^\ell) \circ \mathrm{up}(\delta_j^{\ell+1})\right)$$

up(.)表示一个**上采样**操作。如果下采样的采样因子是 n 的话，它简单的将每个像素水平和垂直方向上拷贝 n 次。这样就可以恢复原来的大小了。实际上，这个函数可以用 Kronecker 乘积来实现：

$$\mathrm{up}(\mathbf{x}) \equiv \mathbf{x} \otimes \mathbf{1}_{n\times n}.$$

好，到这里，对于一个给定的 map，我们就可以计算得到其灵敏度 map 了。然后我们就可以通过简单的对层 l 中的灵敏度 map 中所有节点进行求和快速的计算 bias 基的梯度了：

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^\ell)_{uv}.$$

公式（3）

最后，对卷积核的权值的梯度就可以用 BP 算法来计算了（公式（2））。另外，很多连接的权值是共享的，因此，对于一个给定的权值，我们需要对所有与该权值有联系（权值共享的连接）的连接对该点求梯度，然后对这些梯度进行求和，就像上面对 bias 基的梯度计算一样：

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^\ell} = \sum_{u,v} (\delta_j^\ell)_{uv} (\mathbf{p}_i^{\ell-1})_{uv}$$

这里，$(\mathbf{p}_i^{\ell-1})_{uv}$是$\mathbf{x}_i^{\ell-1}$中的在卷积的时候与$\mathbf{k}_{ij}^{\ell}$逐元素相乘的 patch，输出卷积 map 的(u, v)位置的值是由上一层的(u, v)位置的 patch 与卷积核 k_ij 逐元素相乘的结果。

咋一看，好像我们需要煞费苦心地记住输出 map（和对应的灵敏度 map）每个像素对应于输入 map 的哪个 patch。但实际上，在 Matlab 中，可以通过一个代码就实现。对于上面的公式，可以用 Matlab 的卷积函数来实现：

$$\frac{\partial E}{\partial \mathbf{k}_{ij}^{\ell}} = \text{rot180}(\text{conv2}(\mathbf{x}_i^{\ell-1}, \text{rot180}(\boldsymbol{\delta}_j^{\ell}), 'valid'))$$

我们先对 delta 灵敏度 map 进行旋转，这样就可以进行互相关计算，而不是卷积（在卷积的数学定义中，特征矩阵（卷积核）在传递给 conv2 时需要先翻转（flipped）一下。也就是颠倒下特征矩阵的行和列）。然后把输出反旋转回来，这样我们在前向传播进行卷积的时候，卷积核才是我们想要的方向。

3.2、Sub-sampling Layers 子采样层

对于子采样层来说，有 N 个输入 maps，就有 N 个输出 maps，只是每个输出 map 都变小了。

$$\mathbf{x}_j^{\ell} = f\left(\beta_j^{\ell}\,\text{down}(\mathbf{x}_j^{\ell-1}) + b_j^{\ell}\right)$$

down(.)表示一个下采样函数。典型的操作一般是对输入图像的不同 nxn 的块的所有像素进行求和。这样输出图像在两个维度上都缩小了 n 倍。每个输出 map 都对应一个属于自己的乘性偏置β和一个加性偏置 b。

3.2.1、Computing the Gradients 梯度计算

这里最困难的是计算灵敏度 map。一旦我们得到这个了，那我们唯一需要更新的偏置参数β和 b 就可以轻而易举了（公式（3））。如果下一个卷积层与这个子采样层是全连接的，那么就可以通过 BP 来计算子采样层的灵敏度 maps。

我们需要计算卷积核的梯度，所以我们必须找到输入 map 中哪个 patch 对应输出 map 的哪个像素。这里，就是必须找到当前层的灵敏度 map 中哪个 patch 对应与下一层的灵敏度 map 的给定像素，这样才可以利用公式（1）那样的δ递推，也就是灵敏度反向传播回来。另外，需要乘以输入 patch 与输出像素之间连接的权值，这个权值实际上就是卷积核的权值（已旋转的）。

$$\delta_j^\ell = f'(\mathbf{u}_j^\ell) \circ \mathrm{conv2}(\delta_j^{\ell+1}, \mathrm{rot180}(\mathbf{k}_j^{\ell+1}), \text{'full'})$$

在这之前，我们需要先将核旋转一下，让卷积函数可以实施互相关计算。另外，我们需要对卷积边界进行处理，但在 Matlab 里面，就比较容易处理。Matlab 中全卷积会对缺少的输入像素补 0 。

到这里，我们就可以对 b 和β计算梯度了。首先，加性基 b 的计算和上面卷积层的一样，对灵敏度 map 中所有元素加起来就可以了：

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^\ell)_{uv}.$$

而对于乘性偏置β，因为涉及到了在前向传播过程中下采样 map 的计算，所以我们最好在前向的过程中保存好这些 maps，这样在反向的计算中就不用重新计算了。我们定义：

$$\mathbf{d}_j^\ell = \mathrm{down}(\mathbf{x}_j^{\ell-1}).$$

这样，对β的梯度就可以用下面的方式计算：

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\delta_j^\ell \circ \mathbf{d}_j^\ell)_{uv}$$

3.3、Learning Combinations of Feature Maps 学习特征 map 的组合

大部分时候，通过卷积多个输入 maps，然后再对这些卷积值求和得到一个输出 map，这样的效果往往是比较好的。在一些文献中，一般是人工选择哪些输入 maps 去组合得到一个输出 map。但我们这里尝试去让 CNN 在训练的过程中学习这些组合，也就是让网络自己学习挑选哪些输入 maps 来计算得到输出 map 才是最好的。我们用 αij 表示在得到第 j 个输出 map 的其中第 i 个输入 map 的权值或者贡献。这样，第 j 个输出 map 可以表示为：

$$\mathbf{x}_j^\ell = f\left(\sum_{i=1}^{N_{in}} \alpha_{ij}(\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^\ell) + b_j^\ell\right)$$

需要满足约束：

$$\sum_i \alpha_{ij} = 1, \quad \text{and} \quad 0 \le \alpha_{ij} \le 1.$$

这些对变量αij 的约束可以通过将变量αij 表示为一个组无约束的隐含权值 c_{ij} 的 softmax 函数来加强。（因为 softmax 的因变量是自变量的指数函数，他们的变化率会不同）。

$$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_k \exp(c_{kj})}.$$

因为对于一个固定的 j 来说，每组权值 c_{ij} 都是和其他组的权值独立的，所以为了方面描述，我们把下标 j 去掉，只考虑一个 map 的更新，其他 map 的更新是一样的过程，只是 map 的索引 j 不同而已。

Softmax 函数的导数表示为：

$$\frac{\partial \alpha_k}{\partial c_i} = \delta_{ki}\alpha_i - \alpha_i\alpha_k$$

这里的δ是 Kronecker delta。对于误差对于第 l 层变量αi 的导数为：

$$\frac{\partial E}{\partial \alpha_i} = \frac{\partial E}{\partial u^\ell}\frac{\partial u^\ell}{\partial \alpha_i} = \sum_{u,v}\left(\boldsymbol{\delta}^\ell \circ (\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^\ell)\right)_{uv}.$$

最后就可以通过链式规则去求得代价函数关于权值 c_i 的偏导数了：

$$\begin{aligned}\frac{\partial E}{\partial c_i} &= \sum_k \frac{\partial E}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} \\ &= \alpha_i\left(\frac{\partial E}{\partial \alpha_i} - \sum_k \frac{\partial E}{\partial \alpha_k}\alpha_k\right).\end{aligned}$$

3.3.1、Enforcing Sparse Combinations 加强稀疏性组合

为了限制αi 是稀疏的，也就是限制一个输出 map 只与某些而不是全部的输入 maps 相连。我们在整体代价函数里增加稀疏约束项 Ω(α)。对于单个样本，重写代价函数为：

$$\tilde{E}^n = E^n + \lambda \sum_{i,j} |(\boldsymbol{\alpha})_{ij}|$$

然后寻找这个规则化约束项对权值 ci 求导的贡献。规则化项 Ω(α)对 αi 求导是：

$$\frac{\partial \Omega}{\partial \alpha_i} = \lambda \operatorname{sign}(\alpha_i)$$

然后，通过链式法则，对 ci 的求导是：

$$\frac{\partial \Omega}{\partial c_i} = \sum_k \frac{\partial \Omega}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial c_i}$$

$$= \lambda \left(|\alpha_i| - \alpha_i \sum_k |\alpha_k| \right).$$

所以，权值 ci 最后的梯度是：

$$\frac{\partial \tilde{E}^n}{\partial c_i} = \frac{\partial E^n}{\partial c_i} + \frac{\partial \Omega}{\partial c_i}.$$

3.4、Making it Fast with MATLAB

CNN 的训练主要是在卷积层和子采样层的交互上，其主要的计算瓶颈是：

1）前向传播过程：下采样每个卷积层的 maps；

2)反向传播过程 :上采样高层子采样层的灵敏度 map ,以匹配底层的卷积层输出 maps 的大小；

3）sigmoid 的运用和求导。

对于第一和第二个问题，我们考虑的是如何用 Matlab 内置的图像处理函数去实现上采样和下采样的操作。对于上采样，imresize 函数可以搞定，但需要很大的开销。一个比较快速的版本是使用 Kronecker 乘积函数 kron。通过一个全一矩阵 ones 来和我们需要上采样的矩阵进行 Kronecker 乘积，就可以实现上采样的效果。对于前向传播过程中的下采样，imresize 并没有提供在缩小图像的过程中还计算 nxn 块内像素的和的功能，所以没法用。一个比较好和快速的方法是用一个全一的卷积核来卷积图像，然后简单的通过标准的索引方法来采样最后卷积结果。例如，如果下采样的域是 2x2 的 ,那么我们可以用 2x2 的元素全是 1 的卷积核来卷积图像。然后再卷积后的图像中，

我们每个 2 个点采集一次数据，y=x(1:2:end,1:2:end)，这样就可以得到了两倍下采样，同时执行求和的效果。

对于第三个问题，实际上有些人以为 Matlab 中对 sigmoid 函数进行 inline 的定义会更快，其实不然，Matlab 与 C/C++等等语言不一样，Matlab 的 inline 反而比普通的函数定义更非时间。所以，我们可以直接在代码中使用计算 sigmoid 函数及其导数的真实代码。

转自：http://blog.csdn.net/zouxy09/article/details/9993371

[4] CNN 的反向求导及练习

前言：

CNN 作为 DL 中最成功的模型之一，有必要更进一步研究。虽然在前面的博文 Stacked CNN 简单介绍中大概介绍过 CNN 的使用，不过那是有个前提的：CNN 中的参数必须已提前学习好。而本文的主要目的是介绍 CNN 参数在使用 bp 算法时该怎么训练，毕竟 CNN 中有卷积层和下采样层，虽然和 MLP 的 bp 算法本质上相同，但形式上还是有些区别的，很显然在完成 CNN 反向传播前了解 bp 算法是必须的。本文的实验部分是参考斯坦福 UFLDL 新教程 UFLDL :Exercise: Convolutional Neural Network 里面的内容。

实验基础：

CNN 反向传播求导时的具体过程可以参考论文 Notes on Convolutional Neural Networks, Jake Bouvrie，该论文讲得很全面，比如它考虑了 pooling 层也加入了权值、偏置值及非线性激发(因为这 2 种值也需要 learn)，该论文的解读可参考 zouxy09 的博文 CNN 卷积神经网络推导和实现。除了 bp 算法外，本人认为，理解了下面 4 个子问题基本上就可以弄懂 CNN 的求导了（bp 算法这里就不多做介绍，网上资料实在是太多了），另外为了讲清楚一些细节过程，本文中举的例子都是简化了一些条件的，且 linux 下文本和公式编辑太难弄了，文中难免有些地方会出错，大家原谅下。

首先我们来看看 CNN 系统的目标函数，设有样本(xi, yi)共 m 个，CNN 网络共有 L 层，中间包含若干个卷积层和 pooling 层，最后一层的输出为 f(xi)，则系统的 loss 表达式为(对权值进行了惩罚，一般分类都采用交叉熵形式)：

$$Loss= -\frac{1}{m}*\sum_{i=1}^{m} y_i' \log f(x_i) + \lambda \sum_{k=1}^{L} sum(||W_k||^2)$$

问题一：求输出层的误差敏感项。

现在只考虑个一个输入样本(x, y)的情形，loss 函数和上面的公式类似是用交叉熵来表示的，暂时不考虑权值规则项，样本标签采用 one-hot 编码，CNN 网络的最后一层采用 softmax 全连接(多分类时输出层一般用 softmax)，样本(x,y)经过 CNN 网络后的最终的输出用 f(x)表示，则对应该样本的 loss 值为:

negative log-likelihood　　natural log (ln)

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

其中 f(x)的下标 c 的含义见公式：

$$f(\mathbf{x})_c = p(y = c|\mathbf{x})$$

因为 x 通过 CNN 后得到的输出 f(x)是一个向量，该向量的元素值都是概率值，分别代表着 x 被分到各个类中的概率，而 f(x)中下标 c 的意思就是输出向量中取对应 c 那个类的概率值。

采用上面的符号，可以求得此时 loss 值对输出层的误差敏感性表达式为：

$$\begin{aligned} & \nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \\ = \quad & -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x})) \end{aligned}$$

其中 e(y)表示的是样本 x 标签值的 one-hot 表示，其中只有一个元素为 1,其它都为 0.

其推导过程如下（先求出对输出层某个节点 c 的误差敏感性，参考 Larochelle 关于 DL 的课件:Output layer gradient ）,求出输出层中 c 节点的误差敏感值：

$$
\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left(\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(\frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \, \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left(1_{(y=c)} f(\mathbf{x})_y - f(\mathbf{x})_y \, f(\mathbf{x})_c \right) \\
= & -\left(1_{(y=c)} - f(\mathbf{x})_c \right)
\end{aligned}
$$

$$
\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}
$$

由上面公式可知，如果输出层采用 sfotmax，且 loss 用交叉熵形式，则最后一层的误差敏感值就等于 CNN 网络输出值 f(x)减样本标签值 e(y),即 f(x)-e(y),其形式非常简单，这个公式是不是很眼熟？很多情况下如果 model 采用 MSE 的 loss，即 loss=1/2*(e(y)-f(x))^2，那么 loss 对最终的输出 f(x)求导时其结果就是 f(x)-e(y),虽然和上面的结果一样，但是大家不要搞混淆了，这 2 个含义是不同的，一个是对输出层节点输入值的导数(softmax 激发函数)，一个是对输出层节点输出值的导数(任意激发函数)。而在使用 MSE 的 loss 表达式时，输出层的误差敏感项为(f(x)-e(y)).*f(x)’，两者只相差一个因子。

这样就可以求出第 L 层的权值 W 的偏导数：

$$
\frac{\partial Loss}{\partial W_L} = -\frac{1}{m} * (e(y) - f(x)) \bullet f(x)' + \lambda W_L
$$

输出层偏置的偏导数：

$$\frac{\partial Loss}{\partial b_L} = -\frac{1}{m} * (e(y) - f(x))$$

上面 2 个公式的 e(y)和 f(x)是一个矩阵，已经把所有 m 个训练样本考虑进去了,每一列代表一个样本。

问题二：当接在卷积层的下一层为 pooling 层时，求卷积层的误差敏感项。

假设第 l(小写的 l，不要看成数字’ 1’ 了)层为卷积层，第 l+1 层为 pooling 层，且 pooling 层的误差敏感项为： δ^{l+1}_j ,卷积层的误差敏感项为： δ^l_j ，则两者的关系表达式为：

$$\delta^l_j = upsample(\delta^{l+1}_j) \bullet h(a^l_j)'$$

这里符号●表示的是矩阵的点积操作，即对应元素的乘积。卷积层和 unsample()后的 pooling 层节点是一一对应的，所以下标都是用 j 表示。后面的符号 $h(a^l_j)'$ 表示的是第 l 层第 j 个节点处激发函数的导数(对节点输入的导数)。

其中的函数 unsample()为上采样过程，其具体的操作得看是采用的什么 pooling 方法了。但 unsample 的大概思想为：pooling 层的每个节点是由卷积层中多个节点(一般为一个矩形区域)共同计算得到，所以 pooling 层每个节点的误差敏感值也是由卷积层中多个节点的误差敏感值共同产生的，只需满足两层见各自的误差敏感值相等，下面以 mean-pooling 和 max-pooling 为例来说明。

假设卷积层的矩形大小为 4×4, pooling 区域大小为 2×2，很容易知道 pooling 后得到的矩形大小也为 2*2(本文默认 pooling 过程是没有重叠的，卷积过程是每次移动一个像素，即是有重叠的，后续不再声明),如果此时 pooling 后的矩形误差敏感值如下：

1	3
2	4

则按照 mean-pooling，首先得到的卷积层应该是 4×4 大小，其值分布为(等值复制)：

1	1	3	3
1	1	3	3
2	2	4	4
2	2	4	4

因为得满足反向传播时各层见误差敏感总和不变，所以卷积层对应每个值需要平摊（除以 pooling 区域大小即可，这里 pooling 层大小为 2×2=4)），最后的卷积层值分布为：

0.25	0.25	0.75	0.75
0.25	0.25	0.75	0.75
0.5	0.5	1	1
0.5	0.5	1	1

mean-pooling 时的 unsample 操作可以使用 matalb 中的函数 kron()来实现，因为是采用的矩阵 Kronecker 乘积。C=kron(A, B)表示的是矩阵 B 分别与矩阵 A 中每个元素相乘，然后将相乘的结果放在 C 中对应的位置。比如：

```
>> A=magic(2)

A =

     1     3
     4     2

>> B=magic(3)

B =

     8     1     6
     3     5     7
     4     9     2

>> C=kron(A,B)

C =

     8     1     6    24     3    18
     3     5     7     9    15    21
     4     9     2    12    27     6
    32     4    24    16     2    12
    12    20    28     6    10    14
    16    36     8     8    18     4
```

如果是 max-pooling，则需要记录前向传播过程中 pooling 区域中最大值的位置，这里假设 pooling 层值 1,3,2,4 对应的 pooling 区域位置分别为右下、右上、左上、左下。则此时对应卷积层误差敏感值分布为：

0	0	0	3
0	1	0	0
2	0	0	0
0	0	4	0

当然了，上面 2 种结果还需要点乘卷积层激发函数对应位置的导数值了，这里省略掉。

问题三：当接在 pooling 层的下一层为卷积层时，求该 pooling 层的误差敏感项。

假设第 l 层(pooling 层)有 N 个通道，即有 N 张特征图，第 l+1 层(卷积层)有 M 个特征，l 层中每个通道图都对应有自己的误差敏感值，其计算依据为第 l+1 层所有特征核的贡献之和。下面是第 l+1 层中第 j 个核对第 l 层第 i 个通道的误差敏感值计算方法：

$$\delta_i^l = \sum_{j=1}^{M} \delta^{l+1}{}_j \bigstar Kij$$

符号★表示的是矩阵的卷积操作，这是真正意义上的离散卷积，不同于卷积层前向传播时的相关操作，因为严格意义上来讲，卷积神经网络中的卷积操作本质是一个相关操作，并不是卷积操作，只不过它可以用卷积的方法去实现才这样叫。而求第 i 个通道的误差敏感项时需要将 l+1 层的所有核都计算一遍，然后求和。另外因为这里默认 pooling 层是线性激发函数，所以后面没有乘相应节点的导数。

举个简单的例子，假设拿出第 l 层某个通道图，大小为 3×3，第 l+1 层有 2 个特征核，核大小为 2×2，则在前向传播卷积时第 l+1 层会有 2 个大小为 2×2 的卷积图。如果 2 个特征核分别为：

0.1	0.2	-0.3	0.1
0.2	0.4	0.1	0.2

反向传播求误差敏感项时，假设已经知道第 l+1 层 2 个卷积图的误差敏感值：

1	3	2	1
2	2	1	1

离散卷积函数 conv2()的实现相关子操作时需先将核旋转 180 度(即左右翻转后上下翻转)，但这里实现的是严格意义上的卷积，所以在用 conv2()时，对应的参数核不需要翻转（在有些 toolbox 里面，求这个问题时用了旋转，那是因为它们已经把所有的卷积核都旋转过，这样在前向传播时的相关操作就不用旋转了。并不矛盾）。且这时候该函数需要采用’full’模式，所以最终得到的矩阵大小为 3×3,（其中 3=2+2-1），刚好符第 l 层通道图的大小。采用’ full’模式需先将第 l+1 层 2 个卷积图扩充，周围填 0,padding 后如下：

0	0	0	0	0	0	0	0
0	1	3	0	0	2	1	0
0	2	2	0	0	1	1	0
0	0	0	0	0	0	0	0

扩充后的矩阵和对应的核进行卷积的结果如下情况：

0	0	0	0				0.1	0.5	0.6
0	1	3	0	★	0.1	0.2	=	0.4	1.6	1.6
0	2	2	0		0.2	0.4		0.4	1.2	0.8
0	0	0	0							

0	0	0	0
0	2	1	0
0	1	1	0
0	0	0	0

★

-0.3	0.1
0.1	0.2

=

-0.6	-0.1	0.1
-0.1	0.3	0.3
0.1	0.3	0.2

可以通过手动去验证上面的结果，因为是离散卷积操作，而离散卷积等价于将核旋转后再进行相关操作。而第 l 层那个通道的误差敏感项为上面 2 者的和，呼应问题三，最终答案为：

0.1	0.5	0.6
0.4	1.6	1.6
0.4	1.2	0.8

+

-0.6	-0.1	0.1
-0.1	0.3	0.3
0.1	0.3	0.2

=

-0.5	0.4	0.7
0.3	1.9	1.9
0.5	1.5	1.0

那么这样问题 3 这样解的依据是什么呢？其实很简单，本质上还是 bp 算法，即第 l 层的误差敏感值等于第 l+1 层的误差敏感值乘以两者之间的权值，只不过这里由于是用了卷积，且是有重叠的，l 层中某个点会对 l+1 层中的多个点有影响。比如说最终的结果矩阵中最中间那个 0.3 是怎么来的呢？在用 2×2 的核对 3×3 的输入矩阵进行卷积时，一共进行了 4 次移动，而 3×3 矩阵最中间那个值在 4 次移动中均对输出结果有影响，且 4 次的影响分别在右下角、左下角、右上角、左上角。所以它的值为 2×0.2+1×0.1+1×0.1-1×0.3=0.3，建议大家用笔去算一下，那样就可以明白为什么这里可以采用带’ full’ 类型的 conv2()实现。

问题四：求与卷积层相连那层的权值、偏置值导数。

前面 3 个问题分别求得了输出层的误差敏感值、从 pooling 层推断出卷积层的误差敏感值、从卷积层推断出 pooling 层的误差敏感值。下面需要利用这些误差敏感值模型中参数的导数。这里没有考虑 pooling 层的非线性激发，因此 pooling 层前面是没有权值的，也就没有所谓的权值的导数了。现在将主要精力放在卷积层前面权值的求导上(也就是问题四)。

假设现在需要求第 l 层的第 i 个通道，与第 l+1 层的第 j 个通道之间的权值和偏置的导数，则计算公式如下：

$$\frac{\partial Loss}{\partial Kij} = X_i^l \odot \delta_j^{l+1}$$

$$\frac{\partial Loss}{\partial bj} = \sum_{u,v} \left(\delta_j^{l+1}\right)_{u,v}$$

其中符号⊙表示矩阵的相关操作，可以采用 conv2()函数实现。在使用该函数时，需将第 l+1 层第 j 个误差敏感值翻转。

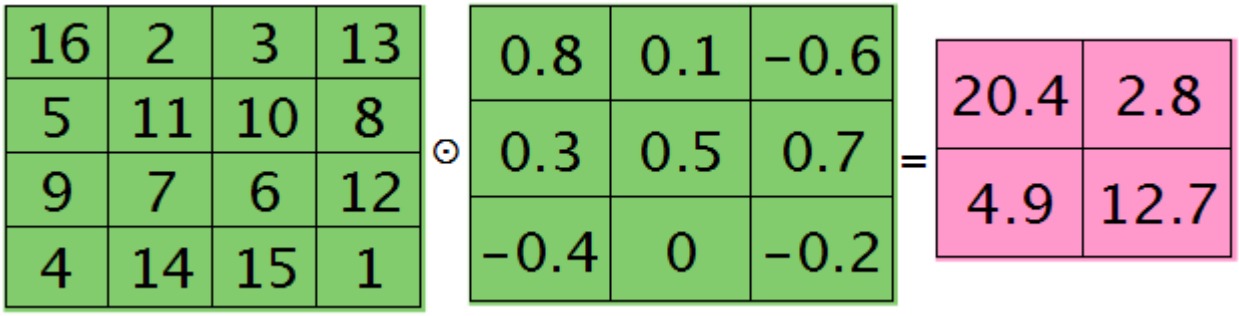
例如，如果第 l 层某个通道矩阵 i 大小为 4×4,如下：

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

第 l+1 层第 j 个特征的误差敏感值矩阵大小为 3×3,如下：

0.8	0.1	-0.6
0.3	0.5	0.7
-0.4	0	-0.2

很明显，这时候的特征 Kij 导数的大小为 2×2 的，且其结果为：



而此时偏置值 bj 的导数为 1.2 ，将 j 区域的误差敏感值相加即可(0.8+0.1-0.6+0.3+0.5+0.7-0.4-0.2=1.2)，因为 b 对 j 中的每个节点都有贡献，按照多项式的求导规则(和的导数等于导数的和)很容易得到。

为什么采用矩阵的相关操作就可以实现这个功能呢？由 bp 算法可知 ,l 层 i 和 l+1 层 j 之间的权值等于 l+1 层 j 处误差敏感值乘以 l 层 i 处的输入，而 j 中某个节点因为是由 i+1 中一个区域与权值卷积后所得，所以 j 处该节点的误差敏感值对权值中所有元素都有贡献，由此可见，将 j 中每个元素对权值的贡献(尺寸和核大小相同)相加，就得到了权值的偏导数了(这个例子的结果是由 9 个 2×2 大小的矩阵之和)，同样，如果大家动笔去推算一下 ,就会明白这时候为什么可以用带’ valid’ 的 conv2()完成此功能。

实验总结：

1. 卷积层过后，可以先跟 pooling 层，再通过非线性传播函数。也可以是先通过非线性传播函数再经过 pooling 层。
2. CNN 的结构本身就是一种规则项。

3. 实际上每个权值的学习率不同时优化会更好。
4. 发现 Ng 以前的 ufldl 中教程里面 softmax 并没有包含偏置值参数，至少他给的 start code 里面没有包含，严格来说是错误的。
5. 当输入样本为多个时，bp 算法中的误差敏感性也是一个矩阵。每一个样本都对应有自己每层的误差敏感性。
6. 血的教训啊，以后循环变量名不能与终止名太相似，一不小心引用下标是就弄错 比如 for filterNum = 1:numFilters 时一不小心把下标用 numFilters 去代替了，花了大半天去查这个 debug.
7. matlab 中 conv2()函数在卷积过程中默认是每次移动一个像素，即重叠度最高的卷积。

实验结果：

按照网页教程 UFLDL：Exercise: Convolutional Neural Network 和 UFLDL：Optimization: Stochastic Gradient Descent ,对 MNIST 数据库进行识别 ,完成练习中 YOU CODE HERE 部分后，该 CNN 网络的识别率为：**95.76%**

只采用了一个卷积层+一个 pooling 层+一个 softmax 层。卷积层的特征个数为 20, 卷积核大小为 9×9, pooling 区域大小为 2×2.

在进行实验前，需下载好本实验的标准代码：https://github.com/amaas/stanford_dl_ex。

然后在 common 文件夹放入下载好的 MNIST 数据库，

见 http://yann.lecun.com/exdb/mnist/.

注意 MNIST 文件名需要和代码中的保持一致。

实验代码： 省略

转自：http://www.cnblogs.com/tornadomeet/p/3468450.html

[5] CNN 卷积神经网络（一）

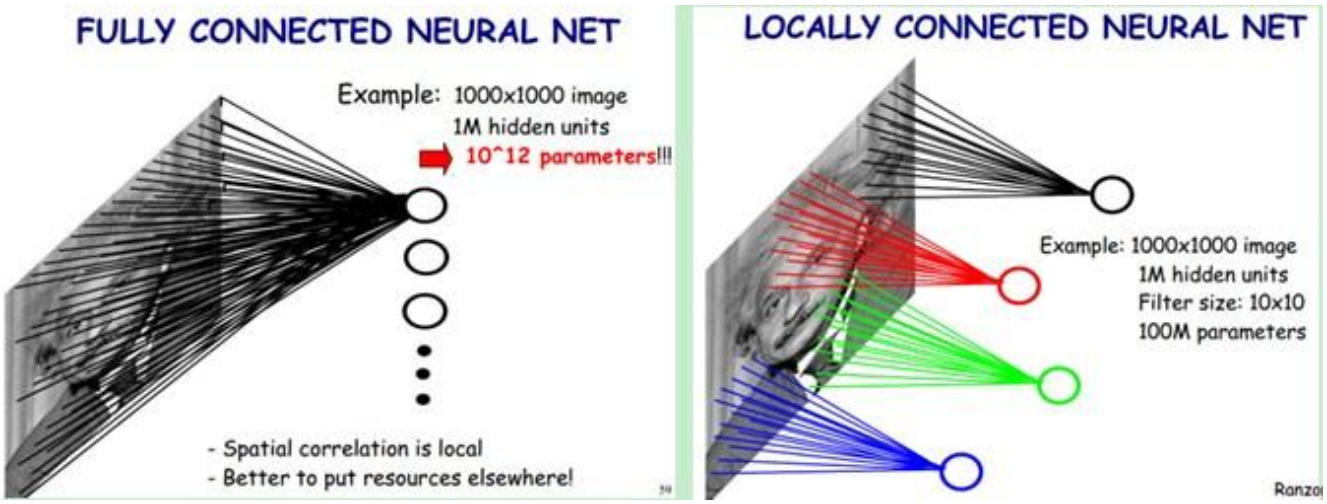
深度解析 CNN

本文整理了网上几位大牛的博客，详细地讲解了 CNN 的基础结构与核心思想，欢迎交流。

1. 概述

卷积神经网络是一种特殊的深层的神经网络模型，它的特殊性体现在两个方面，一方面它的神经元间的连接是非全连接的，另一方面同一层中某些神经元之间的连接的权重是共享的（即相同的）。它的非全连接和权值共享的网络结构使之更类似于生物 神经网络，降低了网络模型的复杂度（对于很难学习的深层结构来说，这是非常重要的），减少了权值的数量。

回想一下 BP 神经网络。BP 网络每一层节点是一个线性的一维排列状态，层与层的网络节点之间是全连接的。这样设想一下，如果 BP 网络中层与层之间的节点连接不再是全连接，而是局部连接的。这样，就是一种最简单的一维卷积网络。如果我们把上述这个思路扩展到二维，这就是我们在大多数参考资料上看到的卷积神经网络。具体参看下图：



上图左：全连接网络。如果我们有 1000x1000 像素的图像，有 1 百万个隐层神经元，每个隐层神经元都连接图像的每一个像素点，就有 1000x1000x1000000=10^12 个连接，也就是 10^12 个权值参数。

上图右：局部连接网络，每一个节点与上层节点同位置附件 10x10 的窗口相连接，则 1 百万个隐层神经元就只有 100w 乘以 100，即 10^8 个参数。其权值连接个数比原来减少了四个数量级。

根据 BP 网络信号前向传递过程，我们可以很容易计算网络节点的输出。例如，对于上图中被标注为红色节点的净输入，就等于所有与红线相连接的上一层神经元节点值与红色线表示的权值之积的累加。这样的计算过程，很多书上称其为卷积。

事实上，对于数字滤波而言，其滤波器的系数通常是对称的。否则，卷积的计算需要先反向对折，然后进行乘累加的计算。上述神经网络权值满足对称吗？我想答案是否定的！所以，上述称其为卷积运算，显然是有失偏颇的。但这并不重要，仅仅是一个名词称谓而已。只是，搞信号处理的人，在初次接触卷积神经网络的时候，带来了一些理解上的误区。

卷积神经网络另外一个特性是权值共享(即同一个特征映射上的神经元使用相同的卷积核)。例如，就上面右边那幅图来说，权值共享，也就是说所有的红色线标注的连接权值相同。这一点，初学者容易产生误解。

上面描述的只是单层网络结构，前 A&T Shannon Lab 的 Yann LeCun 等人据此提出了基于卷积神经网络的一个文字识别系统 LeNet-5。该系统 90 年代就被用于银行手写数字的识别。

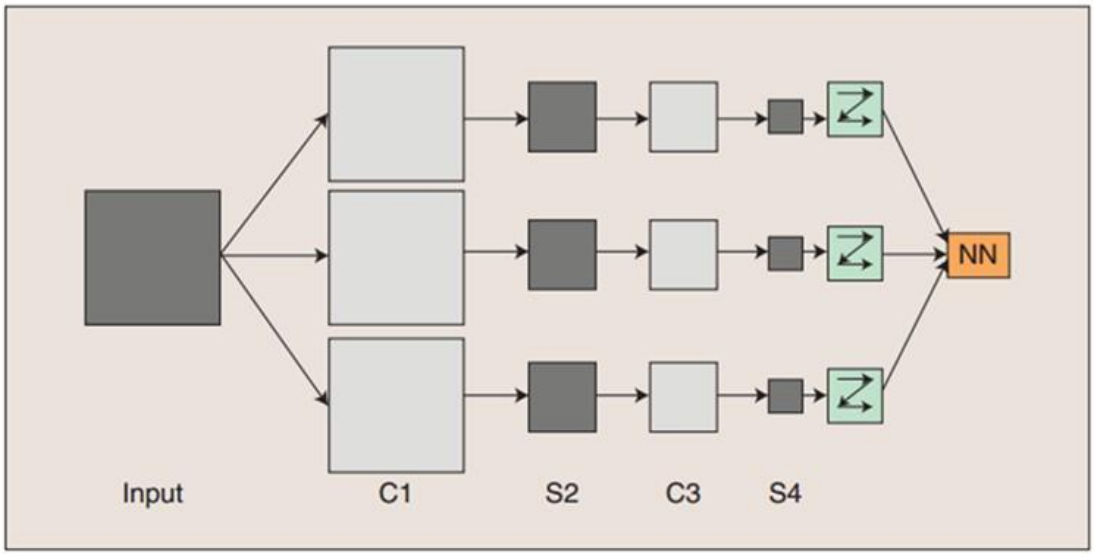
2、 CNN 的结构

卷积网络是为识别二维形状而特殊设计的一个多层感知器，这种网络结构对平移、比例缩放、倾斜或者共他形式的变形具有高度不变性。这些良好的性能是网络在有监督方式下学会的，网络的结构主要有稀疏连接和权值共享两个特点，包括如下形式的约束：

1、 特征提取。每一个神经元从上一层的局部接受域得到突触输人，因而迫使它提取局部特征。一旦一个特征被提取出来，只要它相对于其他特征的位置被近似地保留下来，它的精确位置就变得没有那么重要了。

2 、特征映射。网络的每一个计算层都是由多个特征映射组成的，每个特征映射都是平面形式的。平面中单独的神经元在约束下共享 相同的突触权值集，这种结构形式具有如下的有益效果：a.平移不变性。b.自由参数数量的缩减(通过权值共享实现)。

3、子抽样。每个卷积层后面跟着一个实现局部平均和子抽样的计算层，由此特征映射的分辨率降低。这种操作具有使特征映射的输出对平移和其他 形式的变形的敏感度下降的作用。



图：卷积神经网络的概念示范

输入图像通过和三个可训练的滤波器(卷积核)和可加偏置进行卷积，卷积后在 C1 层产生三个特征映射图，然后特征映射图中每组的四个像素再进行求和，加权值，加偏置，通过一个 Sigmoid 函数得到三个 S2 层的特征映射图。这些映射图再通过滤波得到 C3 层。这个层级结构再和 S2 一样产生 S4。最终，这些像素值被光栅化，并连接成一个向量输入到传统的神经网络，得到输出。

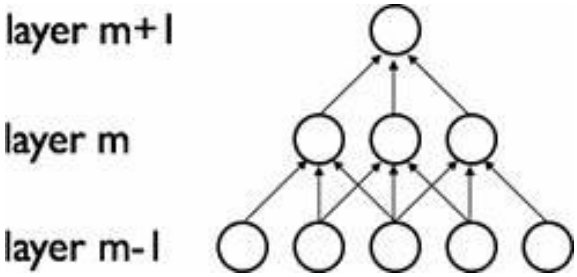
一般地，C 层为特征提取层，每个神经元的输入与前一层的局部感受野相连，并提取该局部的特征，一旦该局部特征被提取后，它与其他特征间的位置关系也随之确定下来；S 层是特征映射层，网络的每个计算层由多个特征映射组成，每个特征映射为一个平面，平面上所有神经元的权值相等。特征映射结构采用影响函数核小的 sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。

此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数，降低了网络参数选择的复杂度。卷积神经网络中的每一个特征提取层（C-层）都紧跟着一个用来求局部平均与二次提取的计算层（S-层），这种特有的两次特征提取结构使网络在识别时对输入样本有较高的畸变容忍能力。

2.1 稀疏连接(Sparse Connectivity)

卷积网络通过在相邻两层之间强制使用局部连接模式来利用图像的空间局部特性，在第 m 层的隐层单元只与第 m-1 层的输入单元的局部区域有连接 ,第 m-1 层的这些局部 区域被称为空间连续的接受域。我们可以将这种结构描述如下：

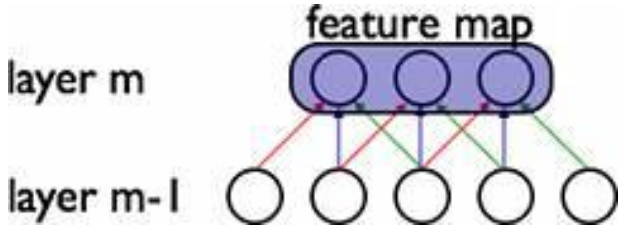
设第 m-1 层为视网膜输入层，第 m 层的接受域的宽度为 3，也就是说该层的每个单元与且仅与输入层的 3 个相邻的神经元相连 ,第 m 层与第 m+1 层具有类似的链接规则，如下图所示。



可以看到 m+1 层的神经元相对于第 m 层的接受域的宽度也为 3 ,但相对于输入层的接受域为 5，这种结构将学习到的过滤器（对应于输入信号中被最大激活的单元）限制在局部空间 模式（因为每个单元对它接受域外的 variation 不做反应）。从上图也可以看出，多个这样的层堆叠起来后，会使得过滤器（不再是线性的）逐渐成为全局的（也就是覆盖到了更 大的视觉区域）。例如上图中第 m+1 层的神经元可以对宽度为 5 的输入进行一个非线性的特征编码。

2.2 权值共享(Shared Weights)

在卷积网络中，每个稀疏过滤器 hi 通过共享权值都会覆盖整个可视域，这些共享权值的单元构成一个特征映射，如下图所示。



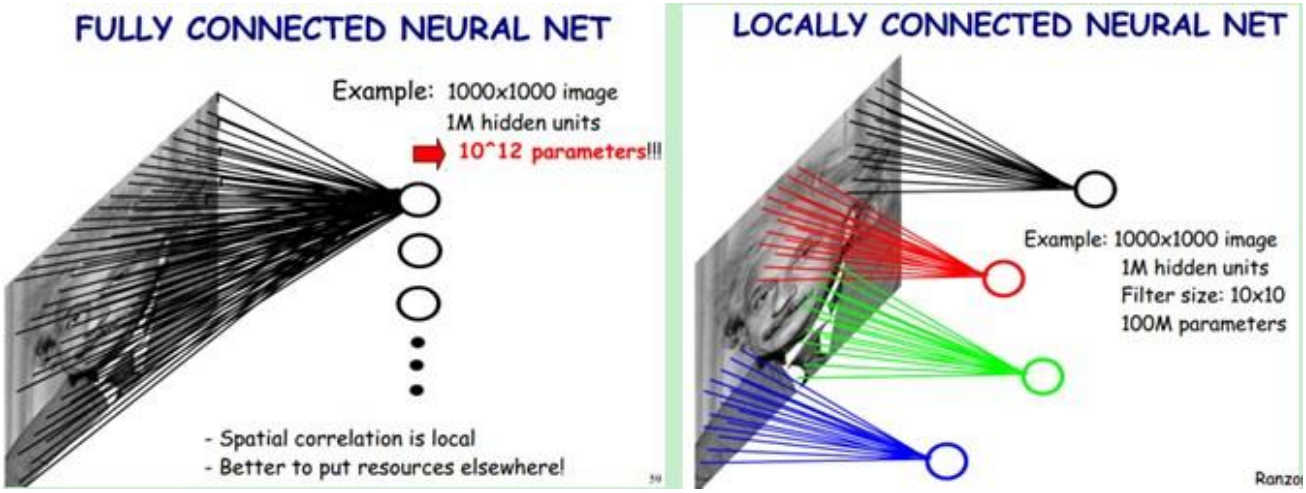
在图中，有 3 个隐层单元，他们属于同一个特征映射。同种颜色的连接权值是相同的，我们仍然可以使用梯度下降的方法来学习这些权值，只需要对原始算法做一些小的改动，这里共享权值的梯度是所有共享参数的梯度的总和。我们不禁会问为什么要权重共享呢？一方面，重复单元能够对特征进行识别，而不考虑它在可视域中的位置。另一方面，权值 共享使得我们能更有效的进行特征抽取，因为它极大的减少了需要学习的自由变量的个数。通过控制模型的规模，卷积网络对视觉问题可以具有很好的泛化能力。

举例讲解

下图左：如果我们有 1000x1000 像素的图像，有 1 百万个隐层神经元，如果他们全连接的话（每个隐层神经元都连接图像的每一个像素点），就有 1000x1000x1000000=10^12 个连接，也就是 10^12 个权值参数。

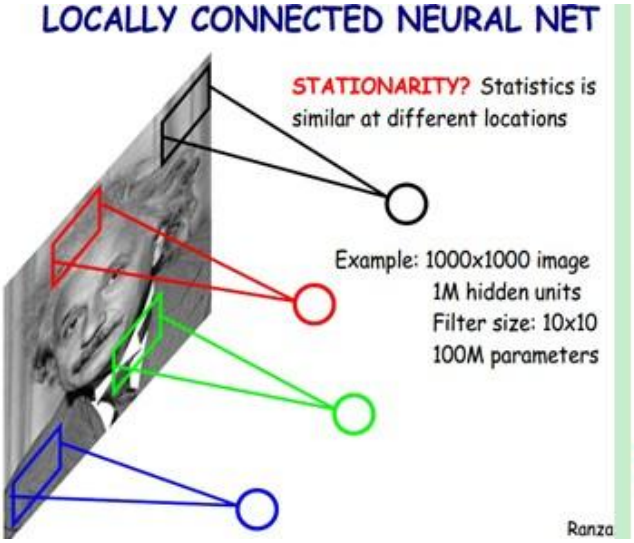
然而图像的空间联系是局部的，就像人是通过一个局部的感受野去感受外界图像一样，每一个神经元都不需要对全局图像做感受，每个神经元只感受局部的图像区域，然后在更高层，将这些感受不同局部的神经元综合起来就可以得到全局的信息了。这样，我们就可以减少连接的数目，也就是减少神经网络需要训练的权值参数的个数了。

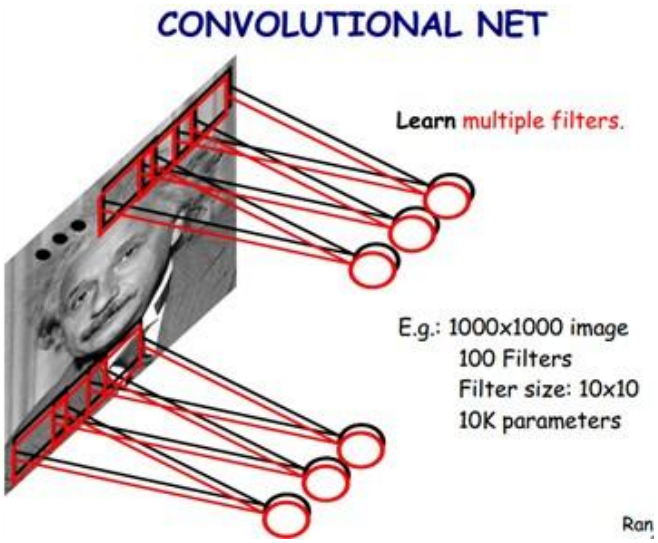
下图右：假如局部感受野是 10x10，隐层每个感受野只需要和这 10x10 的局部图像相连接，所以 1 百万个隐层神经元就只有一亿个连接，即 10^8 个参数。比原来减少了四个 0（数量级），这样训练起来就没那么费力了，但还是感觉很多的啊。



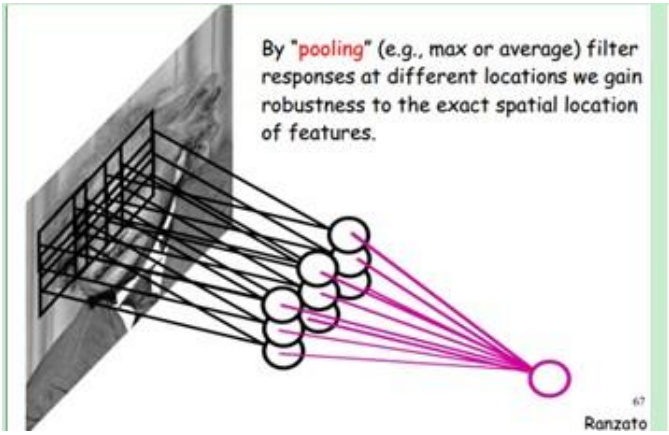
我们知道，隐含层的每一个神经元都连接 10x10 个图像区域，也就是说每一个神经元存在 10x10=100 个连接权值参数（卷积核）。如果每个神经元用的是同一个卷积核去卷积图像，这样我们就只有 100 个参数啊，不管你隐层的神经元个数有多少，两层间的连接只有 100 个参数！这就是权值共享！

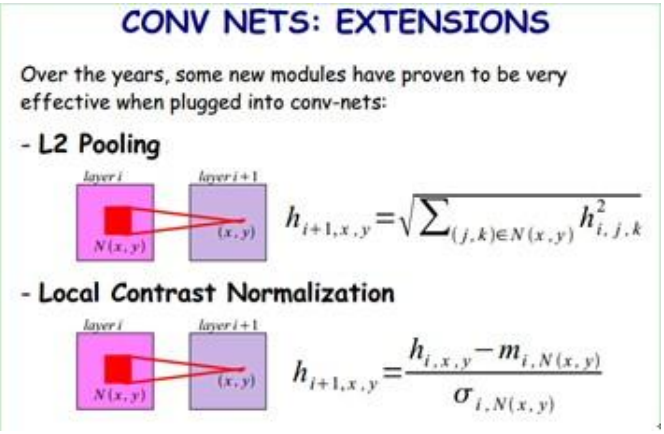
但这样只提取了一种特征，假如一种滤波器，也就是一种卷积核，能够提出图像的一种特征，例如某个方向的边缘。那么我们需要提取不同的特征，就需要多使用几种滤波器。所以假设使用 100 种滤波器，每种滤波器的参数不一样，表示它提出输入图像的不同特征，例如不同的边缘。这样每种滤波器去卷积图像就得到对图像的不同特征的放映，我们称之为 Feature Map。所以 100 种卷积核就有 100 个 Feature Map。这 100 个 Feature Map 就组成了一层神经元。100 种卷积核 x 每种卷积核共享 100 个参数=100x100=10K，也就是 1 万个参数。见下图右：不同的颜色表达不同的滤波器。





刚才说隐层的参数个数和隐层的神经元个数无关，只和滤波器的大小和滤波器种类的多少有关。那么隐层的神经元个数怎么确定呢？它和原图像，也就是输入的大小（神经元个数）、滤波器的大小和滤波器在图像中的滑动步长都有关！例如，我的图像是 1000x1000 像素，而滤波器大小是 10x10，假设滤波器没有重叠，也就是步长为 10，这样隐层的神经元个数就是(1000x1000)/ (10x10)=100x100 个神经元了，假设步长是 8，也就是卷积核会重叠两个像素，那么.....我就不算了，思想懂了就好。注意了，这只是一种滤波器 ,也就是一个 Feature Map 的神经元个数哦 ,如果 100 个 Feature Map 就是 100 倍了。由此可见，图像越大，神经元个数和需要训练的权值参数个数的贫富差距就越大。





需要注意的一点是，上面的讨论都没有考虑每个神经元的偏置部分。所以权值个数需要加 1 。这个也是同一种滤波器共享的。

总之，卷积网络的核心思想是将：局部感受野、权值共享（或者权值复制）以及时间或空间亚采样这三种结构思想结合起来获得了某种程度的位移、尺度、形变不变性。

2.3 The Full Model

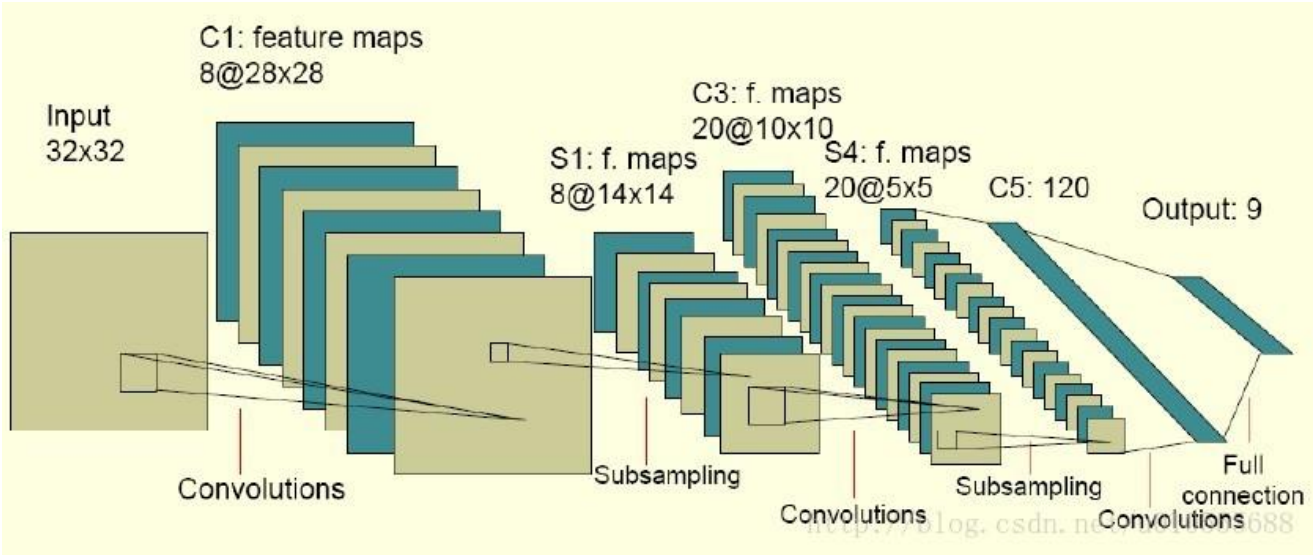
卷积神经网络是一个多层的神经网络，每层由多个二维平面组成，而每个平面由多个独立神经元组成。网络中包含一些简单元和复杂元，分别记为 S-元 和 C-元。S-元聚合在一起组成 S-面，S-面聚合在一起组成 S-层，用 Us 表示。C-元、C-面和 C-层(Us)之间存在类似的关系。网络的任一中间级由 S-层与 C-层 串接而成，而输入级只含一层，它直接接受二维视觉模式，样本特征提取步骤已嵌入到卷积神经网络模型的互联结构中。

一般地，Us 为特征提取层(子采样层)，每个神经元的输入与前一层的局部感受野相连，并提取该局部的特征，一旦该局部特征被提取后，它与其他特征间的位置关系 也随之确定下来；

Uc 是特征映射层(卷积层)，网络的每个计算层由多个特征映射组成，每个特征映射为一个平面，平面上所有神经元的权值相等。特征映射结构采用 影响函数核小的 sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。此外，由于 一个映射面上的神经元共享权值，因而减少了网络自由参数的个数，降低了网络参数选择的复杂度。卷积神经网络中的每一个特征提取层(S-层)都紧跟着一个 用来求局部平

均与二次提取的计算层(C-层)，这种特有的两次特征提取结构使网络在识别时对输入样本有较高的畸变容忍能力。

下图是一个卷积网络的实例，在博文”Deep Learning 模型之：CNN 卷积神经网络（二）文字识别系统 LeNet-5 “中有详细讲解：



图中的卷积网络工作流程如下，输入层由 32×32 个感知节点组成，接收原始图像。然后，计算流程在卷积和子抽样之间交替进行，如下所述：

第一隐藏层进行卷积，它由 8 个特征映射组成，每个特征映射由 28×28 个神经元组成 ,每个神经元指定一个 5×5 的接受域 ,这 28×28 个神经元共享 5×5 个权值参数，即卷积核；

第二隐藏层实现子 抽样和局部平均，它同样由 8 个特征映射组成，但其每个特征映射由 14×14 个神经元组成。每个神经元具有一个 2×2 的接受域 ,一个可训练 系数，一个可训练偏置和一个 sigmoid 激活函数。可训练系数和偏置控制神经元的操作点；

第三隐藏层进行第二次卷积，它由 20 个特征映射组 成，每个特征映射由 10×10 个神经元组成。该隐藏层中的每个神经元可能具有和下一个隐藏层几个特征映射相连的突触连接，它以与第一个卷积 层相似的方式操作。

第四个隐藏层进行第二次子抽样和局部平均汁算。它由 20 个特征映射组成，但每个特征映射由 5×5 个神经元组成，它以 与第一次抽样相似的方式操作。

第五个隐藏层实现卷积的最后阶段，它由 120 个神经元组成，每个神经元指定一个 5×5 的接受域。

最后是个全连接层，得到输出向量。

相继的计算层在卷积和抽样之间的连续交替，我们得到一个“双尖塔”的效果，也就是在每个卷积或抽样层，随着空 间分辨率下降，与相应的前一层相比特征映射的数量增加。卷积之后进行子抽样的思想是受到动物视觉系统中的“简单的”细胞后面跟着“复杂的”细胞的想法的启发而产生的。

图中所示的多层感知器包含近似 100000 个突触连接，但只有大约 2600 个自由参数(每个特征映射为一个平面，平面上所有神经元的权值相等)。自由参数在数量上显著地减少是通过权值共享获得 的，学习机器的能力(以 VC 维的形式度量)因而下降，这又提高它的泛化能力。而且它对自由参数的调整通过反向传播学习的随机形式来实 现。另一个显著的特点是使用权值共享使得以并行形式实现卷积网络变得可能。这是卷积网络对全连接的多层感知器而言的另一个优点。

3、 CNN 的训练

神经网络用于模式识别的主流是有指导学习网络，无指导学习网络更多的是用于聚类分析。对于有指导的模式识别，由于任一样本的类别是已知的，样本在空间的分布不再是依据其自然分布倾向来划分，而是要根据同类样本在空间的分布及不同类样本之间的分离程度找一种适当的空间划分方法，或者找到一个分类边界，使得不同类

样本分别位于不同的区域内。这就需要一个长时间且复杂的学习过程，不断调整用以划分样本空间的分类边界的位置，使尽可能少的样本被划分到非同类区域中。

卷积网络在本质上是一种输入到输出的映射，它能够学习大量的输入与输出之间的映射关系，而不需要任何输入和输出之间的精确的数学表达式，只要用已知的模式对卷积网络加以训练，网络就具有输入输出对之间的映射能力。卷积网络执行的是有导师训练，所以其样本集是由形如：（输入向量，理想输出向量）的向量对构成的。所有这些向量对，都应该是来源于网络即将模拟的系统的实际“运行”结果。它们可以是从实际运行系统中采集来的。在开始训练前，所有的权都应该用一些不同的小随机数进行初始化。“小随机数”用来保证网络不会因权值过大而进入饱和状态，从而导致训练失败；“不同”用来保证网络可以正常地学习。实际上，如果用相同的数去初始化权矩阵，则网络无能力学习。

训练算法与传统的 BP 算法差不多。主要包括 4 步，这 4 步被分为两个阶段：

第一阶段，向前传播阶段

a）从样本集中取一个样本(X,Yp)，将 X 输入网络；

b）计算相应的实际输出 Op。

在此阶段，信息从输入层经过逐级的变换，传送到输出层。这个过程也是网络在完成训练后正常运行时执行的过程。在此过程中，网络执行的是计算（实际上就是输入与每层的权值矩阵相点乘，得到最后的输出结果）：

Op=Fn（...（F2（F1（XpW（1））W（2））...）W（n））

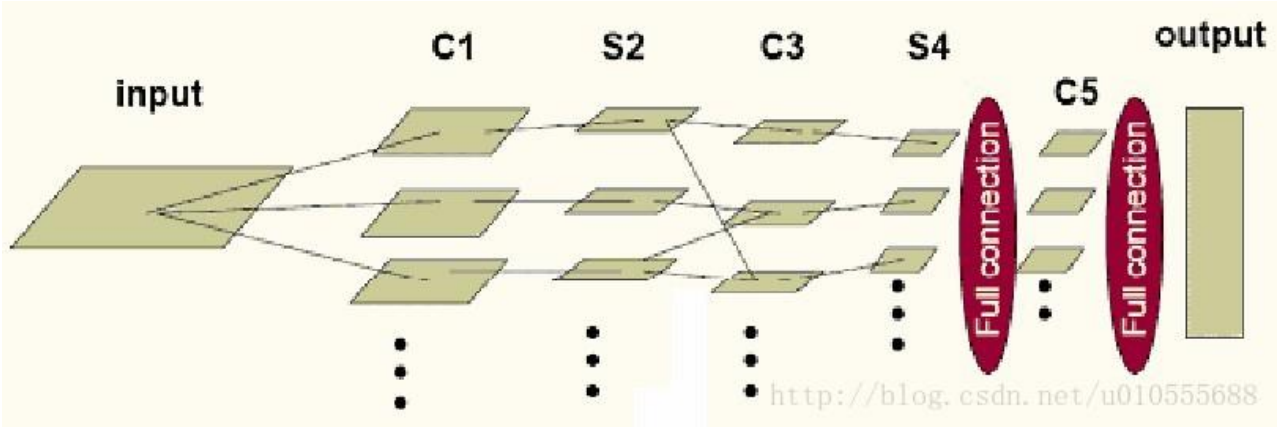
第二阶段，向后传播阶段

a）算实际输出 Op 与相应的理想输出 Yp 的差；

b）按极小化误差的方法反向传播调整权矩阵。

4、 CNN 的学习

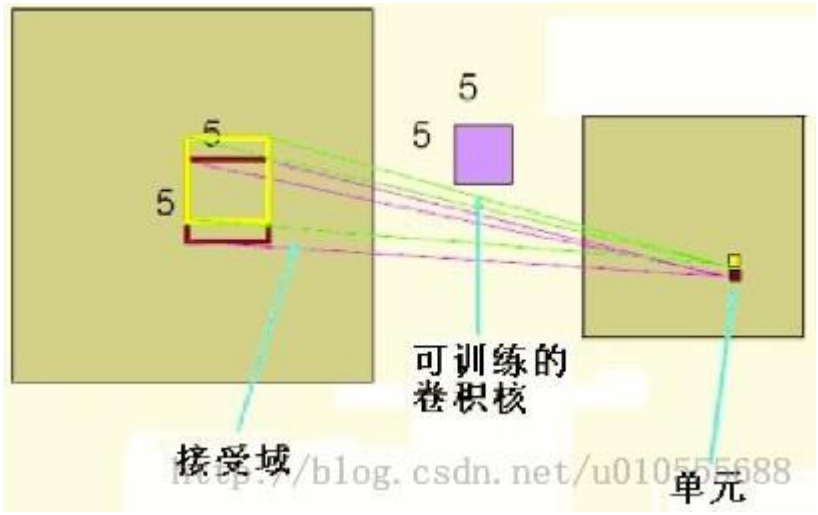
总体而言，卷积网络可以简化为下图所示模型：



其中，input 到 C1、S4 到 C5、C5 到 output 是全连接，C1 到 S2、C3 到 S4 是一一对应的连接，S2 到 C3 为了消除网络对称性，去掉了一部分连接， 可以让特征映射更具多样性。需要注意的是 C5 卷积核的尺寸要和 S4 的输出相同，只有这样才能保证输出是一维向量。

4.1 卷积层的学习

卷积层的典型结构如下图所示：



卷积层的前馈运算是通过如下算法实现的：

卷积层的输出= Sigmoid(Sum(卷积) +偏移量)

其中卷积核和偏移量都是可训练的。下面是其核心代码：

[plain]view plaincopy

```
1.  ConvolutionLayer::fprop(input,output) {
2.    //取得卷积核的个数
3.    int n=kernel.GetDim(0);
4.    for (int i=0;i
5.      //第 i 个卷积核对应输入层第 a 个特征映射，输出层的第 b 个特征映射
6.      //这个卷积核可以形象的看作是从输入层第 a 个特征映射到输出层的第 b 个特征映射的一个链接
7.      int a=table[i][0], b=table[i][1];
8.      //用第 i 个卷积核和输入层第 a 个特征映射做卷积
9.      convolution = Conv(input[a],kernel[i]);
10.     //把卷积结果求和
11.     sum[b] +=convolution;
12.   }
13.   for (i=0;i<(int)bias.size();i++) {
14.     //加上偏移量
15.     sum[i] += bias[i];
16.   }
17.   //调用 Sigmoid 函数
18.   output = Sigmoid(sum);
19. }
```

其中，input 是 n1×n2×n3 的矩阵，n1 是输入层特征映射的个数，n2 是输入层特征映射的宽度，n3 是输入层特征映射的高度。output, sum, convolution, bias 是 n1×(n2-kw+1)×(n3-kh+1)的矩阵，kw,kh 是卷积核的宽度高度(图中是 5×5)。kernel 是卷积核矩阵。table 是连接表，即如果第 a 输入和第 b 个输出之间 有连接，table 里就会有[a,b]这一项，而且每个连接都对应一个卷积核。

卷积层的反馈运算的核心代码如下：

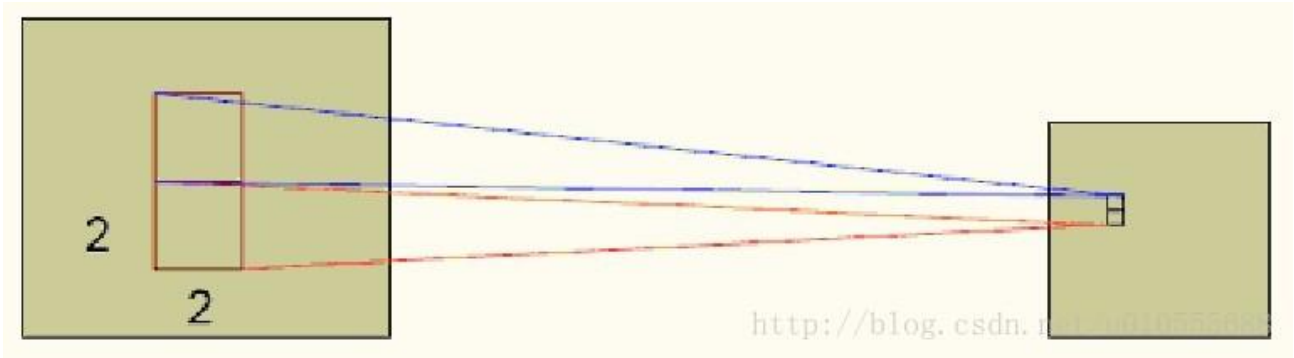
[plain]view plaincopy

```
1.  ConvolutionLayer::bprop(input,output,in_dx,out_dx) {
2.    //梯度通过 DSigmoid 反传
3.    sum_dx = DSigmoid(out_dx);
4.    //计算 bias 的梯度
5.    for (i=0;i
6.       bias_dx[i] = sum_dx[i];
7.    }
8.    //取得卷积核的个数
9.    int n=kernel.GetDim(0);
10.   for (int i=0;i
11.   {
12.      int a=table[i][0],b=table[i][1];
13.      //用第 i 个卷积核和第 b 个输出层反向卷积（即输出层的一点乘卷积模板返回给输入层），并把结果累加到第 a 个
      输入层
14.      input_dx[a] += DConv(sum_dx[b],kernel[i]);
15.      //用同样的方法计算卷积模板的梯度
16.      kernel_dx[i] += DConv(sum_dx[b],input[a]);
17.   }
18. }
```

其中 in_dx,out_dx 的结构和 input,output 相同，代表的是相应点的梯度。

4.2 子采样层的学习

子采样层的典型结构如下图所示：



类似的子采样层的输出的计算式为：

输出= Sigmoid(采样*权重　+偏移量)

其核心代码如下：

[plain]view plaincopy

```
1.  SubSamplingLayer::fprop(input,output) {
2.    int n1= input.GetDim(0);
3.    int n2= input.GetDim(1);
4.    int n3= input.GetDim(2);
5.    for (int i=0;i
6.       for (int j=0;j
7.          for (int k=0;k
8.             //coeff 是可训练的权重，sw 、sh 是采样窗口的尺寸。
9.             sub[i][j/sw][k/sh] += input[i][j][k]*coeff[i];
10.         }
11.      }
12.   }
13.   for (i=0;i
14.      //加上偏移量
15.      sum[i] = sub[i] + bias[i];
16.   }
17.   output = Sigmoid(sum);
18. }
```

子采样层的反馈运算的核心代码如下：

[plain]view plaincopy

```
1.  SubSamplingLayer::bprop(input,output,in_dx,out_dx) {
2.    //梯度通过 DSigmoid 反传
```

```
3.    sum_dx = DSigmoid(out_dx);
4.    //计算 bias 和 coeff 的梯度
5.    for (i=0;i
6.        coeff_dx[i] = 0;
7.        bias_dx[i] = 0;
8.        for (j=0;j
9.            for (k=0;k
10.               coeff_dx[i] += sub[j][k]*sum_dx[i][j][k];
11.               bias_dx[i] += sum_dx[i][j][k]);
12.           }
13.   }
14.   for (i=0;i
15.       for (j=0;j
16.           for (k=0;k
17.               in_dx[i][j][k] = coeff[i]*sum_dx[i][j/sw][k/sh];
18.           }
19.   }
20. }
```

5、CNN 的优点

卷积神经网络 CNN 主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于 CNN 的特征检测层通过训练数据进行学习，所以在使用 CNN 时，避免了显式的特征抽取，而隐式地从训练数据中进行学习；再者由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。

流的分类方式几乎都是基于统计特征的，这就意味着在进行分辨前必须提取某些特征。然而，显式的特征提取并不容易，在一些应用问题中也并非总是可靠的。卷积神经网络，它避免了显式的特征取样，隐式地从训练数据中进行学习。这使得卷积神

经网络明显有别于其他基于神经网络的分类器，通过结构重组和减少权值将特征提取功能融合进多层感知器。它可以直接处理灰度图片，能够直接用于处理基于图像的分类。

卷积网络较一般神经网络在图像处理方面有如下优点： a）输入图像和网络的拓扑结构能很好的吻合；b）特征提取和模式分类同时进行，并同时在训练中产生；c）权重共享可以减少网络的训练参数，使神经网络结构变得更简单，适应性更强。

6、CNN 的实现问题

CNNs 中这种层间联系和空域信息的紧密关系，使其适于图像处理和理解。而且，其在自动提取图像的显著特征方面还表现出了比较优的性能。在一些例子当中，Gabor 滤波器已经被使用在一个初始化预处理的步骤中，以达到模拟人类视觉系统对视觉刺激的响应。在目前大部分的工作中，研究者将 CNNs 应用到了多种机器学习问题中，包括人脸识别，文档分析和语言检测等。为了达到寻找视频中帧与帧之间的相干性的目的，目前 CNNs 通过一个时间相干性去训练，但这个不是 CNNs 特有的。

由于卷积神经网络采用 BP 网络相同的算法。所以，采用现有 BP 网络就可以实现。开源的神经网络代码 FAAN 可以利用。这个开源的实现采用了一些代码优化技术，有双精度，单精度，定点运算三个不同的版本。

由于经典的 BP 网络是一个一维节点分布排列，而卷积神经网络是二维网络结构。所以，要把卷积神经网络的每一层，按照一定的顺序和规则映射为一维节点分布，然后，按照这个分布创建一个多层反向传播算法的网络结构，就可以按照一般的 BP 训练算法去学习网络参数。对于实际环境中新样本的预测，也采用 BP 算法中相同信号前向传递算法进行。具体细节也可以参考网上的一个开源代码，链接如下：

http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi

注：这个代码在创建 CNN 的时候有个明显的 BUG，如果你看明白了我上面对简化的 LeNet-5 的结构描述，一眼就会找出问题所在。

参考：

卷积神经网络（CNN）

GitHub 卷及神经网络代码（MATLAB）

CNN 代码理解（matlab）

http://blog.csdn.net/nan355655600/article/details/17690029

http://blog.csdn.net/zouxy09/article/details/8782018

[6] CNN 卷积神经网络（二）

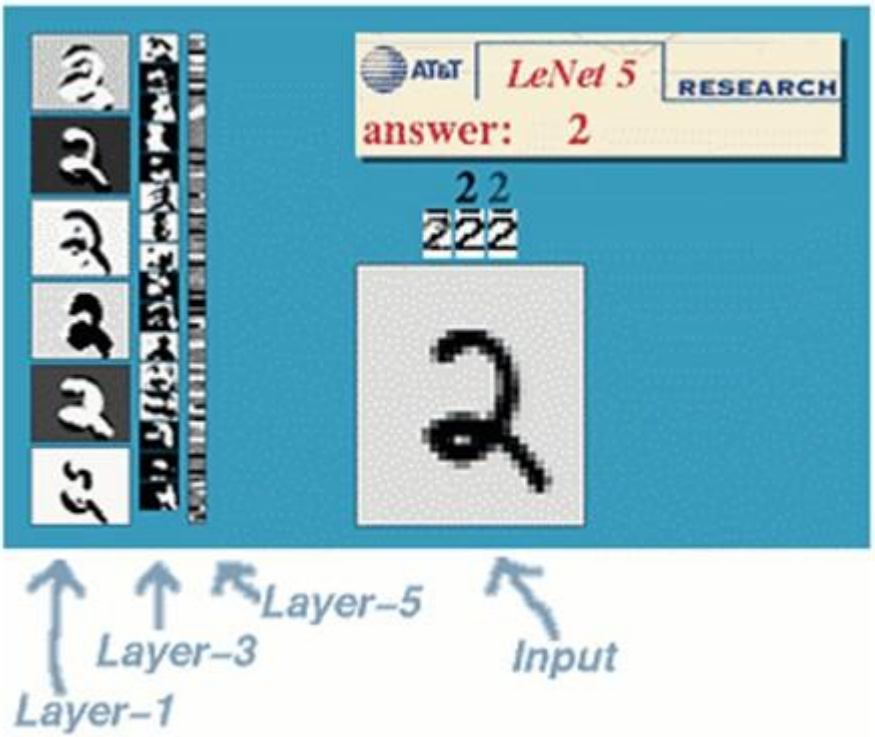
文字识别系统 LeNet-5

在经典的模式识别中，一般是事先提取特征。提取诸多特征后，要对这些特征进行相关性分析，找到最能代表字符的特征，去掉对分类无关和自相关的特征。然而，这些特征的提取太过依赖人的经验和主观意识，提取到的特征的不同对分类性能影响很大，甚至提取的特征的顺序也会影响最后的分类性能。同时，图像预处理的好坏也会影响到提取的特征。那么，如何把特征提取这一过程作为一个自适应、自学习的过程，通过机器学习找到分类性能最优的特征呢？

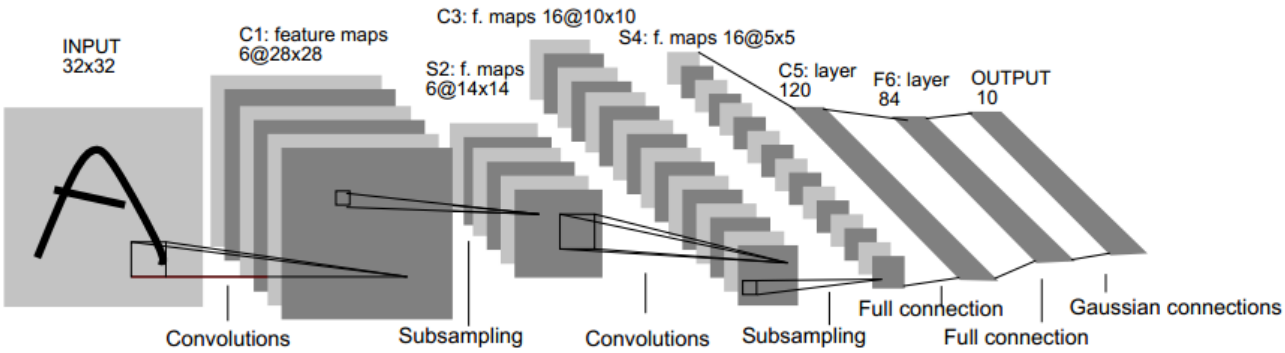
卷积神经元每一个隐层的单元提取图像局部特征，将其映射成一个平面，特征映射函数采用 sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。每个神经元与前一层的局部感受野相连。注意前面我们说了，不是局部连接的神经元权值相同，而是同一平面层的神经元权值相同，有相同程度的位移、旋转不变性。每个特征提取后都紧跟着一个用来求局部平均与二次提取的亚取样层。这种特有的两次特征提取结构使得网络对输入样本有较高的畸变容忍能力。也就是说，卷积神经网络通过局部感受野、共享权值和亚取样来保证图像对位移、缩放、扭曲的鲁棒性。

文字识别系统 LeNet-5

一种典型的用来识别数字的卷积网络是 LeNet-5（效果和 paper 见这里）。当年美国大多数银行就是用它来识别支票上面手写数字的。能够达到这种商用的地步，它的准确性可想而知。毕竟目前学术界和工业界的结合是最受争议的。



那下面咱们也用这个例子来说明下。



LeNet-5 共有 7 层，不包含输入，每层都包含可训练参数（连接权重）。输入图像为 32*32 大小。这要比 Mnist 数据库（一个公认的手写数据库）中最大的字母还大。这样做的原因是希望潜在的明显特征如笔画断点或角点能够出现在最高层特征监测子感受野的中心。

首先，简要解释下上面这个用于文字识别的 LeNet-5 深层卷积网络：

1. 输入图像是 32x32 的大小，局部滑动窗的大小是 5x5 的，由于不考虑对图像的边界进行拓展，则滑动窗将有 28x28 个不同的位置，也就是 C1 层的大小是 28x28。这里设定有 6 个不同的 C1 层，每一个 C1 层内的权值是相同的。

2. S2 层是一个下采样层。简单的说，由 4 个点下采样为 1 个点，也就是 4 个数的加权平均。但在 LeNet-5 系统，下采样层比较复杂，因为这 4 个加权系数也需要学习得到，这显然增加了模型的复杂度。在斯坦福关于深度学习的教程中，这个过程叫做 Pool。

3. 根据对前面 C1 层同样的理解 ,我们很容易得到 C3 层的大小为 10x10. 只不过，C3 层的变成了 16 个 10x10 网络! 试想一下，如果 S2 层只有 1 个平面，那么由 S2 层得到 C3 就和由输入层得到 C1 层是完全一样的。但是，S2 层由多层，那么，我们只需要按照一定的顺利组合这些层就可以了。具体的组合规则，在 LeNet-5 系统中给出了下面的表格：

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

简单的说，例如对于 C3 层第 0 张特征图，其每一个节点与 S2 层的第 0 张特征图，第 1 张特征图，第 2 张特征图，总共 3 个 5x5 个节点相连接。后面依次类推，C3 层每一张特征映射图的权值是相同的。

4. S4 层是在 C3 层基础上下采样，前面已述。在后面的层由于每一层节点个数比较少，都是全连接层，这个比较简单，不再赘述。

涉及问题：

1、每个图如何卷积：

（1）一个图如何变成几个？

（2）卷积核如何选择？

2、节点之间如何连接？

3、S2-C3 如何进行分配？

4、16-120 全连接如何连接？

5、最后 output 输出什么形式？

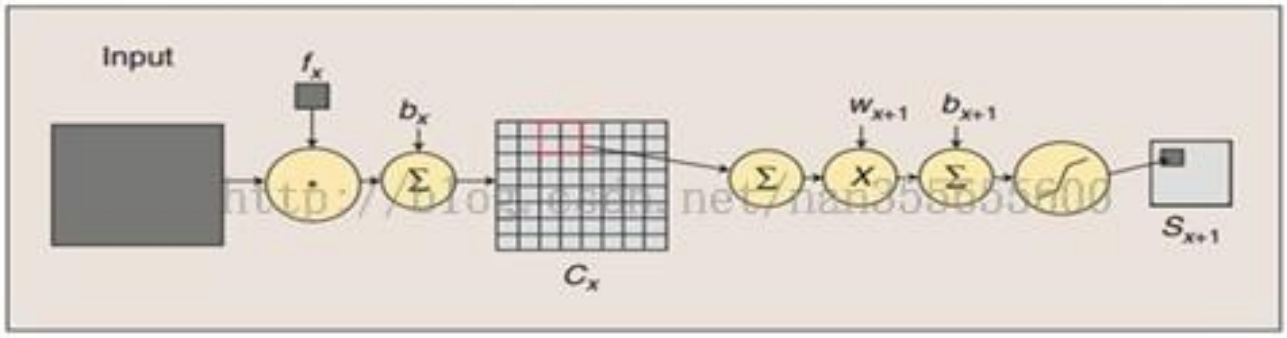
①各个层解释：

我们先要明确一点：每个层有多个 Feature Map，每个 Feature Map 通过一种卷积滤波器提取输入的一种特征，然后每个 Feature Map 有多个神经元。

(1) C1 层是一个卷积层（为什么是卷积？卷积运算一个重要的特点就是，通过卷积运算，可以使原信号特征增强，并且降低噪音），由 6 个特征图 Feature Map 构成。特征图中每个神经元与输入中 5*5 的邻域相连。特征图的大小为 28*28，这样能防止

输入的连接掉到边界之外（是为了 BP 反馈时的计算，不致梯度损失，个人见解）。C1 有 156 个可训练参数（每个滤波器 5*5=25 个 unit 参数和一个 bias 参数，一共 6 个滤波器，共(5*5+1)*6=156 个参数），共 156*(28*28)=122,304 个连接。

(2) S2 层是一个下采样层（为什么是下采样？利用图像局部相关性的原理，对图像进行子抽样，可以减少数据处理量同时保留有用信息），有 6 个 14*14 的特征图。特征图中的每个单元与 C1 中相对应特征图的 2*2 邻域相连接。S2 层每个单元的 4 个输入相加，乘以一个可训练参数，再加上一个可训练偏置。结果通过 sigmoid 函数计算。可训练系数和偏置控制着 sigmoid 函数的非线性程度。如果系数比较小，那么运算近似于线性运算，亚采样相当于模糊图像。如果系数比较大，根据偏置的大小亚采样可以被看成是有噪声的“或”运算或者有噪声的“与”运算。每个单元的 2*2 感受野并不重叠，因此 S2 中每个特征图的大小是 C1 中特征图大小的 1/4（行和列各 1/2）。S2 层有 12 个可训练参数和 5880 个连接。



图：卷积和子采样过程

卷积过程包括：用一个可训练的滤波器 fx 去卷积一个输入的图像（第一阶段是输入的图像，后面的阶段就是卷积特征 map 了），然后加一个偏置 bx，得到卷积层 Cx。子采样过程包括：每邻域四个像素求和变为一个像素，然后通过标量 Wx+1 加权，再增加偏置 bx+1，然后通过一个 sigmoid 激活函数，产生一个大概缩小四倍的特征映射图 Sx+1。

所以从一个平面到下一个平面的映射可以看作是作卷积运算，S-层可看作是模糊滤波器，起到二次特征提取的作用。隐层与隐层之间空间分辨率递减，而每层所含的平面数递增，这样可用于检测更多的特征信息。

(3) C3 层也是一个卷积层，它同样通过 5x5 的卷积核去卷积层 S2，然后得到的特征 map 就只有 10x10 个神经元，但是它有 16 种不同的卷积核，所以就存在 16 个特征 map 了。这里需要注意的一点是：C3 中的每个特征 map 是连接到 S2 中的所有 6 个或者几个特征 map 的，表示本层的特征 map 是上一层提取到的特征 map 的不同组合(这个做法也并不是唯一的）。（看到没有，这里是组合，就像之前聊到的人的视觉系统一样，底层的结构构成上层更抽象的结构，例如边缘构成形状或者目标的部分）。

刚才说 C3 中每个特征图由 S2 中所有 6 个或者几个特征 map 组合而成。为什么不把 S2 中的每个特征图连接到每个 C3 的特征图呢？原因有 2 点。第一，不完全的连接机制将连接的数量保持在合理的范围内。第二，也是最重要的，其破坏了网络的对称性。由于不同的特征图有不同的输入，所以迫使他们抽取不同的特征（希望是互补的）。

例如，存在的一个方式是：C3 的前 6 个特征图以 S2 中 3 个相邻的特征图子集为输入。接下来 6 个特征图以 S2 中 4 个相邻特征图子集为输入。然后的 3 个以不相邻的 4 个特征图子集为输入。最后一个将 S2 中所有特征图为输入。这样 C3 层有 1516 个可训练参数和 151600 个连接。

(4) S4 层是一个下采样层，由 16 个 5*5 大小的特征图构成。特征图中的每个单元与 C3 中相应特征图的 2*2 邻域相连接，跟 C1 和 S2 之间的连接一样。S4 层有 32 个可训练参数（每个特征图 1 个因子和一个偏置）和 2000 个连接。

(5) C5 层是一个卷积层，有 120 个特征图。每个单元与 S4 层的全部 16 个单元的 5*5 邻域相连。由于 S4 层特征图的大小也为 5*5（同滤波器一样），故 C5 特征图的

大小为 1*1：这构成了 S4 和 C5 之间的全连接。之所以仍将 C5 标示为卷积层而非全相联层，是因为如果 LeNet-5 的输入变大，而其他的保持不变，那么此时特征图的维数就会比 1*1 大。C5 层有 48120 个可训练连接。

(6) F6 层有 84 个单元（之所以选这个数字的原因来自于输出层的设计），与 C5 层全相连。有 10164 个可训练参数。如同经典神经网络，F6 层计算输入向量和权重向量之间的点积，再加上一个偏置。然后将其传递给 sigmoid 函数产生单元 i 的一个状态。

(7) 最后，输出层由欧式径向基函数（Euclidean Radial Basis Function）单元组成，每类一个单元，每个有 84 个输入。换句话说，每个输出 RBF 单元计算输入向量和参数向量之间的欧式距离。输入离参数向量越远，RBF 输出的越大。一个 RBF 输出可以被理解为衡量输入模式和与 RBF 相关联类的一个模型的匹配程度的惩罚项。用概率术语来说，RBF 输出可以被理解为 F6 层配置空间的高斯分布的负 log-likelihood。给定一个输入模式，损失函数应能使得 F6 的配置与 RBF 参数向量（即模式的期望分类）足够接近。这些单元的参数是人工选取并保持固定的（至少初始时候如此）。这些参数向量的成分被设为-1 或 1。虽然这些参数可以以-1 和 1 等概率的方式任选，或者构成一个纠错码，但是被设计成一个相应字符类的 7*12 大小（即 84）的格式化图片。这种表示对识别单独的数字不是很有用，但是对识别可打印 ASCII 集中的字符串很有用。

用这种分布编码而非更常用的“1 of N”编码用于产生输出的另一个原因是，当类别比较大的时候，非分布编码的效果比较差。原因是大多数时间非分布编码的输出必须为 0。这使得用 sigmoid 单元很难实现。另一个原因是分类器不仅用于识别字母，也用于拒绝非字母。使用分布编码的 RBF 更适合该目标。因为与 sigmoid 不同，他们在输入空间的较好限制的区域内兴奋，而非典型模式更容易落到外边。

RBF 参数向量起着 F6 层目标向量的角色。需要指出这些向量的成分是+1 或-1，这正好在 F6 sigmoid 的范围内，因此可以防止 sigmoid 函数饱和。实际上，+1 和-1 是 sigmoid 函数的最大弯曲的点处。这使得 F6 单元运行在最大非线性范围内。必须避免 sigmoid 函数的饱和，因为这将会导致损失函数较慢的收敛和病态问题。

②问题讲解

第一个问题：

（1）输入-C1

用 6 个 5*5 大小的 patch（即权值，训练得到，随机初始化，在训练过程中调节）对 32*32 图片进行卷积，得到 6 个特征图。

（2）S2-C3

C3 那 16 张 10*10 大小的特征图是怎么来？

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

将 S2 的特征图用 1 个输入层为 150（=5*5*6，不是 5*5）个节点，输出层为 16 个节点的网络进行 convolution。

该第 3 号特征图的值（假设为 H3）是怎么得到的呢？

首先我们把网络 150-16（以后这样表示，表面输入层节点为 150，隐含层节点为 16）中输入的 150 个节点分成 6 个部分，每个部分为连续的 25 个节点。取出倒数第 3

个部分的节点（为 25 个），且同时是与隐含层 16 个节点中的第 4（因为对应的是 3 号，从 0 开始计数的）个相连的那 25 个值，reshape 为 5*5 大小，用这个 5*5 大小的特征 patch 去 convolution S2 网络中的倒数第 3 个特征图，假设得到的结果特征图为 h1。

同理，取出网络 150-16 中输入的倒数第 2 个部分的节点（为 25 个），且同时是与隐含层 16 个节点中的第 5 个相连的那 25 个值，reshape 为 5*5 大小，用这个 5*5 大小的特征 patch 去 convolution S2 网络中的倒数第 2 个特征图 ,假设得到的结果特征图为 h2。

最后，取出网络 150-16 中输入的最后 1 个部分的节点（为 25 个），且同时是与隐含层 16 个节点中的第 5 个相连的那 25 个值，reshape 为 5*5 大小，用这个 5*5 大小的特征 patch 去 convolution S2 网络中的最后 1 个特征图 ,假设得到的结果特征图为 h3。

最后将 h1，h2，h3 这 3 个矩阵相加得到新矩阵 h，并且对 h 中每个元素加上一个偏移量 b，且通过 sigmoid 的激发函数，即可得到我们要的特征图 H3 了。

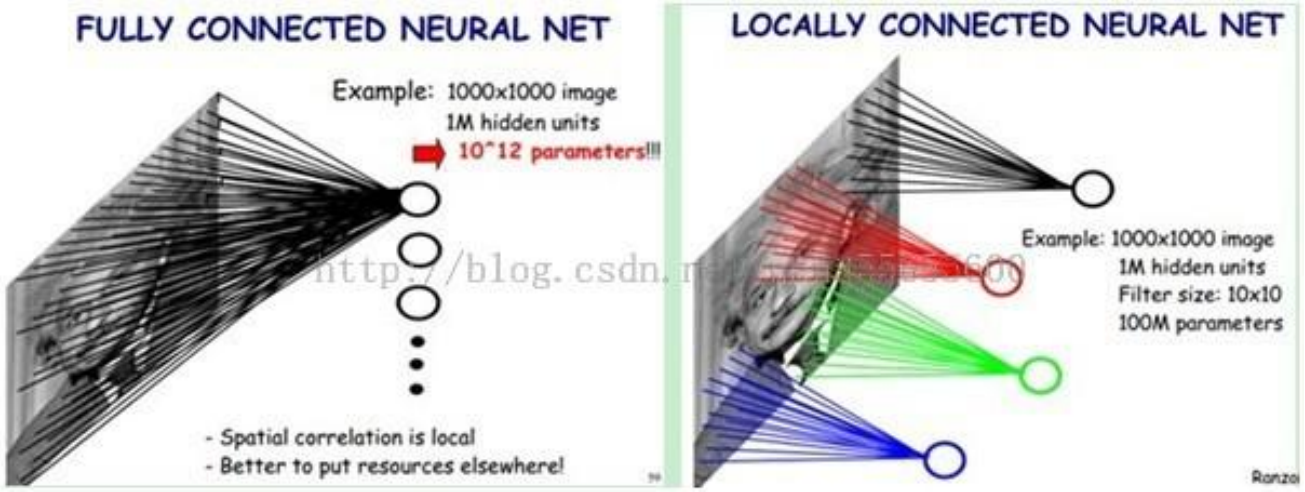
第二个问题:

上图 S2 中为什么是 150 个节点？（涉及到权值共享和参数减少）

CNN 一个牛逼的地方就在于通过感受野和权值共享减少了神经网络需要训练的参数的个数。

下图左：如果我们有 1000x1000 像素的图像，有 1 百万个隐层神经元，那么他们全连接的话（每个隐层神经元都连接图像的每一个像素点），就有 1000x1000x1000000=10^12 个连接，也就是 10^12 个权值参数。然而图像的空间联系是局部的，就像人是通过一个局部的感受野去感受外界图像一样，每一个神经元都不

需要对全局图像做感受，每个神经元只感受局部的图像区域，然后在更高层，将这些感受不同局部的神经元综合起来就可以得到全局的信息了。这样，我们就可以减少连接的数目，也就是减少神经网络需要训练的权值参数的个数了。如下图右：假如局部感受野是 10x10，隐层每个感受野只需要和这 10x10 的局部图像相连接，所以 1 百万个隐层神经元就只有一亿个连接，即 10^8 个参数。比原来减少了四个 0（数量级），这样训练起来就没那么费力了，但还是感觉很多的啊，那还有啥办法没？

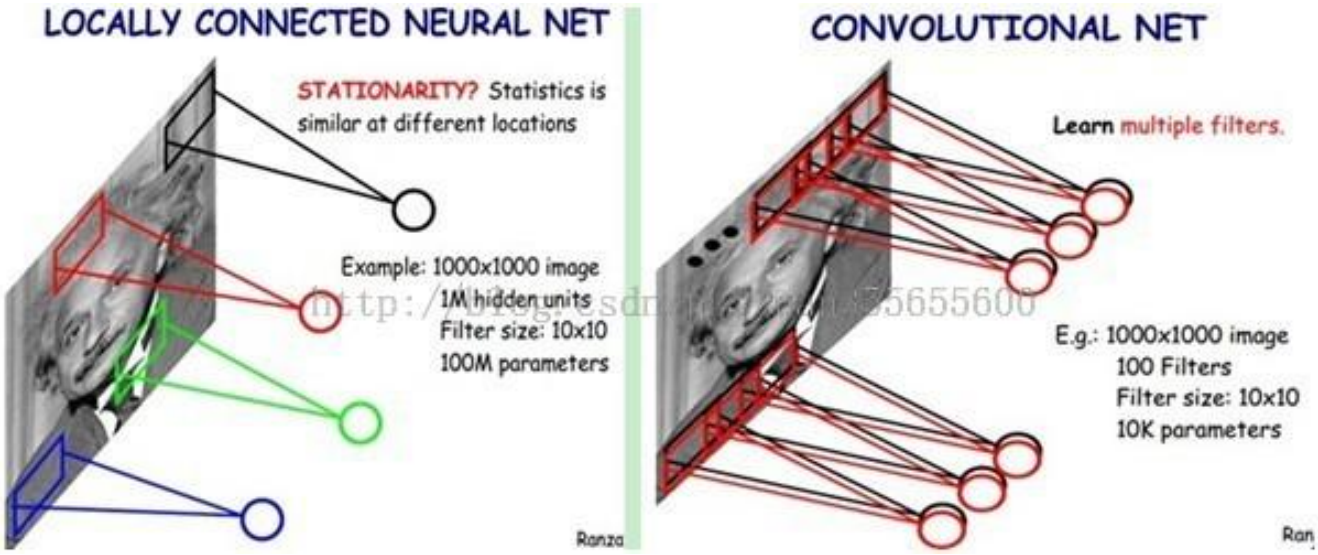


我们知道，隐含层的每一个神经元都连接 10x10 个图像区域，也就是说每一个神经元存在 10x10=100 个连接权值参数。那如果我们每个神经元这 100 个参数是相同的呢？也就是说每个神经元用的是同一个卷积核去卷积图像。这样我们就只有多少个参数？？只有 100 个参数啊！！！亲！不管你隐层的神经元个数有多少，两层间的连接我只有 100 个参数啊！亲！这就是权值共享啊！亲！这就是卷积神经网络的主打卖点啊！亲！也许你会问，这样做靠谱吗？为什么可行呢？这个……共同学习。

好了，你就会想，这样提取特征也忒不靠谱吧，这样你只提取了一种特征啊？对了，真聪明，我们需要提取多种特征对不？假如一种滤波器，也就是一种卷积核就是提出图像的一种特征，例如某个方向的边缘。那么我们需要提取不同的特征，怎么办，加多几种滤波器不就行了吗？对了。所以假设我们加到 100 种滤波器，每种滤波器的

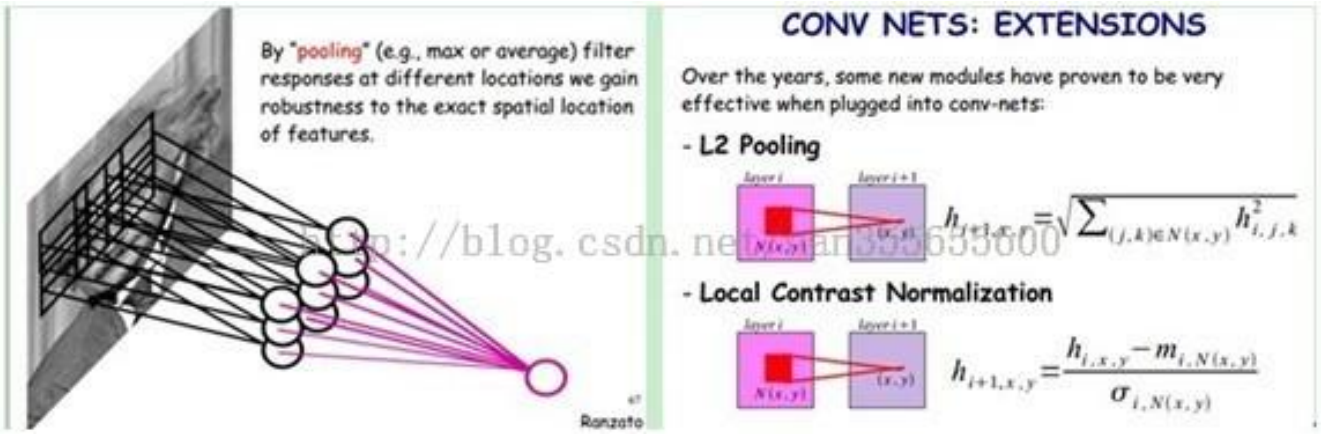
参数不一样，表示它提出输入图像的不同特征，例如不同的边缘。这样每种滤波器去卷积图像就得到对图像的不同特征的放映，我们称之为 Feature Map。所以 100 种卷积核就有 100 个 Feature Map。这 100 个 Feature Map 就组成了一层神经元。到这个时候明了了吧。我们这一层有多少个参数了？100 种卷积核 x 每种卷积核共享 100 个参数=100x100=10K，也就是 1 万个参数。才 1 万个参数啊！亲！

见下图右：不同的颜色表达不同的滤波器。



嘿哟，遗漏一个问题了。刚才说隐层的参数个数和隐层的神经元个数无关，只和滤波器的大小和滤波器种类的多少有关。那么隐层的神经元个数怎么确定呢？它和原图像，也就是输入的大小（神经元个数）、滤波器的大小和滤波器在图像中的滑动步长都有关！例如，我的图像是 1000x1000 像素，而滤波器大小是 10x10，假设滤波器没有重叠，也就是步长为 10，这样隐层的神经元个数就是(1000x1000)/ (10x10)=100x100 个神经元了，假设步长是 8，也就是卷积核会重叠两个像素，那么..... 我就不算了，思想懂了就好。注意了，这只是一种滤波器，也就是一个 Feature Map 的神经元个数哦，如果 100 个 Feature Map 就是 100 倍了。由此可见，图像越大，神经元个数和需要训练的权值参数个数的贫富差距就越大。

所以这里可以知道刚刚 14*14 的图像计算它的节点，按步长为 3 计算，则一幅图可得 5*5 个神经元个数，乘以 6 得到 150 个神经元个数。



需要注意的一点是，上面的讨论都没有考虑每个神经元的偏置部分。所以权值个数需要加 1 。这个也是同一种滤波器共享的。

总之，卷积网络的核心思想是将：局部感受野、权值共享（或者权值复制）以及时间或空间亚采样这三种结构思想结合起来，获得了某种程度的位移、尺度、形变不变性。

第三个问题：

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

如果 C1 层减少为 4 个特征图，同样的 S2 也减少为 4 个特征图，与之对应的 C3 和 S4 减少为 11 个特征图，则 C3 和 S2 连接情况如图：

	0	1	2	3	4	5	6	7	8	9	10
0	X			X	X		X		X	X	X
1	X	X				X	X	X		X	X
2		X	X		X		X	X	X		X
3			X	X		X		X	X	X	X

第四个问题：

全连接：

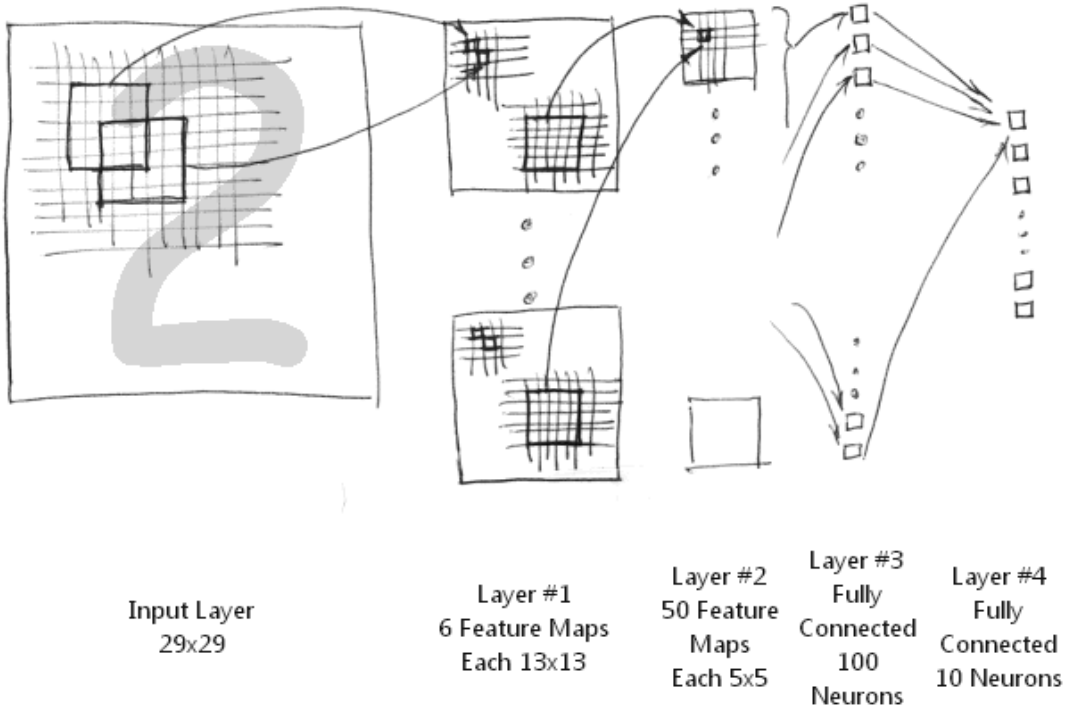
C5 对 C4 层进行卷积操作，采用全连接方式，即每个 C5 中的卷积核均在 S4 所有 16 个特征图上进行卷积操作。

第五个问题：

采用 one-of-c 的方式，在输出结果的 1*10 的向量中最大分量对应位置极为网络输出的分类结果。对于训练集的标签也采用同样的方式编码，例如 1000000000，则表明是数字 0 的分类。

简化的 LeNet-5 系统

简化的 LeNet-5 系统把下采样层和卷积层结合起来，避免了下采样层过多的参数学习过程，同样保留了对图像位移，扭曲的鲁棒性。其网络结构图如下所示：



简化的 LeNet-5 系统包括输入层的话，只有 5 层结构，而原始 LeNet-5 结构不包含输入层就已经是 7 层网络结构了。它实现下采样非常简单，直接取其第一个位置节点上的值可以了。

1、输入层。MNIST 手写数字图像的大小是 28x28 的，这里通过补零扩展为 29x29 的大小。这样输入层神经节点个数为 29x29 等于 841 个。

2、第一层。由 6 张不同的特征映射图组成。每一张特征图的大小是 13x13. 注意，由于卷积窗大小为 5x5，加上下采样过程，易得其大小为 13x13. 所以，　第二层共有 6x13x13 等于 1014 个神经元节点。每一张特征图加上偏置共有 5x5+1 等于 26 个权值需要训练，总共有 6x26 等于 156 个不同的权值。即总共有 1014x156 等于 26364 条连接线。

3、 第二层。由 50 张不同的特征映射图组成。每一张特征图的大小是 5x5. 注意，由于卷积窗大小为 5x5，加上下采样过程，易得其大小为 5x5. 由于上一　层是由多个特征映射图组成，那么，如何组合这些特征形成下一层特征映射图的节点呢？简化的 LeNet-5 系统采用全部所有上层特征图的组合。也就是原始 LeNet-5 特征映射组合图的最后一列的组合方式。因此，总共有 5x5x50 等于 1250 个神经元节点，有 (5x5+1)x6x50 等于 7800 个权值，总共有 1250x26 等于 32500 条连接线。

4、第三层。 这一层是一个一维线性排布的网络节点,与前一层是全连接的网络，其节点个数设为为 100，故而总共有 100x(1250+1)等于 125100 个不同的权值，同时，也有相同数目的连接线。

5、第四层。这一层是网络的输出层 ,如果要识别 0-9 数字的话 ,就是 10 个节点。该层与前一层是全连接的，故而，总共有 10x(100+1)等于 1010 个权值，有相同数目的连接线。

参考：http://blog.csdn.net/nan355655600/article/details/17690029

http://blog.csdn.net/zouxy09/article/details/8782018

[7] CNN 卷积神经网络（三）

CNN 常见问题总结

一、遇到的问题

（1）梯度消失

我在实现过程中犯的第一个错误是没有循序渐进。仗着自己写过一些神经网络的代码以为手到擒来，直接按照 LeNet-5 的结构写，过于复杂的结构给测试和调试都带来了很大的麻烦，可谓不作死就不会死。

简单分析一下 LeNet-5 的结构 :第一层 8 个 5*5 的卷积核 ,第二层分别作 2*2pooling，第三层 16 个 5*5 的卷积核，第四层 2*2pooling，随后是三个节点数分别为 120、84、10 的全连接层（做手写数字识别）。这时的参数个数为卷积核及其偏置+pooling 层放缩系数及偏置+全连接层权重及偏置，全链接层的参数个数比前四层的参数个数多了两个数量级。

过多的参数个数会极大地提升运算时间，降低测试的效率；同时过多的层数伴随着严重的梯度消失问题（vanishing gradient），严重影响训练效果。在上面提到的网络结构下，我发现传到第一层的梯度已经小于 1e-10，只有后面的全连接层在变化，卷积层几乎训不动了。

我对付这个问题的手段包括：

1. 减少层数
2. 增大学习率（learning rate）

3. 用 ReLU 代替 sigmoid

其中前两种虽然也有效果，只能算是权宜之计，第三种才算是正经的解决了这个问题。在采用 ReLU 作为激活函数之后，传到输入层的梯度已经达到 1e-5~1e-4 这个量级。我用 MNIST 数据集里的 5000 个样本训练，1000 个样本测试，发现测试结果差不多，收敛速度快了 2 倍左右。据@kevin 好好学习的说法，ReLU 还使网络的激活具有了稀疏的特性，我对这方面就没研究了。

（2）非线性映射的位置

在前面提到的 Bouvrie 的文章中，非线性映射放在卷积层还是 pooling 层后面都可以，微博上几位大牛也是这个观点。奇怪的是，我在一开始实现的时候把 sigmoid 放在了 pooling 层后面，虽然很渣但至少能跑出结果。后来我把 sigmoid 放在卷积层后面就训不动了，感觉跟梯度消失的时候很像。我猜测也许是卷积层和 pooling 层激活程度不同导致传回去的梯度大小不一样。

（3）权重衰减（weight decay）

因为我用的数据很少（数据多了跑起来太慢），我担心会出现过拟合，所以在 cost function 里加了一项正则项。但结果是不加正则项训练结果很好，一加就出现严重的 under fitting，不管怎么调参数都没用。原来一直听说 CNN 的权重共享就相当于自带某种正则化，现在看起来确实是这样。

（4）随机梯度下降（SGD）的参数选择

最主要的就一个参数，minibatch 的大小。在 Deep Learning Toolbox 的 demo 里，这个值取的是 50。但是我的实验中，似乎 minibatch 取 1 的时候收敛最快。

我的感觉：训练样本顺序随机的话，每次产生的梯度也是随机的，那么 50 个样本平均一下梯度就很小了，也许这样比较稳健，但收敛会比较慢；相反，每个样本更新一次梯度就大得多，不过也许会比较盲目。

另外还有一个参数是 learning rate。在实验中，增大 minibatch 会出现训不动的情况，这时适当增大 learning rate 能够让 training cost 继续下降。

另外 Yann LeCun 似乎说过每一层应该采取不同的 learning rate，不过我没试。

（5）参数的随机初始化

我试过将参数初始化到(0, 1)区间和(-1, 1)区间，感觉似乎差别不大？

（6）增加全连接层数后的性能

关于这点我不是很确定，但似乎除了增加训练时间外没什么实际作用...

二、网络的改进

（1）自动学习组合系数

为了打破对称性，从第一个卷积层到第二个卷积层并不是全链接，例如第二层第一个卷积核只卷积第一层的第 123 个 feature map ,第二层第二个卷积核只卷积第一层的第 345 个 feature map。在 LeNet 里这个组合是人为规定的，Bouvrie 的文章中提出这个组合系数也是可以学出来的。但是我的实验里人为规定和自动学习的效果好像差别不大，不知道更复杂的数据集上会怎么样。

（2）ReLU 的位置

如果网络的最后一层是 softmax 分类器的话似乎其前一层就不能用 ReLU，因为 ReLU 输出可能相差很大（比如 0 和几十），这时再经过 softmax 就会出现一个节点为

1 其它全 0 的情况。softmax 的 cost function 里包含一项 log(y)，如果 y 正好是 0 就没法算了。所以我在倒数第二层还是采用 sigmoid。

（3）其他高级玩法

本来还打算玩一些其他更有趣的东西 ,比如 dropout、maxout、max pooling 等等。但是时间有限，老板已经嫌我不务正业了。

三、总结

这次的收获：

1. 写复杂的算法不能急于求成，要循序渐进
2. 测试很重要。上 Andrew Ng 公开课时候觉得 gradient check 很烦，现在看来简直是神器
3. 机器越快越好……

转自：http://blog.csdn.net/huangbo10/article/details/24941079