卷积神经网络算法的一个实现

前言

从理解卷积神经到实现它，前后花了一个月时间，现在也还有一些地方没有理解透彻，CNN 还是有一定难度的，不是看哪个的博客和一两篇论文就明白了，主要还是靠自己去专研，阅读推荐列表在末尾的参考文献。目前实现的 CNN 在 MINIT 数据集上效果还不错，但是还有一些 bug，因为最近比较忙，先把之前做的总结一下，以后再继续优化。

卷积神经网络 CNN 是 Deep Learning 的一个重要算法，在很多应用上表现出卓越的效果，[1]中对比多重算法在文档字符识别的效果，结论是 CNN 优于其他所有的算法。CNN 在手写体识别取得最好的效果，[2]将 CNN 应用在基于人脸的性别识别，效果也非常不错。前段时间我用 BP 神经网络对手机拍照图片的数字进行识别，效果还算不错，接近 98%，但在汉字识别上表现不佳，于是想试试卷积神经网络。

1、CNN 的整体网络结构

卷积神经网络是在 BP 神经网络的改进，与 BP 类似，都采用了前向传播计算输出值，反向传播调整权重和偏置；CNN 与标准的 BP 最大的不同是：CNN 中相邻层之间的神经单元并不是全连接，而是部分连接，也就是某个神经单元的感知区域来自于上层的部分神经单元，而不是像 BP 那样与所有的神经单元相连接。CNN 的有三个重要的思想架构：

局部区域感知

权重共享

空间或时间上的采样

局部区域感知能够发现数据的一些局部特征，比如图片上的一个角，一段弧，这些基本特征是构成动物视觉的基础[3]；而 BP 中，所有的像素点是一堆混乱的点，相互之间的关系没有被挖掘。

CNN 中每一层的由多个 map 组成，每个 map 由多个神经单元组成，同一个 map 的所有神经单元共用一个卷积核（即权重），卷积核往往代表一个特征，比如某个卷积和代表一段弧，那么把这个卷积核在整个图片上滚一下，卷积值较大的区域就很有可能是一段弧。注意卷积核其实就是权重，我们并不需要单独去计算一个卷积，而是一个固定大小的权重矩阵去图像上匹配时，这个操作与卷积类似，因此我们称为卷积神经网络，实际上，BP 也可以看做一种特殊的卷积神经网络，只是这个卷积核就是某层的所有权重，即感知区域是整个图像。权重共享策略减少了需要训练的参数，使得训练出来的模型的泛华能力更强。

采样的目的主要是混淆特征的具体位置，因为某个特征找出来后，它的具体位置已经不重要了，我们只需要这个特征与其他的相对位置，比如一个“8”，当我们得到了上面一个"o"时，我们不需要知道它在图像的具体位置，只需要知道它下面又是一个“o”我们就可以知道是一个'8'了，因为图片中"8"在图片中偏左或者偏右都不影响我们认识它，这种混淆具体位置的策略能对变形和扭曲的图片进行识别。

CNN 的这三个特点是其对输入数据在空间（主要针对图像数据）上和时间（主要针对时间序列数据，参考 TDNN）上的扭曲有很强的鲁棒性。CNN 一般采用卷积层与

采样层交替设置，即一层卷积层接一层采样层，采样层后接一层卷积...这样卷积层提取出特征，再进行组合形成更抽象的特征，最后形成对图片对象的描述特征，CNN 后面还可以跟全连接层，全连接层跟 BP 一样。下面是一个卷积神经网络的示例：

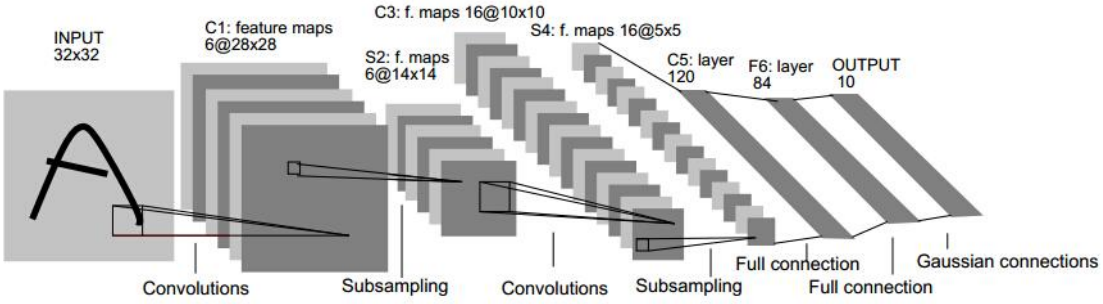


图 1（图片来源）

卷积神经网络的基本思想是这样，但具体实现有多重版本，我参考了 matlab 的 Deep Learning 的工具箱 DeepLearnToolbox，这里实现的 CNN 与其他最大的差别是采样层没有权重和偏置，仅仅只对卷积层进行一个采样过程，这个工具箱的测试数据集是 MINIST，每张图像是 28*28 大小，它实现的是下面这样一个 CNN：

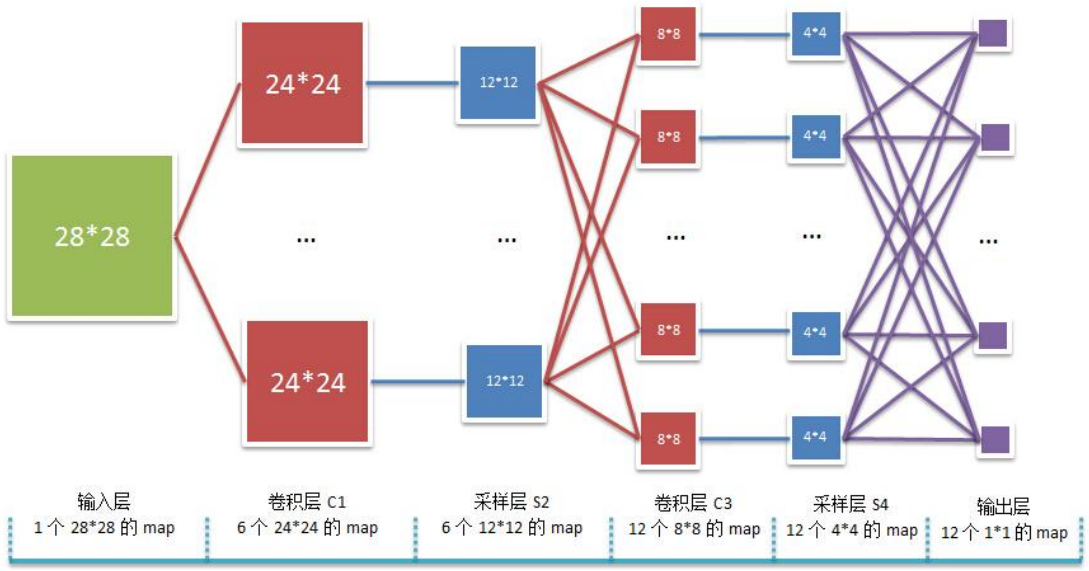


图 2

2、网络初始化

CNN 的初始化主要是初始化卷积层和输出层的卷积核（权重）和偏置，DeepLearnToolbox 里面对卷积核和权重进行随机初始化，而对偏置进行全 0 初始化。

3、前向传输计算

前向计算时，输入层、卷积层、采样层、输出层的计算方式不相同。

3.1 输入层：输入层没有输入值，只有一个输出向量，这个向量的大小就是图片的大小，即一个 28*28 矩阵;

3.2 卷积层：卷积层的输入要么来源于输入层，要么来源于采样层，如上图红色部分。卷积层的每一个 map 都有一个大小相同的卷积核，Toolbox 里面是 5*5 的卷积核。下面是一个示例，为了简单起见，卷积核大小为 2*2，上一层的特征 map 大小为 4*4，用这个卷积在图片上滚一遍，得到一个一个(4-2+1)*（4-2+1）=3*3 的特征 map，卷积核每次移动一步，因此。在 Toolbox 的实现中，卷积层的一个 map 与上层的所有 map 都关联，如上图的 S2 和 C3，即 C3 共有 6*12 个卷积核，卷积层的每一

个特征 map 是不同的卷积核在前一层所有 map 上作卷积并将对应元素累加后加一个偏置，再求 sigmod 得到的。还有需要注意的是，卷积层的 map 个数是在网络初始化指定的，而卷积层的 map 的大小是由卷积核和上一层输入 map 的大小决定的，假设上一层的 map 大小是 n*n、卷积核的大小是 k*k，则该层的 map 大小是(n-k+1)*(n-k+1)，比如上图的 24*24 的 map 大小 24=（28-5+1）。斯坦福的深度学习教程更加详细的介绍了卷积特征提取的计算过程。

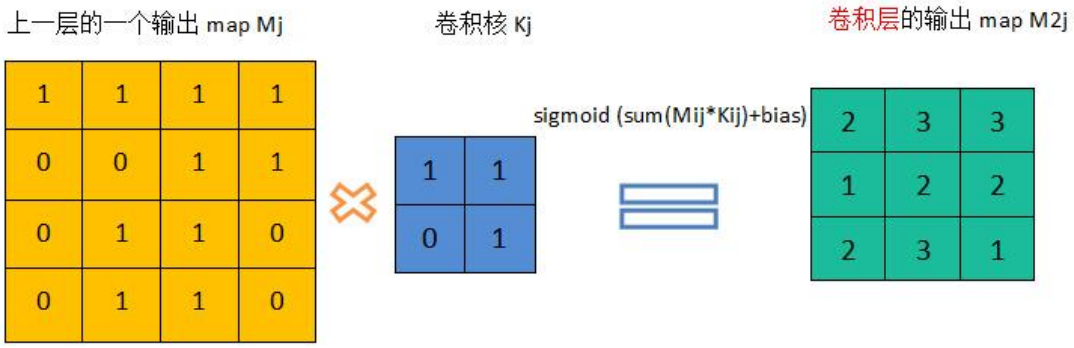


图 3

3.3 采样层（subsampling,Pooling）：采样层是对上一层 map 的一个采样处理，这里的采样方式是对上一层 map 的相邻小区域进行聚合统计，区域大小为 scale*scale，有些实现是取小区域的最大值，而 ToolBox 里面的实现是采用 2*2 小区域的均值。注意，卷积的计算窗口是有重叠的，而采用的计算窗口没有重叠，ToolBox 里面计算采样也是用卷积(conv2(A,K,'valid'))来实现的，卷积核是 2*2，每个元素都是 1/4，去掉计算得到的卷积结果中有重叠的部分，即：

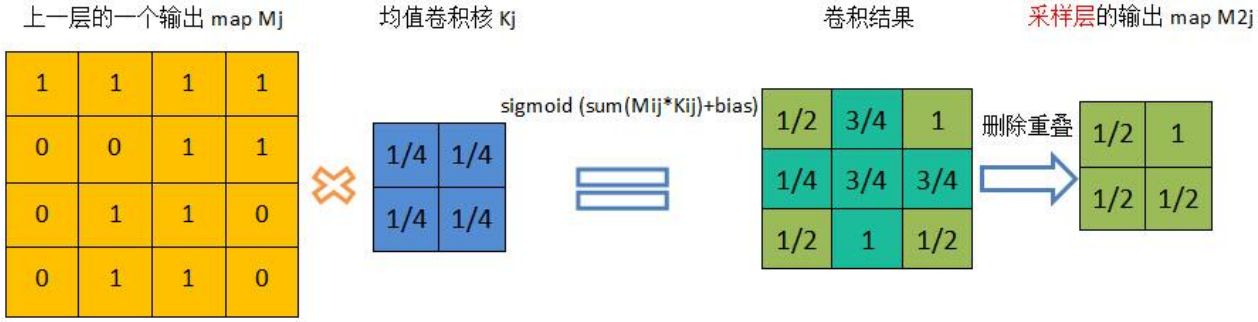


图 4

4、反向传输调整权重

反向传输过程是 CNN 最复杂的地方，虽然从宏观上来看基本思想跟 BP 一样，都是通过最小化残差来调整权重和偏置，但 CNN 的网络结构并不像 BP 那样单一，对不同的结构处理方式不一样，而且因为权重共享，使得计算残差变得很困难，很多论文[1][5]和文章[4]都进行了详细的讲述，但我发现还是有一些细节没有讲明白，特别是采样层的残差计算，我会在这里详细讲述。

4.1 输出层的残差

和 BP 一样，CNN 的输出层的残差与中间层的残差计算方式不同，输出层的残差是输出值与类标值得误差值，而中间各层的残差来源于下一层的残差的加权和。输出层的残差计算如下：

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

公式来源

这个公式不做解释，可以查看公式来源，看斯坦福的深度学习教程的解释。

4.2 下一层为采样层（subsampling）的卷积层的残差

当一个卷积层 L 的下一层(L+1)为采样层，并假设我们已经计算得到了采样层的残差，现在计算该卷积层的残差。从最上面的网络结构图我们知道，采样层（L+1）的 map 大小是卷积层 L 的 1/（scale*scale），ToolBox 里面，scale 取 2，但这两层的 map 个数是一样的，卷积层 L 的某个 map 中的 4 个单元与 L+1 层对应 map 的一个单元关联，可以对采样层的残差与一个 scale*scale 的全 1 矩阵进行克罗内克积进行扩充，使得采样层的残差的维度与上一层的输出 map 的维度一致，Toolbox 的代码如下，其中 d 表示残差，a 表示输出值：

```
net.layers{l}.d{j} = net.layers{l}.a{j} .* (1 - net.layers{l}.a{j}) .* expand(net.layers{l + 1}.d{j}, [net.layers{l + 1}.scale net.layers{l + 1}.scale 1])
```

扩展过程：

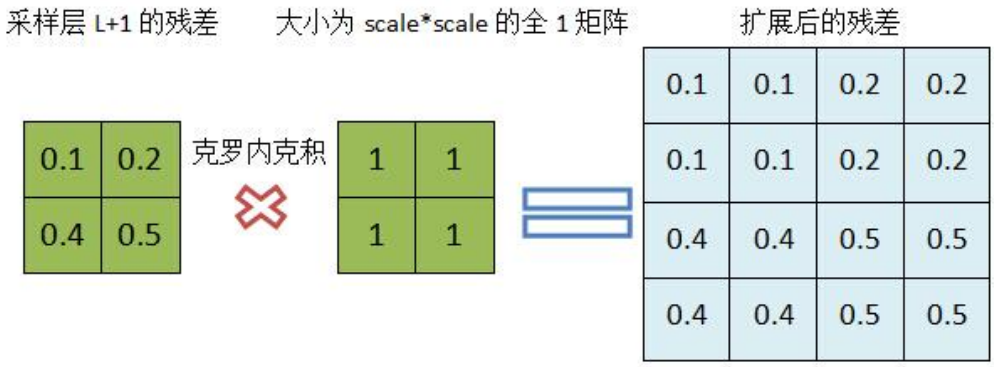


图 5

利用卷积计算卷积层的残差：

图 6

4.3 下一层为卷积层（subsampling）的采样层的残差

当某个采样层 L 的下一层是卷积层(L+1)，并假设我们已经计算出 L+1 层的残差，现在计算 L 层的残差。采样层到卷积层直接的连接是有权重和偏置参数的，因此不像卷积层到采样层那样简单。现再假设 L 层第 j 个 map Mj 与 L+1 层的 M2j 关联，按照 BP 的原理，L 层的残差 Dj 是 L+1 层残差 D2j 的加权和，但是这里的困难在于，我们很难理清 M2j 的那些单元通过哪些权重与 Mj 的哪些单元关联，Toolbox 里面还是采用

卷积（稍作变形）巧妙的解决了这个问题，其代码为：

```
convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'full');
```

rot180 表示对矩阵进行 180 度旋转（可通过行对称交换和列对称交换完成），为什么这里要对卷积核进行旋转，答案是：通过这个旋转，'full'模式下得卷积的正好抓住了前向传输计算上层 map 单元与卷积和及当期层 map 的关联关系，需要注意的是 matlab 的内置函数 convn 在计算卷积前，会对卷积核进行一次旋转，因此我们之前的所有卷积的计算都对卷积核进行了旋转：

```
a =
     1     1     1
     1     1     1
     1     1     1
k =
     1     2     3
     4     5     6
     7     8     9

>> convn(a,k,'full')
ans =

     1     3     6     5     3
     5    12    21    16     9
    12    27    45    33    18
    11    24    39    28    15
     7    15    24    17     9
```

convn 在计算前还会对待卷积矩阵进行 0 扩展，如果卷积核为 k*k，待卷积矩阵为 n*n，需要以 n*n 原矩阵为中心扩展到(n+2(k-1))*(n+2(k-1))，所有上面 convn(a,k,'full')的计算过程如下：

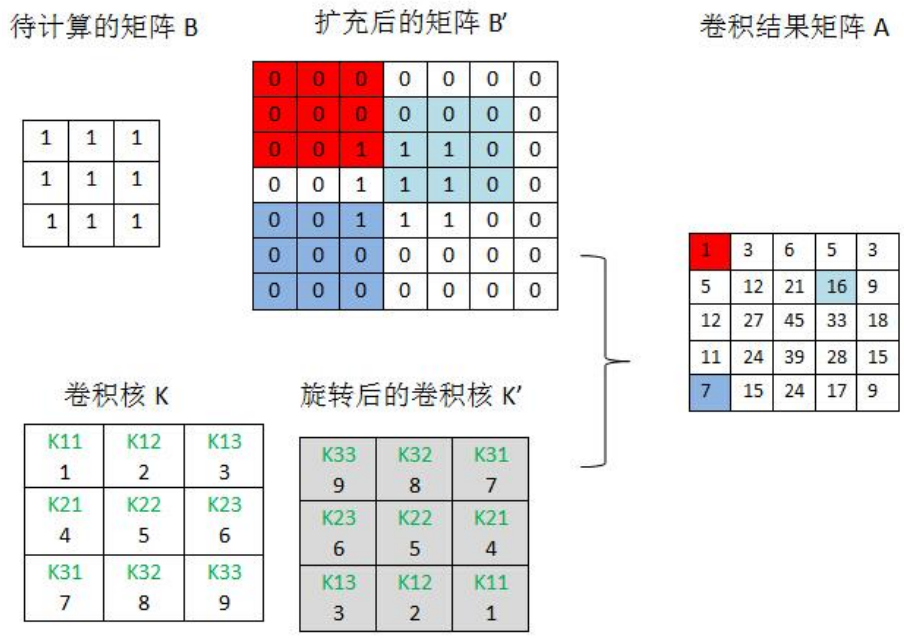


图 7

实际上 convn 内部是否旋转对网络训练没有影响，只要内部保持一致（即都要么旋转，要么都不旋转），所有我的卷积实现里面没有对卷积核旋转。如果在 convn 计算前，先对卷积核旋转 180 度，然后 convn 内部又对其旋转 180 度，相当于卷积核没有变。

为了描述清楚对卷积核旋转 180 与卷积层的残差的卷积所关联的权重与单元，正是前向计算所关联的权重与单元，我们选一个稍微大一点的卷积核，即假设卷积层采用用 3*3 的卷积核，其上一层采样层的输出 map 的大小是 5*5，那么前向传输由采样层得到卷积层的过程如下：

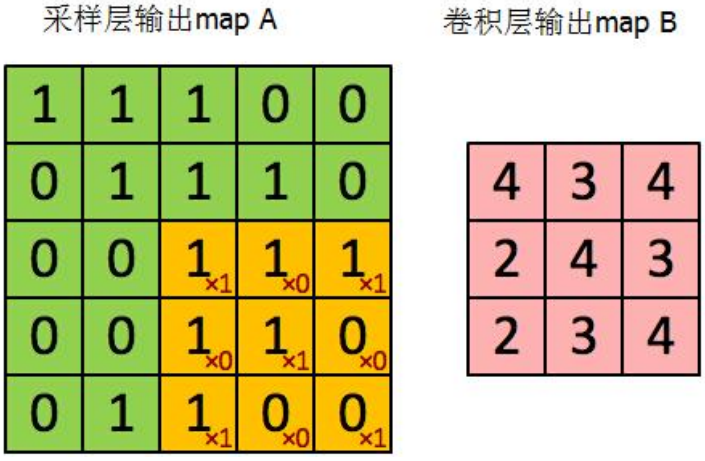


图 8

这里我们采用自己实现的 convn（即内部不会对卷积核旋转），并假定上面的矩阵 A、B 下标都从 1 开始，那么有：

B11 = A11*K11 + A12*K12 + A13*K13 + A21*K21 + A22*K22 + A23*K23 + A31*K31 + A32*K32 + A33*K33

B12 = A12*K11 + A13*K12 + A14*K13 + A22*K21 + A23*K22 + A24*K23 + A32*K31 + A33*K32 + A34*K33

B13 = A13*K11 + A14*K12 + A15*K13 + A23*K21 + A24*K22 + A25*K23 + A33*K31 + A34*K32 + A35*K33

B21 = A21*K11 + A22*K12 + A23*K13 + A31*K21 + A32*K22 + A33*K23 + A41*K31 + A42*K32 + A43*K33

B22 = A22*K11 + A23*K12 + A24*K13 + A32*K21 + A33*K22 + A34*K23 + A42*K31 + A43*K32 + A44*K33

B23 = A23*K11 + A24*K12 + A25*K13 + A33*K21 + A34*K22 + A35*K23 + A43*K31 + A44*K32 + A45*K33

B31 = A31*K11 + A32*K12 + A33*K13 + A41*K21 + A42*K22 + A43*K23 + A51*K31 + A52*K32 + A53*K33

B32 = A32*K11 + A33*K12 + A34*K13 + A42*K21 + A43*K22 + A44*K23 + A52*K31 + A53*K32 + A54*K33

B33 = A33*K11 + A34*K12 + A35*K13 + A43*K21 + A44*K22 + A45*K23 + A53*K31 + A54*K32 + A55*K33

我们可以得到 B 矩阵每个单元与哪些卷积核单元和哪些 A 矩阵的单元之间有关联：

A11 [K11] [B11]

A12 [K12, K11] [B12, B11]

A13 [K13, K12, K11] [B12, B13, B11]

A14 [K13, K12] [B12, B13]

A15 [K13] [B13]

A21 [K21, K11] [B21, B11]

A22 [K22, K21, K12, K11] [B12, B22, B21, B11]

A23 [K23, K22, K21, K13, K12, K11] [B23, B22, B21, B12, B13, B11]

A24 [K23, K22, K13, K12] [B23, B12, B13, B22]

A25 [K23, K13] [B23, B13]

A31 [K31, K21, K11] [B31, B21, B11]

A32 [K32, K31, K22, K21, K12, K11] [B31, B32, B22, B12, B21, B11]

A33 [K33, K32, K31, K23, K22, K21, K13, K12, K11] [B23, B22, B21, B31, B12, B13, B11, B33, B32]

A34 [K33, K32, K23, K22, K13, K12] [B23, B22, B32, B33, B12, B13]

A35 [K33, K23, K13] [B23, B13, B33]

A41 [K31, K21] [B31, B21]

A42 [K32, K31, K22, K21] [B32, B22, B21, B31]

A43 [K33, K32, K31, K23, K22, K21] [B31, B23, B22, B32, B33, B21]

A44 [K33, K32, K23, K22] [B23, B22, B32, B33]

A45 [K33, K23] [B23, B33]

A51 [K31] [B31]

A52 [K32, K31] [B31, B32]

A53 [K33, K32, K31] [B31, B32, B33]

A54 [K33, K32] [B32, B33]

A55 [K33] [B33]

然后再用 matlab 的 convn(内部会对卷积核进行 180 度旋转)进行一次

convn(B,K,'full')，结合图 7，看红色部分，除去 0，A11=B'33*K'33=B11*K11，发现 A11 正好与 K11、B11 关联对不对；我们再看一个 A24=B'34*K'21+B'35*K'22+B'44*K'31+B'45*K'32=B12*K23+B13*K22+B22*K13+B23*K12，发现参与 A24 计算的卷积核单元与 B 矩阵单元，正好是前向计算时关联的单元，所以我们可以通过旋转卷积核后进行卷积而得到采样层的残差。

残差计算出来后，剩下的就是用更新权重和偏置，这和 BP 是一样的，因此不再细究，有问题欢迎交流。

5、代码实现

详细的代码不再这里贴了，我依旧放在了 github，欢迎参考和指正。我又是在重造车轮了，没有使用任何第三方的库类，这里贴一下调用代码：

```
publicstaticvoid runCnn() {
//创建一个卷积神经网络
        LayerBuilder builder = new LayerBuilder();
        builder.addLayer(Layer.buildInputLayer(new Size(28, 28)));
        builder.addLayer(Layer.buildConvLayer(6, new Size(5, 5)));
        builder.addLayer(Layer.buildSampLayer(new Size(2, 2)));
        builder.addLayer(Layer.buildConvLayer(12, new Size(5, 5)));
        builder.addLayer(Layer.buildSampLayer(new Size(2, 2)));
        builder.addLayer(Layer.buildOutputLayer(10));
        CNN cnn = new CNN(builder, 50);

//导入数据集
        String fileName = "dataset/train.format";
        Dataset dataset = Dataset.load(fileName, ",", 784);
        cnn.train(dataset, 3);//
        String modelName = "model/model.cnn";
        cnn.saveModel(modelName);
        dataset.clear();
        dataset = null;

//预测
        // CNN cnn = CNN.loadModel(modelName);
        Dataset testset = Dataset.load("dataset/test.format", ",", -1);
        cnn.predict(testset, "dataset/test.predict");
    }
```

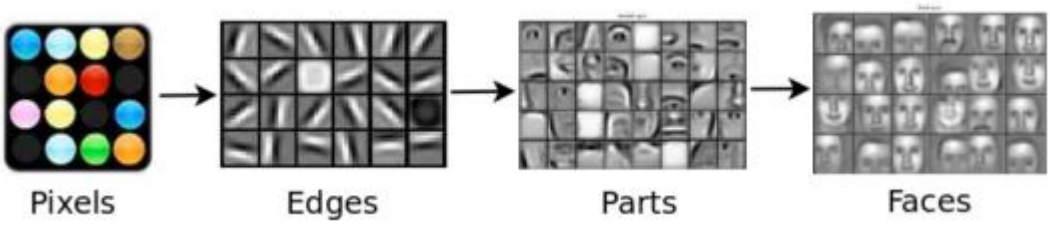
6、参考文献

[1].YANN LECUN. Gradient-Based Learning Applied to Document Recognition.
[2].Shan Sung LIEW. Gender classification: A convolutional neural network approach.

[3] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction teraction,and functional architecture in the cat’s visual cortex,”
[4] tornadomeet. http://www.cnblogs.com/tornadomeet/p/3468450.html.
[5] Jake Bouvrie. Notes on Convolutional Neural Networks.
[6] C++ 实现的详细介绍. http://www.codeproject.com/Articles/16650/Neural-Network-for-Recogniti on-of-Handwritten-Digi
[7] matlab DeepLearnToolbox https://github.com/rasmusbergpalm/DeepLearnToolbox

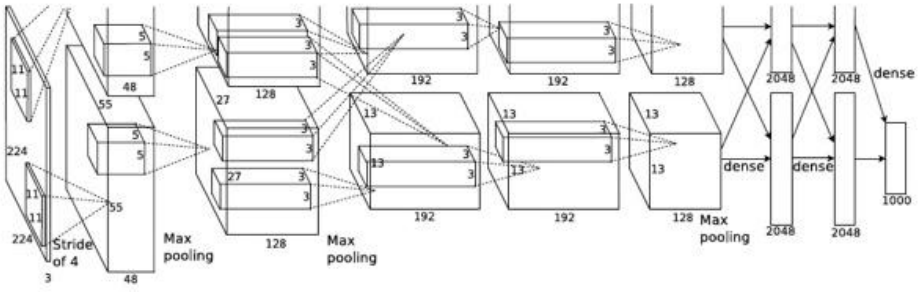
卷积神经网络

卷积神经网络(Convolutional Neural Networks：CNN)是人工神经网络(ANN)的一种，是深度学习的一种学习算法。它在图像识别和分类、自然语言处理广告系统中都有应用。
有意思是，卷积神经网络的提出其实就是来自于生物视觉模型。1981 年的诺贝尔医学奖，颁发给了 David Hubel、Torsten Wiesel。他们的工作给人们呈现了视觉系统是如何将来自外界的视觉信号传递到视皮层，并通过一系列处理过程（包括边界检测、运动检测、立体深度检测和颜色检测），最后在大脑中构建出一幅视觉图像。这个发现激发了人们对于神经系统的进一步思考，神经-中枢-大脑的工作过程，或许是一个不断迭代、不断抽象的过程。如下图所示，从原始信号摄入开始（瞳孔摄入像素 Pixels），首先进行初步处理（大脑皮层某些细胞发现边缘和方向），抽象（大脑判定眼前的物体的形状是圆形的），然后进一步抽象（大脑进一步判定该物体是只气球）。



六十年代的生理学的发现，促成了计算机人工智能在四十年后的突破性发展。1989 年，Yann LeCun （纽约大学教授，现 Facebook AI 研究室主任）提出了卷积神经网络，这是第一个真正成功训练多层网络结构的学习算法，但在当时的计算能力下效果欠佳。直到 2006 年，Geoffrey Hinton 提出基于深信度网（Deep Belief Net）和限制波尔兹曼机（RestrictedBoltzmann Machine）的学习算法，重新点燃了人工智能领域对于神经网络的热情。
卷积神经网络现在计算机视觉领域表现出众。ImageNet Classification with

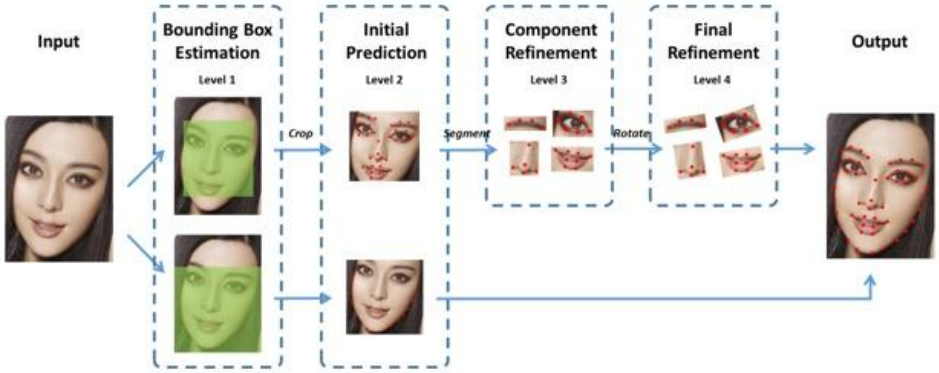
Deep Convolutional Neural Networks，这篇发表于 NIPS2012 的文章，是 Hinton 与其学生为了回应别人对于 Deep Learning 的质疑而将 CNN 结合 GPU 并行技术用于 Imagenet Challenge（图像识别目前最大的数据库，被誉为计算机视觉圣杯），最终取得了非常惊人的结果。他们的算法在 2012 年取得世界最好结果，使分类错误率从 26.2%下降到 16%。2013 年 Matthew Zeiler 的算法继续将错误率下降到 11.2%。Matthew Zeiler 还创立了 Clarifai，让我们可以有机会使用他们的图像识别算法对图片标注分类或图像搜索。



The convnet from Krizhevsky et al.'s NIPS 2012 ImageNet classification paper.

在月初 Kaggle 结束的 Galaxy Zoo challenge 中，参赛者要对星系图像进行分类(Spiral/Elliptical, Merging/Not merging, Clockwise/Anti-clockwise)，获胜的算法也是使用了卷积神经网络。Sander Dieleman 已放出了代码和详尽的文档： My solution for the Galaxy Zoo challenge

2014 年 3 月，Facebook 刚刚公布了他们在 CVPR2014 的文章：DeepFace: Closing the Gap to Human-Level Performance in Face Verification。他们用四百万人脸图片训练了一个九层的卷积神经网络来获得脸部特征，神经网络处理的参数高达 1.2 亿。他们在著名的公共测试数据集 LFW（labeled face in the wild，1:1 地判断是否两个照片是一个人）上达到了 97.25%的正确率。这个数字已经基本接近人眼的辨识水平。北京 Face++团队的算法达到 97.27%的正确率，在美图秀秀中就有应用，他们的主页提供了 API、demo 展示和论文。现在的最新进展是，香港中文大学基于 Fisher Discriminant Analysis 的算法将 Face Verification 正确率提高到 98.52%，超过了人类水平（97.53%）。



System overview in Face++ paper in ICCV2013

除了计算机视觉领域，卷积神经网络在人工智能和 robotics 也有很大潜力。

Hinton 的另外一个学生创立了人工智能公司 DeepMind，没有商业产品，只凭一篇论文，就已被 Google 招聘式收购。 他们使用深度学习（CNN）结合强化学习（Reinforcement Learning），在 Stella 模拟机上让机器自己玩了 7 个 Atari 2600 的游戏，结果不仅战胜了其他机器人，甚至在其中 3 个游戏中超越了人类游戏专家。很有意思，具体可以见 InfoQ 的看 DeepMind 如何用 Reinforcement learning 玩游戏，以及论文原文 Playing Atari with Deep Reinforcement Learning

从线性分类器到卷积神经网络

前言

本文大致分成两大部分，第一部分尝试将本文涉及的分类器统一到神经元类模型中，第二部分阐述卷积神经网络（CNN）的发展简述和目前的相关工作。

本文涉及的分类器（分类方法）有：

线性回归

逻辑回归（即神经元模型）

神经网络（NN）

支持向量机（SVM）

卷积神经网络（CNN）

从神经元的角度来看，上述分类器都可以看成神经元的一部分或者神经元组成的网络结构。

各分类器简述

逻辑回归

说逻辑回归之前需要简述一下线性回归。

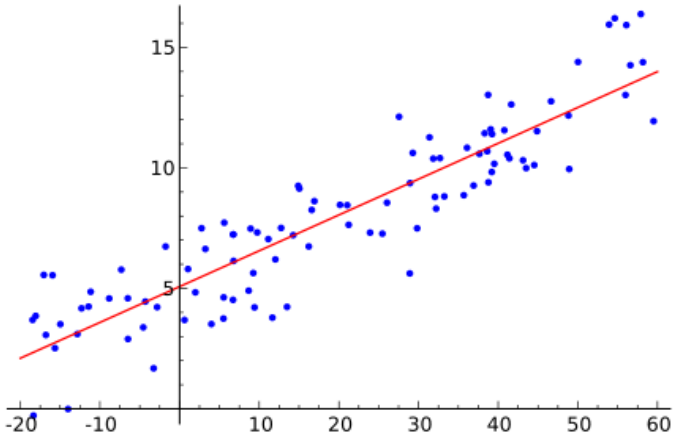


图 1 单变量的线性回归

图 1 中描述了一个单变量的线性回归模型：蓝点代表自变量 x 的分布——显然 x 呈现线性分布。于是我们可以用下面的式子对其进行拟合，即我们的目标函数可以写成:

$$\hat{y}=\omega x+b$$

从单变量到多变量模型，需要将 x 变成向量 $\vec{x}$，同时权重 ω 也需要变成向量 $\vec{\omega}$。

$$\hat{y}=\vec{\omega}\cdot\vec{x}+b$$

而一般线性回归的损失函数会用欧氏距离来进行衡量，即常说的最小二乘法，单个样本的损失函数可以写成：

$$\varepsilon=\frac{1}{2}(y-\hat{y})^2$$

而逻辑回归，可以简单理解成线性回归的结果加上了一个 sigmoid 函数。

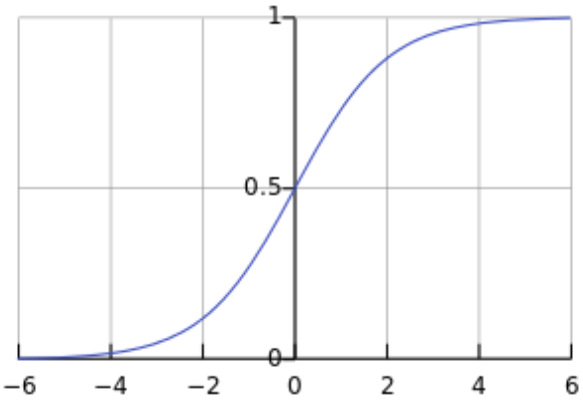


图 2 sigmoid 函数图像

从本质上来说，加上 sigmoid 函数的目的在于能够将函数输出的值域从$(-\infty,\infty)$映射到$(0,1)$之间，于是可以说逻辑回归的输出能够代表一个事件发生的**概率**。

逻辑分类的目标函数和单样本损失函数是：

$$\hat{y}=sigmoid(\vec{w}\cdot\vec{x}+b)$$

$$\varepsilon=-y\log(\hat{y})-(1-y)\log(1-\hat{y})$$

这里为何要使用一个复杂的损失函数这构造了一个**凸函数**，而如果直接使用最小二乘进行定义，损失函数会变成非凸函数。

实际上逻辑回归模型虽然名字带有**回归**，实际上一般用于二**分类**问题。即对 y^设置一个阈值（一般是 0.5），便实现了二分类问题。

而逻辑回归模型的另外一个名称为神经元模型——即我们认为大脑的神经元有着像上述模型一样的结构：一个神经元会根据与它相连的神经元的输入（x）做出反应，决定自身的激活程度（一般用 sigmoid 函数衡量激活程度），并将激活后的数值（y）输出到其他神经元。

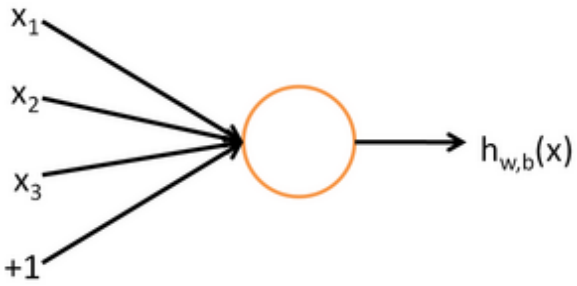


图 3 逻辑回归模型，即单个的神经元模型

神经网络（Neural Network，简称 NN）

逻辑回归的决策平面是线性的，所以，它一般只能够解决样本是**线性可分**的情况。

如果样本呈现非线性的时候，我们可以引入多项式回归。

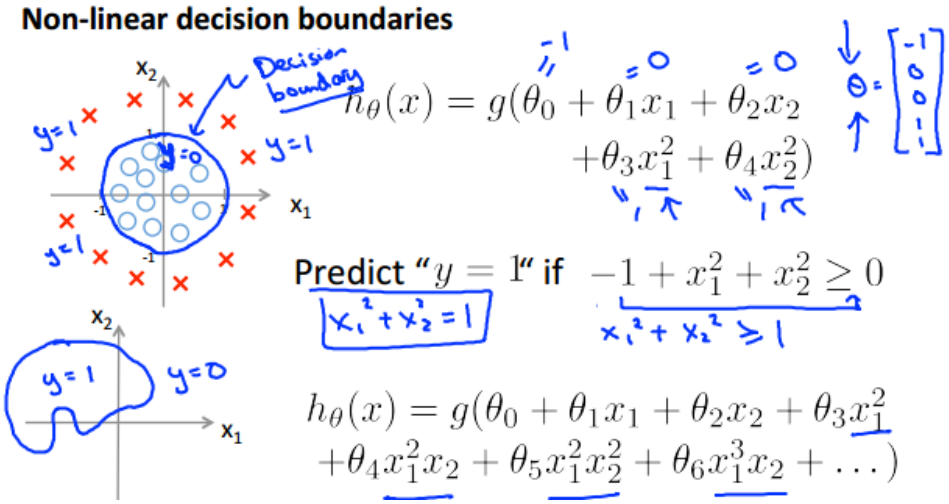


图 4 多项式回归解决样本线性不可分的情况，图片来自 Andrew Ng 的 Machine Learning 课程的课件

其实，多项式回归也可以看成是线性回归或者逻辑回归的一个特例——将线性回归或者逻辑回归的特征 x 转化为 x2,x3...等非线性的特征组合，然后对其进行线性的拟合。

多项式回归虽然能够解决非线性问题，但需要人工构造非线性的特征，那么，是否有一种模型，既能够应付样本非线性可分的情况，又同时能够自动构造非线性的特征呢？

答案是有的，这个模型就是神经网络。

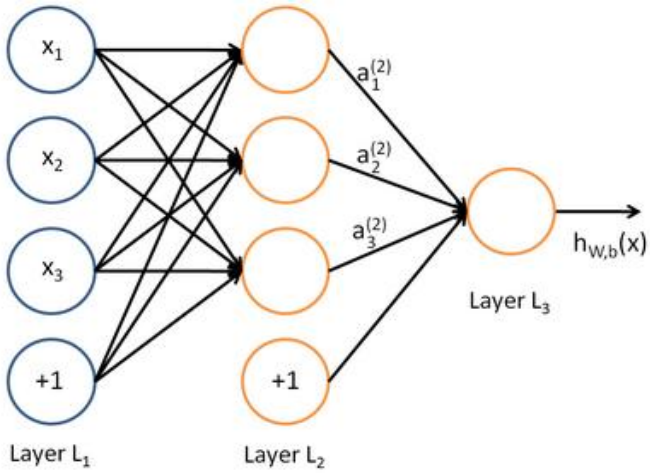


图 5 带一个隐层的神经网络模型

如图 5 所示，每个圆圈都是一个神经元（或者说是一个逻辑回归模型）。

所以神经网络可以看成“线性组合-非线性激活函数-线性组合-非线性激活函数…”这样的较为复杂网络结构，它的决策面是复杂的，于是能够适应样本非线性可分的情况。

另一方面，图 5 中中间一列的橙色神经元构成的层次我们成为隐层。我们认为**隐层的神经元对原始特征进行了组合，并提取出来了新的特征，而这个过程是模型在训练过程中自动“学习”出来的。**

神经网络的 fomulation 相对较为复杂，已经超出本文的范围，可参考 Standford 的深度学习教程

支持向量机（简称 SVM）

神经网络的出现一度引起研究热潮，但神经网络有它的缺点：

一般来说需要大量的训练样本

代价函数边界复杂，非凸，存在多个局部最优值。

参数意义模糊，比如隐层的神经元个数应该取多少一直没有定论。

浅层神经网络对于特征学习的表达能力有限。

深层神经网络的参数繁多，一方面容易导致过拟合问题，另一方面因为训练时间过长而导致不可学习。

于是，在上世纪 90 年代 SVM 被提出来后，神经网络一度衰落了。

那么 SVM 是什么？

SVM，更准确的说是 L-SVM，本质上依然是一个线性分类器，SVM 的核心思想在于它的分类准则——最大间隔（max margin）.

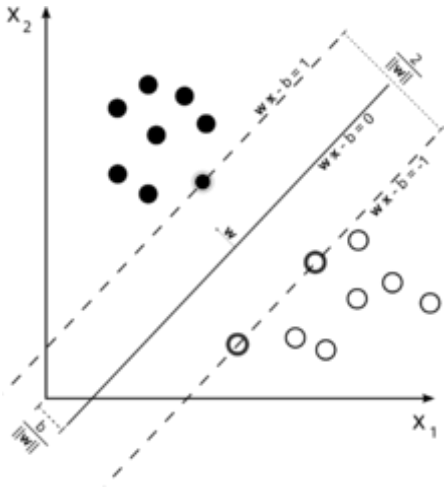


图 6 L-SVM 本质上是最大分类间隔的线性分类器

同为线性分类器的拓展，逻辑回归和 L-SVM 有着千丝万缕的关系，Andrew Ng 的课件有一张图很清晰地把这两者的代价函数联系起来了（见图 7）。

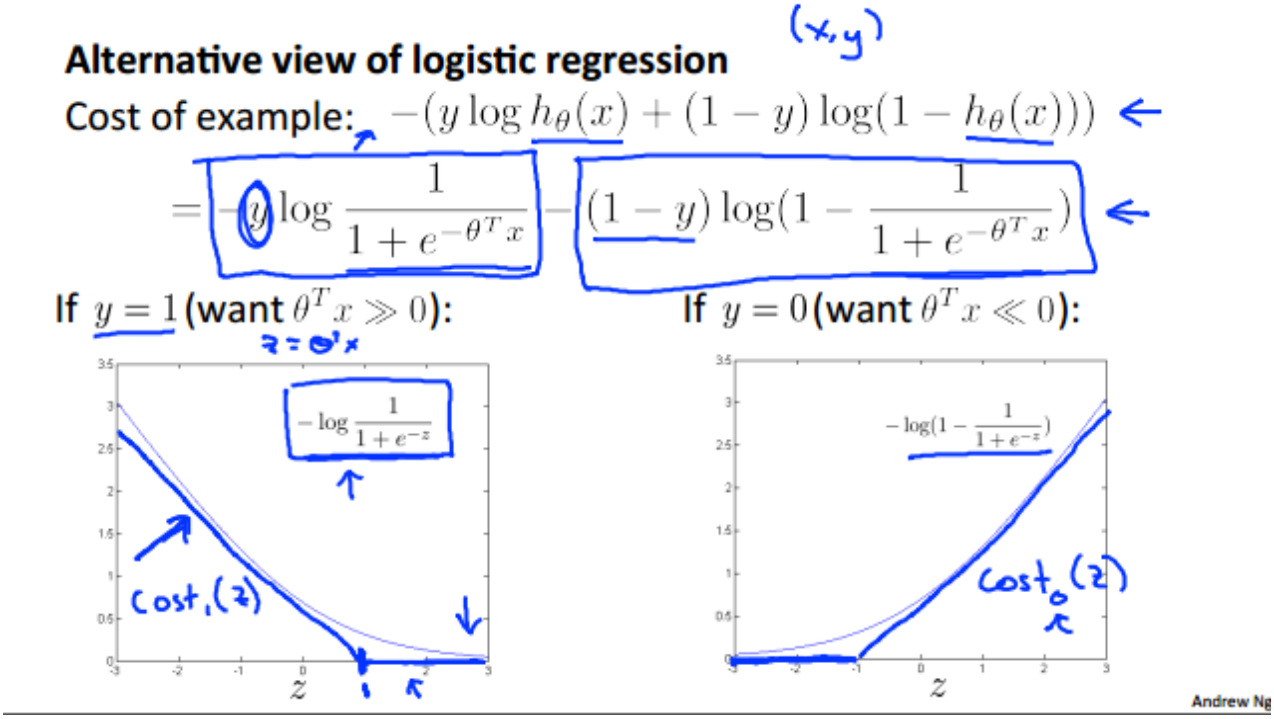


图 7 L-SVM 和逻辑回归的代价函数对比，SVM 的有一个明显的转折点

由于 L-SVM 是线性分类器，所以不能解决样本线性不可分的问题。于是后来人们引入了核函数的概念，于是得到了 K-SVM（K 是 Kernel 的意思）。从本质上讲，核函数是用于将原始特征映射到高维的特征空间中去，并认为在高为特征空间中能够实现线性可分。个人认为，这个跟多项式回归的

思想是类似的，只不过核函数涵括的范围更加广，以及形式上更加优雅，使得它能够避免维数灾难。

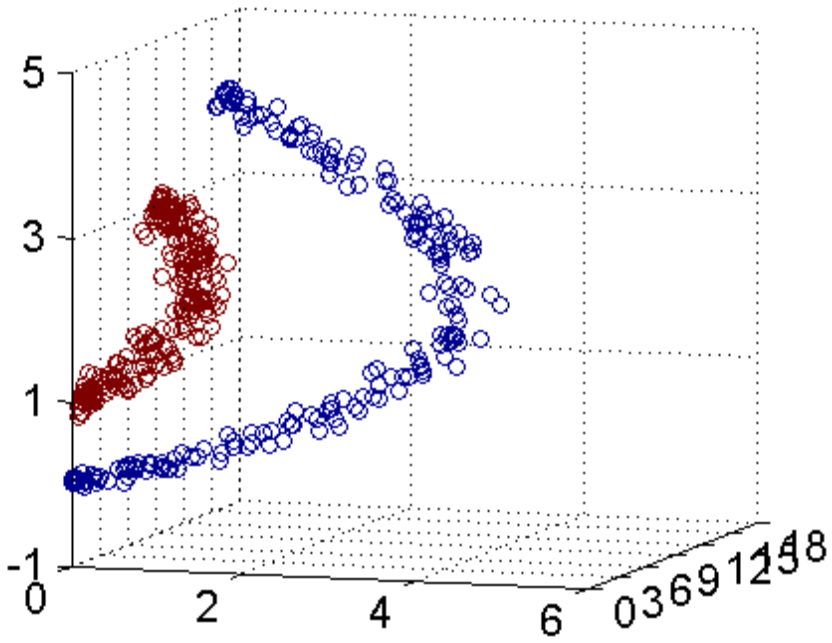


图 8 Kernel 能够对特征进行非线性映射(图片 from pluskid)

SVM 比起神经网络有着以下的优点：

代价函数是凸函数，存在全局最优值。

能够应付小样本集的情况

不容易过拟合，并且有着不错的泛化性能和鲁棒性

核函数的引入，解决了非线性问题，同时还避免了维数灾难

更多关于 SVM 的内容可以参考 July 的这篇文章：支持向量机通俗导论（理解 SVM 的三层境界）

然而，其实我们依然可以将 SVM 看成一种特殊的神经元模型：

L-SVM 本质上跟单神经元（即逻辑回归）模型的最大差别，只是代价函数的不同，所以可以将 SVM 也理解成一种神经元，只不过它的激活函数不是 sigmoid 函数，而是 SVM 独有的一种激活函数的定义方法。

K-SVM 只是比起 L-SVM 多了一个负责用于非线性变换的核函数，这个跟神经网络的隐层的思想也是一脉相承的。所以 K-SVM 实际上是两层的神经元网络结构：第一层负责非线性变换，第二层负责回归。

《基于核函数的 SVM 机与三层前向神经网络的关系》一文中，认为这两者从表达性来说是等价的。（注：这里的“三层前向神经网络”实际上是带一个隐层的神经网络，说是三层是因为它把网络的输入也看成一个层。）

卷积神经网络

近年来，神经网络又重新兴盛起来了。尤以“卷积神经网络”为其代表。于是本文下面的篇幅主要围绕卷积神经网络进行展开。

生物学基础

引自 Deep Learning（深度学习）学习笔记整理系列之（七）。

1962 年 Hubel 和 Wiesel 通过对猫视觉皮层细胞的研究，提出了感受野(receptive field)的概念，1984 年日本学者 Fukushima 基于感受野概念提出的神经认知机(neocognitron)可以看作是卷积神经网络的第一个实现网络，也是感受野概念在人工神经网络领域的首次应用。神经认知机将一个视觉模式分解成许多子模式（特征），然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统模型化，使其能够在即使物体有位移或轻微变形的时候，也能完成识别。

通常神经认知机包含两类神经元，即承担**特征抽取的 S-元**和**抗变形的 C-元**。S-元中涉及两个重要参数，即感受野与阈值参数，前者确定输入连接的数目，后者则控制对特征子模式的反应程度。许多学者一直致力于提高神经认知机的性能的研究：在传统的神经认知机中，每个 S-元的感光区中

由 C-元带来的视觉模糊量呈正态分布。如果感光区的边缘所产生的模糊效果要比中央来得大，S-元将会接受这种非正态模糊所导致的更大的变形容忍性。我们希望得到的是，训练模式与变形刺激模式在感受野的边缘与其中心所产生的效果之间的差异变得越来越大。为了有效地形成这种非正态模糊，Fukushima 提出了带双 C-元层的改进型神经认知机。

基本的网络结构

一个较为出名的 CNN 结构为 LeNet5，它的 demo 可以参看这个网站。图 9 是 LeNet 的结构示意图。

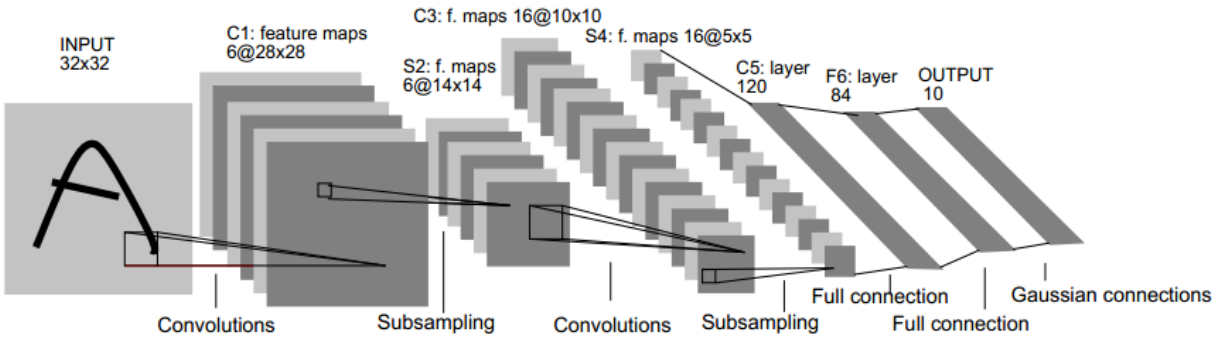


图 9 LeNet5 的网络结构示意图

图中的 Convolutions 对应了上一段说的 S-元，Subsampling 对应了上一段中说的 C-元。

对于 Convolution 层的每个神经元，它们的权值都是共享的，这样做的好处是大大减少了神经网络的参数个数。

对于 Sampling 层的每个神经元，它们是上一层 Convolution 层的局部范围的均值(或者最大值)，能够有效地提供局部的平移和旋转不变性。

为何神经网络重新兴起？

卷积神经网络属于一种深度的神经网络，如上文所说，深度神经网络在之

前是不可计算的，主要是由于网络层次变深后会导致下面问题：

由于网络参数增多，导致了严重的过拟合现象

在训练过程中，对深度网络使用 BP 算法传播时候梯度迅速减少，导致前面的网络得不到训练，网络难以收敛。

而这两个问题在目前都得到了较好的解决：

共享权值：即上文提到的卷积层的卷积核权值共享，大大减少了网络中参数的数量级。

加大数据量：一个是通过众包的方式来增加样本的量级，比如，目前 ImageNet 已经有了 120 万的带标注的图片数据。另一个是通过对已有的样本进行随机截取、局部扰动、小角度扭动等方法，来倍增已有的样本数。

改变激活函数：使用 ReLU)作为激活函数，由于 ReLU 的导数对于正数输入来说恒为 1，能够很好地将梯度传到位于前面的网络当中。

Dropout 机制：Hinton 在 2012 提出了 Dropout 机制，能够在训练过程中将通过随机禁止一半的神经元被修改，避免了过拟合的现象。

GPU 编程：使用 GPU 进行运算，比起 CPU 时代运算性能有了数量级的提升。

上述问题得到有效解决后，神经网络的优势就得到充分的显现了：

复杂模型带来的强大的表达能力

有监督的自动特征提取

于是神经网络能够得到重新兴起，也就可以解释了。下文会选取一些样例来说明神经网络的强大之处。

CNN 样例 1 AlexNet

在 ImageNet 举办的大规模图像识别比赛 ILSVRC2012 中分类比赛中，Hinton 的学生 Alex 搭建了一个 8 层的 CNN，最终 top-5 的漏报率是 16%，抛离而第二名的 27%整整有 11 个百分点。

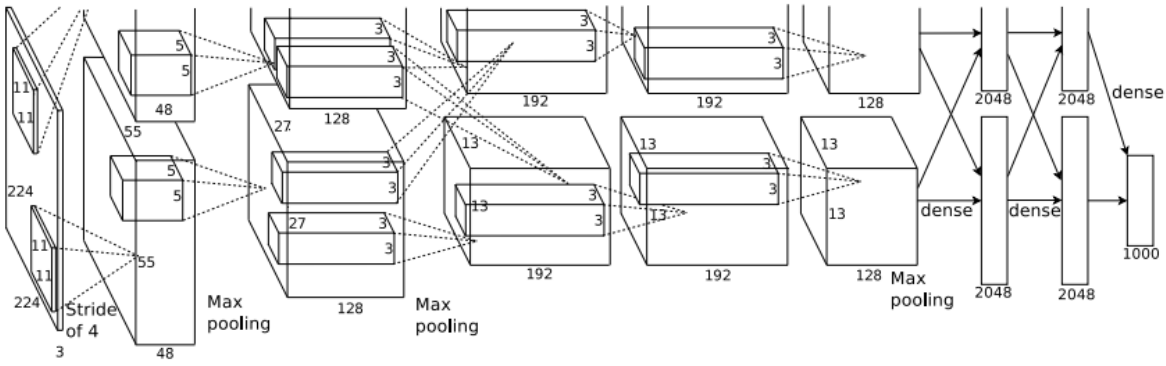


图 10 AlexNet 的 CNN 结构，包括 5 个卷积层，和 3 个全连接层，最后一个 softmax 分类器

这个网络中用到的技术有：

ReLU 激活函数

多 GPU 编程

局部正则化（Local Response Normalization）

重叠的下采样（Overlapping Pooling）

通过随机截取和 PCA 来增加数据

Dropout

CNN 样例 2 deconvnet

在下一年的比赛 ILSVRC2013 中，在同样的数据集同样的任务下，Matthew 进一步将漏报率降到了 11%。他使用了一个被命名为“Deconvolutional Network”（简称 deconvnet）的技术。

Matthew 的核心工作在于尝试将 CNN 学习出来的特征映射回原图片，来对每个卷积层最具有判别性的部分实现可视化——也就是，观察 CNN 在

卷积层中学习到了什么。

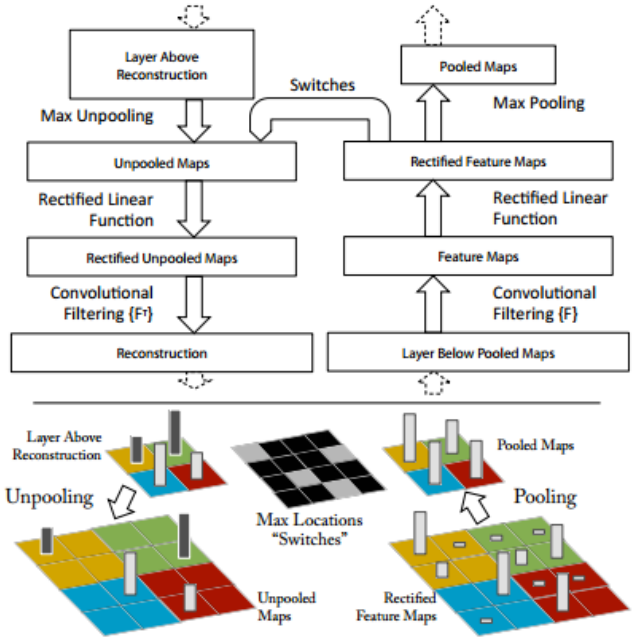


图 11 deconvnet 的思想是将网络的输出还原成输入

CNN 样例 3 DeepPose

DeepPose 的贡献在于它对 CNN 使用了级联的思想：首先，可以用第一层 CNN 大致定位出人物的关节位置，然后使用反复使用第二层神经网络对第一层网络进行微调，以达到精细定位的目的。

从另外一个角度，这个工作也说明了，CNN 不仅能够应付分类问题，也能够应付定位的问题。

图 12 DeepPose 通过级联地使用 CNN 来达到精细定位关节的位置

CNN 样例 4 CNN vs 人工特征

[CNN Features off-the-shelf: an Astounding Baseline for Recognition]

该工作旨在验证 CNN 提取出来的特征是否有效，于是作者做了这样的一个实验：将在 ILSVRC2013 的分类+定位比赛中获胜的 OverFeat 团队使用 CNN 提取出来的特征，加上一个 L-SVM 后构成了一个分类器，去跟各个物体分类的数据集上目前最好(state-of-the-art)的方法进行比较，结果几乎取得了全面的优胜。

总结

本文对数个分类器模型进行了介绍，并尝试统一到神经元网络模型的框架之中。

神经元网络模型的发展一直发生停滞，只是在中途有着不同的发展方向和重点。

神经元网络模型是个极其有效的模型，近年来由于样本数量和计算性能都得到了几何量级的提高，CNN 这一深度框架得以发挥它的优势，在计算机视觉的数个领域都取得了不菲的成就。

目前来说，对 CNN 本身的研究还不够深入，CNN 效果虽然优秀，但对于我们来说依然是一个黑盒子。弄清楚这个黑盒子的构造，从而更好地去改进它，会是一个相当重要的工作。

参考

Machine Learning@Coursera
UFLDL Tutorial
支持向量机通俗导论（理解 SVM 的三层境界）
支持向量机系列 from pluskid
基于核函数的 SVM 机与三层前向神经网络的关系,张铃,计算机学报，vol.25,No.7,July 2012
Deep Learning（深度学习）学习笔记整理系列之（七）

ImageNet Classification with Deep Convolutional Neural Networks,Alex Krizhevsky.etc,NIPS2012
Visualizing and Understanding Convolutional Networks,Matthew D. Zeiler,arXiv:1311.2901v3 [cs.CV] 28 Nov 2013
DeepPose: Human Pose Estimation via Deep Neural Networks,arXiv:1312.4659v1 [cs.CV] 17 Dec 2013
CNN Features off-the-shelf: an Astounding Baseline for Recognition,arXiv:1403.6382v3 [cs.CV] 12 May 2014

转载自: http://zhangliliang.com/2014/06/14/from-lr-to-cnn/

神经网络：卷积神经网络

一、前言

这篇卷积神经网络是前面介绍的多层神经网络的进一步深入，它将深度学习的思想引入到了神经网络当中，通过卷积运算来由浅入深的提取图像的不同层次的特征，而利用神经网络的训练过程让整个网络自动调节卷积核的参数，从而无监督的产生了最适合的分类特征。这个概括可能有点抽象，我尽量在下面描述细致一些，但如果要更深入了解整个过程的原理，需要去了解DeepLearning。

这篇文章会涉及到卷积的原理与图像特征提取的一般概念，并详细描述卷积神经网络的实现。但是由于精力有限，没有对人类视觉的分层以及机器学习等原理有进一步介绍，后面会在深度学习相关文章中展开描述。

二、卷积

卷积是分析数学中一种很重要的运算，其实是一个很简单的概念，但是很多做图像处理的人对这个概念都解释不清，为了简单起见，这里面我们只介绍离散形式的卷积，那么在图像上，对图像用一个卷积核进行卷积运算，实际上是一个滤波的过程。我们先看一下卷积的基本数学表示：

$$f(x,y)\circ w(x,y)=\sum_{s=-aa}\sum_{t=-bb}w(s,t)f(x-s,y-t)$$

其中 $I=f(x,y)$是一个图像，$f(x,y)$是图像 I 上面 x 行 y 列上点的灰度值。而 $w(x,y)$有太多名字叫了，滤波器、卷积核、响应函数等等，而 a 和 b 定义了卷积核即 $w(x,y)$的大小。

从上面的式子中，可以很明显的看到，卷积实际上是提供了一个权重模板，这个模板在图像上滑动，并将中心依次与图像中每一个像素对齐，然后对这个模板覆盖的所有像素进行加权，并将结果作为这个卷积核在图像上该点的响应。所以从整个卷积运算我们可以看到以下几点：

1）卷积是一种线性运算

2）卷积核的大小，定义了图像中任何一点参与运算的邻域的大小。

3）卷积核上的权值大小说明了对应的邻域点对最后结果的贡献能力，权重越大，贡献能力越大。

4）卷积核沿着图像所有像素移动并计算响应，会得到一个和原图像等大图像。

5）在处理边缘上点时，卷积核会覆盖到图像外层没有定义的点，这时候有几种方法设定这些没有定义的点，可以用内层像素镜像复制，也可以全设置为 0。

35	124	211
248	195	143
82	20	206

★

w1	w2	w3
w4	w5	w6
w7	w8	w9

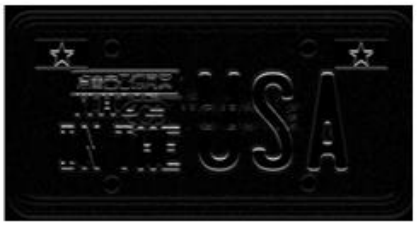
$$35 * w_1 + 124 * w_2 + 211 * w_3 + 248 * w_4 + 195 * w_5 + 143 * w_6 + 82 * w_7 + 20 * w_8 + 206 * w_9$$

三、卷积特征层

其实图像特征提取在博客里的一些其他文章都有提过，这里我想说一下图像特征提取与卷积的关系。其实大部分的图像特征提取都依赖于卷积运算，比如显著的边缘特征就是用各种梯度卷积算子对图像进行滤波的结果。一个图像里目标特征主要体现在像素与周围像素之间形成的关系，这些邻域像素关系形成了线条、角点、轮廓等。而卷积运算正是这种用邻域点按一定权重去重新定义该点值的运算。

水平梯度的卷积算子：

-1	-2	-1
0	0	0
1	2	1



竖直梯度的卷积算子：

-1	0	1
-2	0	2
-1	0	1

根据深度学习关于人的视觉分层的理论，人的视觉对目标的辨识是分层的，低层会提取一些边缘特征，然后高一些层次进行形状或目标的认知，更高层的会分析一些运动和行为。也就是说高层的特征是低层特征的组合，从低层到高层的特征表示越来越抽象，越来越能表现语义或者意图。而抽象层面越高，存在的可能猜测就越少，就越利于分类。

而深度学习就是通过这种分层的自动特征提取来达到目标分类，先构建一些基本的特征层，然后用这些基础特征去构建更高层的抽象，更精准的分类特征。

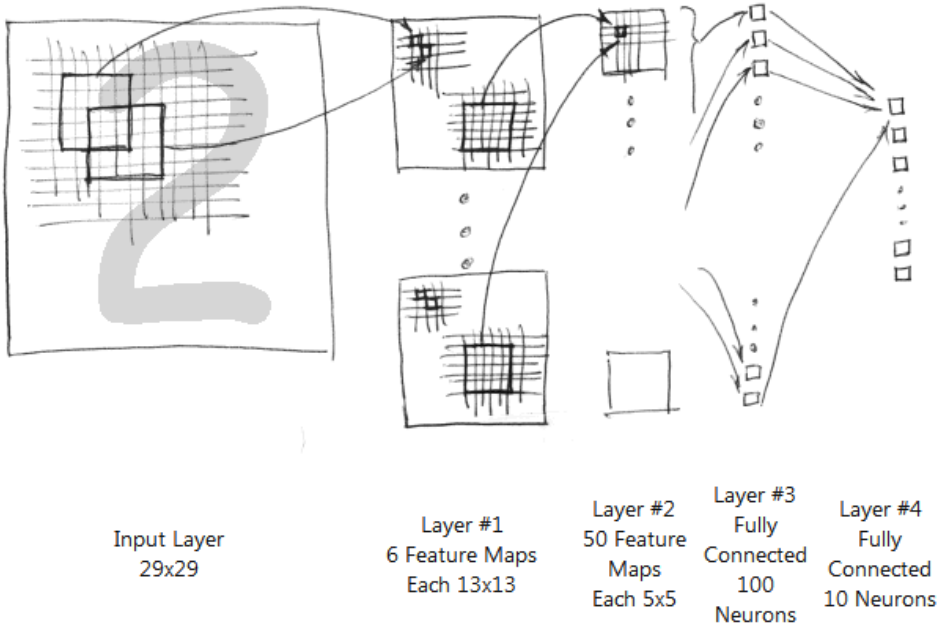
那整个卷积神经网络的结构也就很容易理解了，它在普通的多层神经网络的前面加了 2 层特征层，这两层特征层是通过权重可调整的卷积运算实现的。

四、卷积神经网络

在原来的多层神经网络结构中，每一层的所有结点会按照连续线的权重向前计算，成为下一层结点的输出。而每一条权重连结线都彼此不同，互不共享。每一个下一层结点的值与前一层所有结点都相关。

与普通多层神经网络不同的是，卷积神经网络里，有特征抽取层与降维层，这些层的结点连结是部分连接且，一幅特征图由一个卷积核生成，这一幅特征图上的所有结点共享这一组卷积核的参数。

这里我们设计一个 5 层的卷积神经网络，一个输入层，一个输出层，2 个特征提取层，一个全连接的隐藏层。下面详细说明每一层的设计思路。



在一般的介绍卷积神经网络的文章中你可能会看到在特征层之间还有 2 层降维层，在这里我们将卷积与降维同步进行，只用在卷积运算时，遍历图像中像素时按步长间隔遍历即可。

输入层：普通的多层神经网络，第一层就是特征向量。一般图像经过人为的特征挑选，通过特征函数计算得到特征向量，并作为神经网络的输入。而卷积神经网络的输出层则为整个图像，如上图示例 29*29,那么我们可以将图像按列展开，形成 841 个结点。而第一层的结点向前没有任何的连结线。

第 1 层： 第 1 层是特征层，它是由 6 个卷积模板与第一层的图像做卷积形成的 6 幅 13*13 的图像。也就是说这一层中我们算上一个偏置权重，一共有只有（5*5+1）*6=156 权重参数，但是，第二层的结点是将 6 张图像按列展开，即有 6*13*13=1014 个结点。
而第 2 层构建的真正难点在于，连结线的设定，当前层的结点并不是与前一层的所有结点相连，而是只与 25 邻域点连接，所以第二层一定有（25+1）*13*13*6=26364 条连线，但是这些连结线共享了 156 个权值。按前面多层网络 C++的设计中，每个连线对象有 2 个成员，一个是权重的索引，一个是上一层结点的索引，所以这里面要正确的设置好每个连线的索引值，这也是卷积神经网络与一般全连结层的区别。

第 2 层： 第 2 层也是特征层，它是由 50 个特征图像组成，每个特征图像是 5*5 的大小，这个特征图像上的每一点都是由前一层 6 张特征图像中每个图像取 25 个邻域点最后在一起加权而成，所以每个点也就是一个结点有（5*5+1）*6=156 个连结线，那么当前层一共有 5*5*50=1250 个结点,所以一共有 195000 个权重连结线,但是只有(5*5+1)*6*50=7800 个权重参数，每个权重连结线的值都可以在 7800 个权重参数数组中找到索引。
所以第 3 层的关键也是，如何建立好每根连结线的权重索引与与前一层连结的结点的索引。

第 3 层： 和普通多神经网络没有区别了，是一个隐藏层，有 100 个结点构成，每个结点与上一层中 1250 个结点相联，共有 125100 个权重与 125100 个连结线。

第 4 层： 输出层，它个结点个数与分类数目有关，假设这里我们设置为 10 类，则输出层为 10 个结点，相应的期望值的设置在多层神经网络里已经介绍过了，每个输出结点与上面隐藏层的 100 个结点相连，共有（100+1）*10=1010 条连结线，1010 个权重。

从上面可以看出，卷积神经网络的核心在于卷积层的创建，所以在设计 CNNLayer 类的时候，需要两种创建网络层的成员函数，一个用于创建普通的全连接层，一个用于创建卷积层。

```
1class CNNlayer
2 {
3private:
4        CNNlayer* preLayer;
5        vector<CNNneural> m_neurals;
6        vector<double> m_weights;
7public:
8        CNNlayer(){ preLayer = nullptr; }
9// 创建卷积层
10void createConvLayer(unsigned curNumberOfNeurals, unsigned preNumberOfNeurals, unsigned preNumberOfFeatMaps, unsigned curNumberOfFeatMaps);
11// 创建普通层
12void createLayer(unsigned curNumberOfNeurals,unsigned preNumberOfNeurals);
13void backPropagate(vector<double>& dErrWrtDxn, vector<double>& dErrWrtDxnm, double eta);
14 };
```

创建普通层，在前面介绍的多层神经网络中已经给出过代码，它接收两个参数，一个是前面一层结点数，一个是当前层结点数。

而卷积层的创建则复杂的多，所有连结线的索引值的确定需要对整个网络有较清楚的了解。这里设计的 createConvLayer 函数，它接收 4 个参数，分别对应，当前层结点数，前一层结点数，前一层特征图的个数和当前层特征图像的个数。

下面是 C++代码，要理解这一部分可以会稍有点难度，因为特征图实际中都被按列展开了，所以邻域这个概念会比较抽象，我们考虑把特征图像还原，从图像的角度去考虑。

```
1void CNNlayer::createConvLayer
2    (unsigned curNumberOfNeurals, unsigned preNumberOfNeurals, unsigned preNumberOfFeatMaps, unsigned curNumberOfFeatMaps)
3 {
4// 前一层和当前层特征图的结点数
5        unsigned preImgSize = preNumberOfNeurals / preNumberOfFeatMaps;
6        unsigned curImgSize = curNumberOfNeurals / curNumberOfFeatMaps;
7
8// 初始化权重
9        unsigned numberOfWeights = preNumberOfFeatMaps*curNumberOfFeatMaps*(5 * 5 + 1);
10for (unsigned i = 0; i != numberOfWeights; i++)
11       {
12               m_weights.push_back(0.05*rand() / RAND_MAX);
```

```
13        }
14// 建立所有连结线
15
16for (unsigned i = 0; i != curNumberOfFeatMaps; i++)
17        {
18                    unsigned imgRow = sqrt(preImgSize);    // 上一层特征图像的大小 ,imgRow=imgCol
19//  间隙 2 进行取样，邻域周围 25 个点
20for (int c = 2; c < imgRow-2; c= c+ 2)
21            {
22for (int r = 2; r < imgRow-2; r =r + 2)
23                {
24                    CNNneural neural;
25for (unsigned k = 0; k != preNumberOfNeurals; k++)
26                    {
27for (int kk = 0; kk < (5*5+1); kk ++)
28                        {
29                            CNNconnection connection;
30// 权重的索引
31                            connection.weightIdx = i*(curNumberOfFeatMaps*(5 * 5 + 1)) + k*(5 * 5 + 1) + kk;
32// 结点的索引
33                            connection.neuralIdx = k*preImgSize + c*imgRow + r;
34                            neural.m_connections.push_back(connection);
35                        }
36                        m_neurals.push_back(neural);
37                    }
38                }
39            }
40        }
41 }
```

五、训练与识别

整个网络结构搭建好以后，训练只用按照多层神经网络那样训练即可，其中的权值更新策略都是一致的。所以总体来说，卷积神经网络与普通的多层神经网络，就是结构上不同。卷积神经网络多了特征提取层与降维层，他们之间结点的连结方式是部分连结，多个连结线共享权重。而多层神经网络前后两层之间结点是全连结。除了这以外，权值更新、训练、识别都是一致的。

训练得到一组权值，也就是说在这组权值下网络可以更好的提取图像特征用于分类识别。

关于源码的问题：个人非常不推荐直接用别人的源码，所以我的博客里所有文章不会给出整个工程的源码，但是会给出一些核心函数的代码，如果你仔细阅读文章，一定能够很好的理解算法的核心思想。尝试着去自己实现，会对你的理解更有帮助。有什么疑问可以直接在下面留言。

转载自: http://www.cnblogs.com/ronny/p/ann_03.html

卷积神经网络（CNN）

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。

卷积网络最初是受视觉神经机制的启发而设计的，是为识别二维形状而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。1962 年 Hubel 和 Wiesel 通过对猫视觉皮层细胞的研究，提出了感受野(receptive field)的概念，1984 年日本学者 Fukushima 基于感受野概念提出的神经认知机(neocognitron)模型，它可以看作是卷积神经网络的第一个实现网络，也是感受野概念在人工神经网络领域的首次应用。

神经认知机将一个视觉模式分解成许多子模式(特征)，然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统模型化，使其能够在即使物体有 位移或轻微变形的时候，也能完成识别。神经认知机能够利用位移恒定能力从激励模式中学习，并且可识别这些模式的变化形。在其后的应用研究中，Fukushima 将神经认知机主要用于手写数字的识别。随后，国内外的研究人员提出多种卷积神经网络形式，在邮政编码识别（Y. LeCun etc）、车牌识别和人脸识别等方面 得到了广泛的应用。

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。

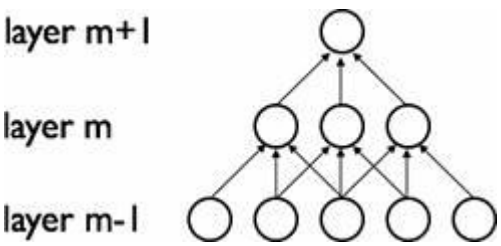
2 特征映射。网络的每一个计算层都是由**多个特征映射组**成的，每个特征映射都是平面形式的。平面中单独的神经元在约束下**共享 相同的突触权值**集，这种结构形式具有如下的有益效果：a.平移不变性。b.自由参数数量的缩减(通过权值共享实现)。

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。

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。

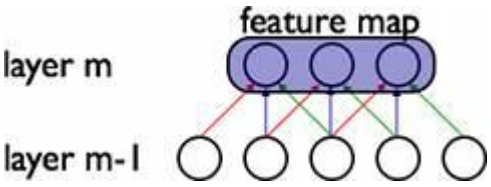


可以看到 m+1 层的神经元相对于第 m 层的接受域的宽度也为 3 ,但相对于输入层的接受域为 5，这种结构将学习到的过滤器（对应于输入信号中被最大激活的单元）限制在局部空间

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。



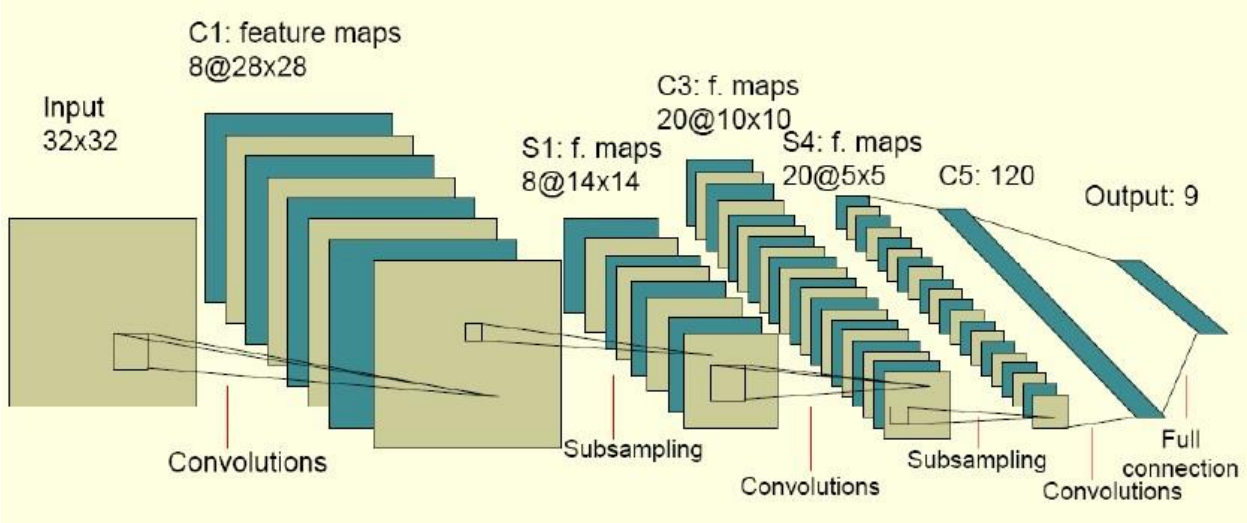
在图中，有 3 个隐层单元，他们属于同一个特征映射。同种颜色的链接的权值是相同的，我们仍然可以使用梯度下降的方法来学习这些权值，只需要对原始算法做一些小的改动，这里共享权值的梯度是所有共享参数的梯度的总和。我们不禁会问为什么要权重共享呢？一方面，重复单元能够对特征进行识别，而不考虑它在可视域中的位置。另一方面，权值 共享使得我们能更有效的进行特征抽取，因为它极大的减少了需要学习的自由变量的个数。通过控制模型的规模，卷积网络对视觉问题可以具有很好的泛化能力。

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，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，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(这一句表示没看懂，那位如果看懂了，请给我讲解一下)。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



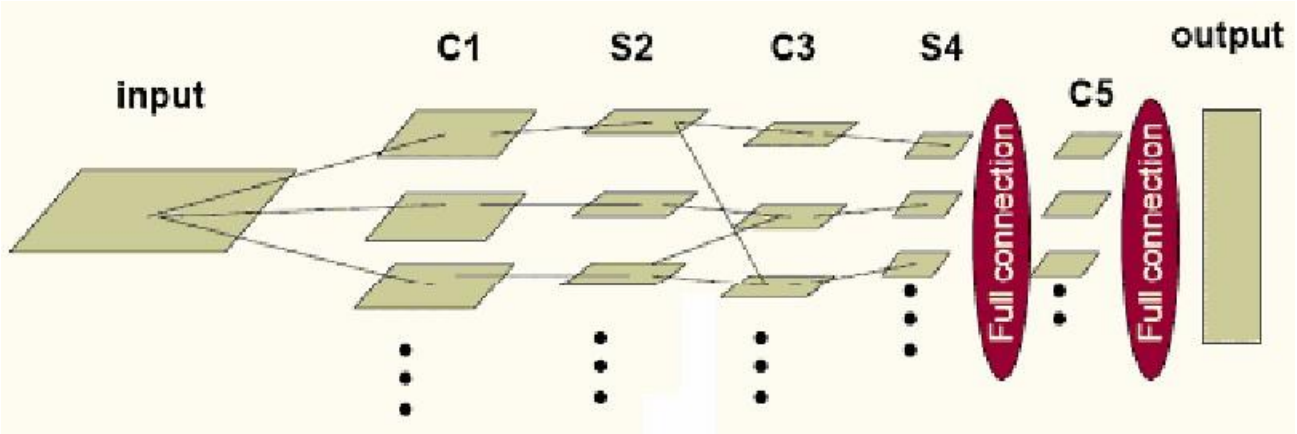
图中的卷积网络工作流程如下，输入层由 32×32 个感知节点组成，接收原始图像。然后，计算流程在卷积和子抽样之间交替进行，如下所 述：第一隐藏层进行卷积，它由 8 个特征映射组成，每个特征映射由 28×28 个神经元组成，每个神经元指定一个 5×5 的接受域；第二隐藏层实现子 抽样和局部平均，它同样由 8 个特征映射组成，但其每个特征映射由 14×14 个神经元组成。每个神经元具有一个 2×2 的接受域，一个可训练 系数，一个可训练偏置和一个 sigmoid 激活函数。可训练系数和偏置控制神经元的操作点。第三隐藏层进

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。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 的学习

总体而言，前面提到的卷积网络可以简化为下图所示模型：

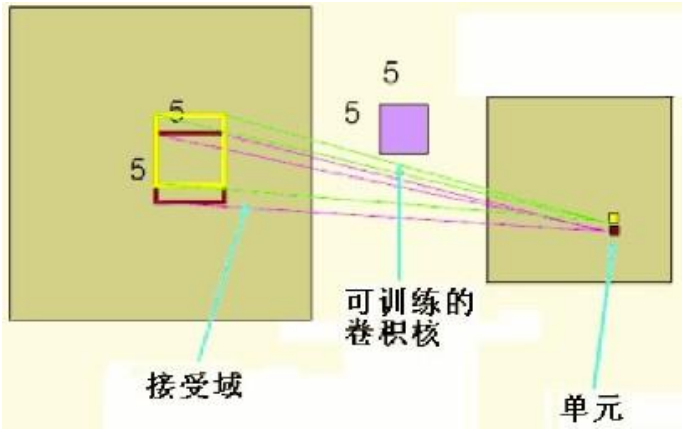


其中，input 到 C1、S4 到 C5、C5 到 output 是全连接，C1 到 S2、C3 到 S4 是一一对应的连接 ,S2 到 C3 为了消除网络对称性 ,去掉了一部分连接， 可以让特征映射更具多样性。

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。

3.1 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：

```
ConvolutionLayer::fprop(input,output) {
  //取得卷积核的个数
  int n=kernel.GetDim(0);
  for (int i=0;i<n;i++) {
      //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
      //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
      int a=table[i][0], b=table[i][1];
      //用第 i 个卷积核和输入层第 a 个特征映射做卷积
      convolution = Conv(input[a],kernel[i]);
      //把卷积结果求和
      sum[b] +=convolution;
  }
  for (i=0;i<(int)bias.size();i++) {
      //加上偏移量
      sum[i] += bias[i];
  }
```

```
    //调用 Sigmoid 函数
    output = Sigmoid(sum);
}
```

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。

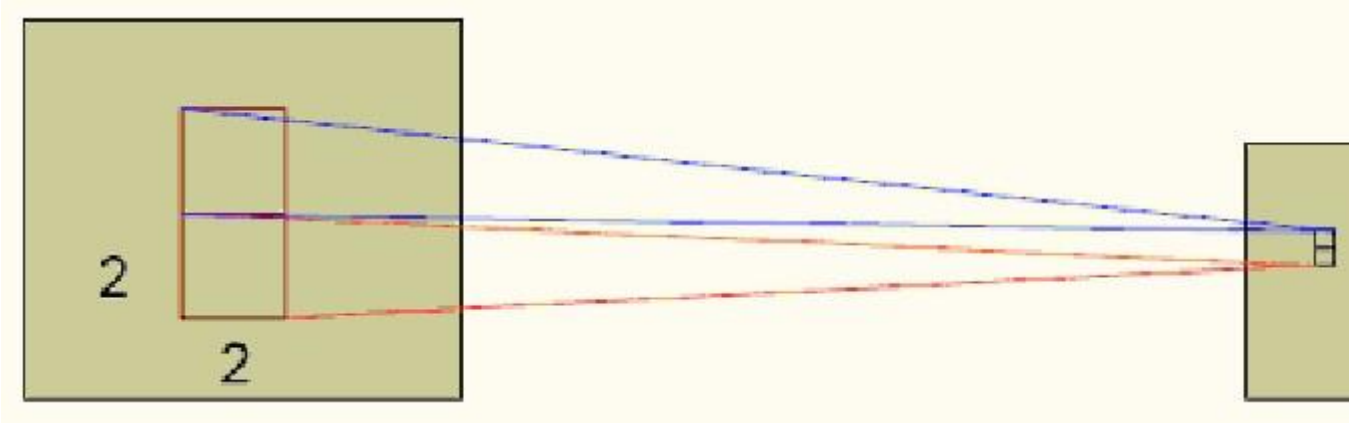
卷积层的反馈运算的核心代码如下：

```
ConvolutionLayer::bprop(input,output,in_dx,out_dx) {
    //梯度通过 DSigmoid 反传
    sum_dx = DSigmoid(out_dx);
    //计算 bias 的梯度
    for (i=0;i<bias.size();i++)    {
            bias_dx[i] = sum_dx[i];
    }
    //取得卷积核的个数
    int n=kernel.GetDim(0);
    for (int i=0;i<n;i++)
    {
            int a=table[i][0],b=table[i][1];
            //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
            input_dx[a] += DConv(sum_dx[b],kernel[i]);
            //用同样的方法计算卷积模板的梯度
            kernel_dx[i] += DConv(sum_dx[b],input[a]);
    }
}
```

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

3.2 子采样层的学习

子采样层的典型结构如下图所示。



类似的字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：

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

子采样层的反馈运算的核心代码如下：

```
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<n1;i++) {
```

```
        coeff_dx[i] = 0;
        bias_dx[i] = 0;
        for (j=0;j<n2/sw;j++)
            for (k=0;k<n3/sh;k++) {
                coeff_dx[i] += sub[j][k]*sum_dx[i][j][k];
                bias_dx[i] += sum_dx[i][j][k]);
            }
    }
    for (i=0;i<n1;i++) {
        for (j=0;j<n2;j++)
            for (k=0;k<n3;k++) {
                in_dx[i][j][k] = coeff[i]*sum_dx[i][j/sw][k/sh];
            }
    }
}
```

3.3 全连接层的学习

全连接层的学习与传统的神经网络的学习方法类似，也是使用 BP 算法，这里就不详述了。

关于 CNN 的完整代码可以参考 https://github.com/ibillxia/DeepLearnToolbox/tree/master/CNN 中的 Matlab 代码。

References

[1] Learn Deep Architectures for AI, Chapter 4.5.

[2] Deep Learning Tutorial, Release 0.1, Chapter 6.

[3] Convolutional Networks for Images Speech and Time-Series.

[4] 基于卷积网络的三维模型特征提取. 王添翼.

[5] 卷积神经网络的研究及其在车牌识别系统中的应用. 陆璐.

Original

Link: http://ibillxia.github.io/blog/2013/04/06/Convolutional-Neural-Networks/

Posted by Bill Xia Apr 6th, 2013 Posted in PRML Tagged with CNN, LeNet, NeuralNetworks

转载自: http://ibillxia.github.io/blog/2013/04/06/Convolutional-Neural-Networks/

Deep Learning（深度学习）学习笔记整理系列之（七）

Deep Learning（深度学习）学习笔记整理系列

zouxy09@qq.com
http://blog.csdn.net/zouxy09
作者：**Zouxy**

version 1.0 2013-04-08

声明：

1）该 Deep Learning 的学习系列是整理自网上很大牛和机器学习专家所无私奉献的资料的。具体引用的资料请看参考文献。具体的版本声明也参考原文献。

2）本文仅供学术交流，非商用。所以每一部分具体的参考资料并没有详细对应。如果某部分不小心侵犯了大家的利益，还望海涵，并联系博主删除。

3）本人才疏学浅，整理总结的时候难免出错，还望各位前辈不吝指正，谢谢。

4）阅读本文需要机器学习、计算机视觉、神经网络等等基础（如果没有也没关系了，没有就看看，能不能看懂，呵呵）。

5）此属于第一版本，若有错误，还需继续修正与增删。还望大家多多

指点。大家都共享一点点，一起为祖国科研的推进添砖加瓦（呵呵，好高尚的目标啊）。请联系：zouxy09@qq.com

目录：

接上

9.5、Convolutional Neural Networks 卷积神经网络

卷积神经网络是人工神经网络的一种，已成为当前语音分析和图像识别领域的研究热点。它的权值共享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，减少了权值的数量。该优点在网络的输入是多维图像时表现的更为明显，使图像可以直接作为网络的输入，避免了传统识别算法中复杂的特征提取和数据重建过程。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。

CNNs 是受早期的延时神经网络（TDNN）的影响。延时神经网络通过在时间维度上共享权值降低学习复杂度，适用于语音和时间序列信号的处理。

CNNs 是第一个真正成功训练多层网络结构的学习算法。它利用空间关系减少需要学习的参数数目以提高一般前向 BP 算法的训练性能。CNNs 作为一个深度学习架构提出是为了最小化数据的预处理要求。在 CNN 中，图像的一小部分（局部感受区域）作为层级结构的最低层的输入，信息再依次传输到不同的层，每层通过一个数字滤波器去获得观测数据的最显著的特征。这个方法能够获取对平移、缩放和旋转不变的

观测数据的显著特征，因为图像的局部感受区域允许神经元或者处理单元可以访问到最基础的特征，例如定向边缘或者角点。

1）卷积神经网络的历史

1962 年 Hubel 和 Wiesel 通过对猫视觉皮层细胞的研究，提出了感受野(receptive field)的概念，1984 年日本学者 Fukushima 基于感受野概念提出的神经认知机(neocognitron)可以看作是卷积神经网络的第一个实现网络，也是感受野概念在人工神经网络领域的首次应用。神经认知机将一个视觉模式分解成许多子模式（特征），然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统模型化，使其能够在即使物体有位移或轻微变形的时候，也能完成识别。

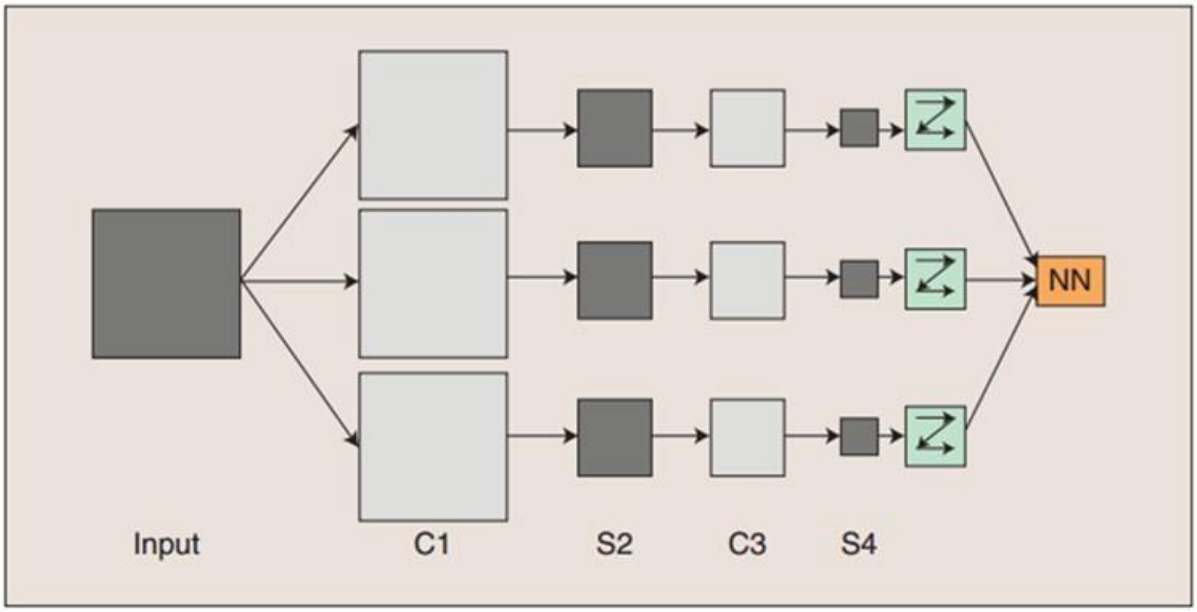
通常神经认知机包含两类神经元，即承担特征抽取的 S-元和抗变形的 C-元。S-元中涉及两个重要参数，即感受野与阈值参数，前者确定输入连接的数目，后者则控制对特征子模式的反应程度。许多学者一直致力于提高神经认知机的性能的研究：在传统的神经认知机中，每个 S-元的感光区中由 C-元带来的视觉模糊量呈正态分布。如果感光区的边缘所产生的模糊效果要比中央来得大，S-元将会接受这种非正态模糊所导致的更大的变形容忍性。我们希望得到的是，训练模式与变形刺激模式在感受野的边缘与其中心所产生的效果之间的差异变得越来越大。为了有效地形成这种非正态模糊，Fukushima 提出了带双 C-元层的改进型神经认知机。

Van Ooyen 和 Niehuis 为提高神经认知机的区别能力引入了一个新的参数。事实上，该参数作为一种抑制信号，抑制了神经元对重复激

励特征的激励。多数神经网络在权值中记忆训练信息。根据 Hebb 学习规则，某种特征训练的次数越多，在以后的识别过程中就越容易被检测。也有学者将进化计算理论与神经认知机结合，通过减弱对重复性激励特征的训练学习，而使得网络注意那些不同的特征以助于提高区分能力。上述都是神经认知机的发展过程，而卷积神经网络可看作是神经认知机的推广形式，神经认知机是卷积神经网络的一种特例。

2）卷积神经网络的网络结构

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。



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和可加偏置进行卷积，滤波过程如图一，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。这些映射图再进过滤波得到 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。

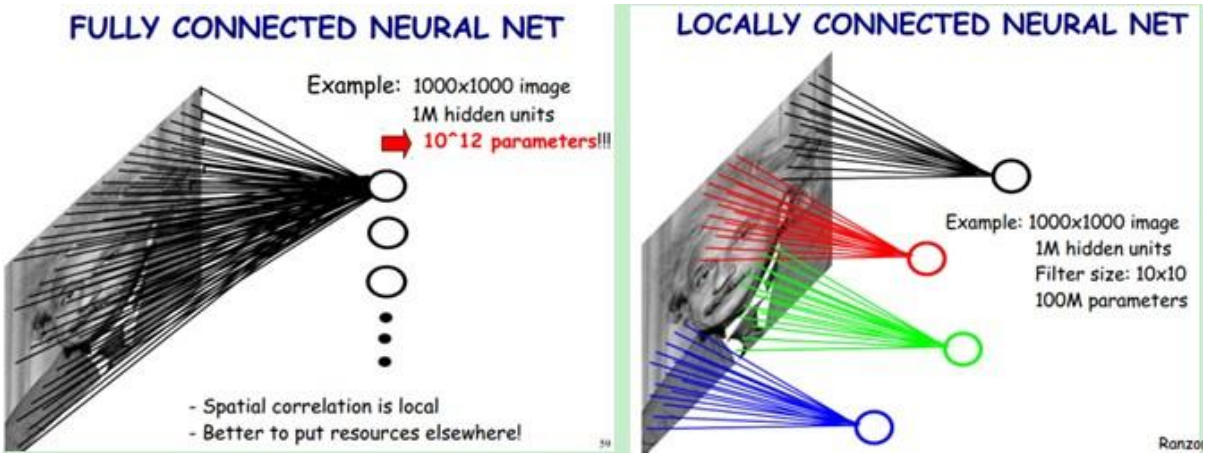
此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数，降低了网络参数选择的复杂度。卷积神经网络中的每一个特征提取层（C-层）都紧跟着一个用来求局部平均与二次提取的计算层（S-层），这种特有的两次特征提取结构使网络在识别时对输入样本有较高的畸变容忍能力。

3）关于参数减少与权值共享

上面聊到，好像 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。那究竟是啥的呢？

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

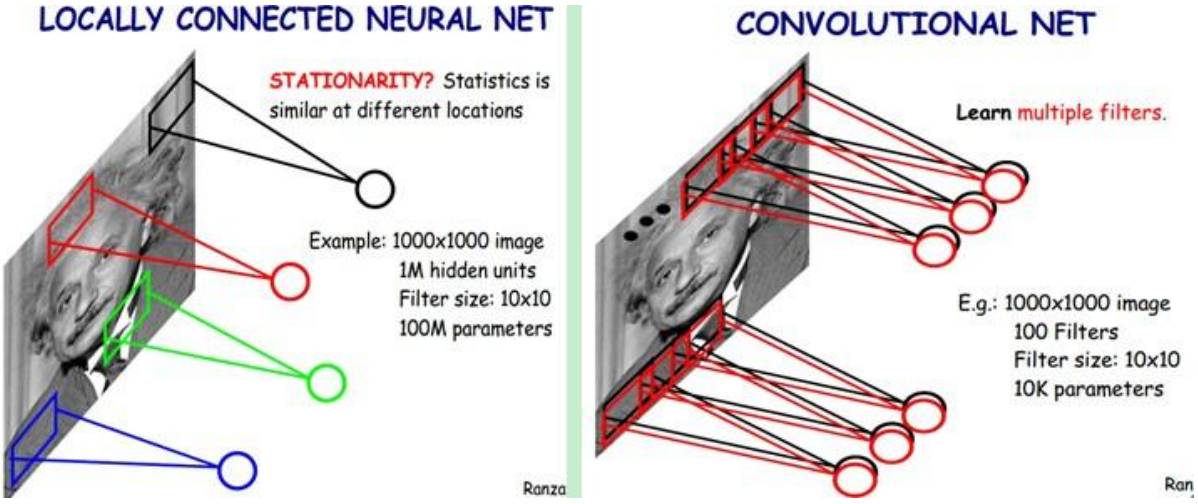
接的数目，也就是减少神经网络需要训练的权值参数的个数了。如下图右：假如局部感受野是 10x10，隐层每个感受野只需要和这 10x10 的局部图像相连接，所以 1 百万个隐层神经元就只有一亿个连接，即 10^8 个参数。比原来减少了四个 0（数量级），这样训练起来就没那么费力了，但还是感觉很多的啊，那还有啥办法没？



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！（有点烦了，呵呵）也许你会问，这样做靠谱吗？为什么可行呢？这个……共同学习。

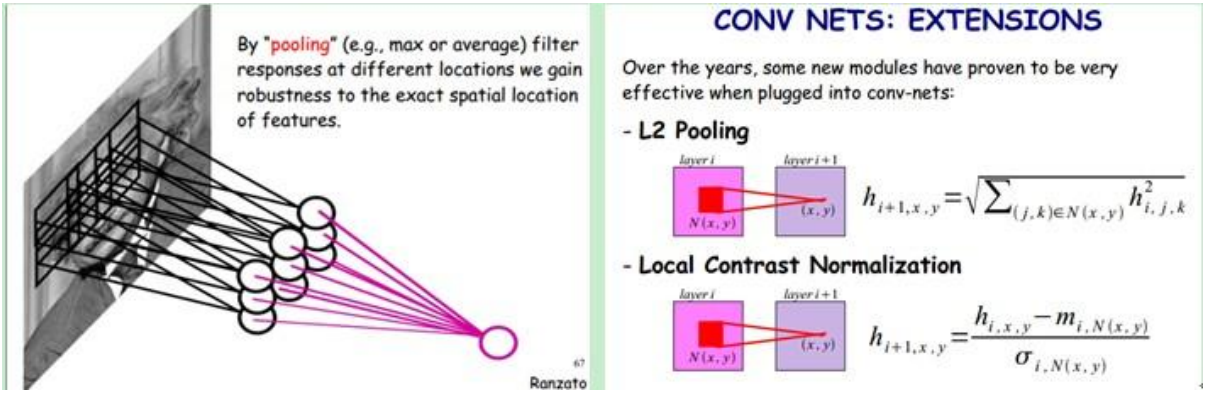
好了，你就会想，这样提取特征也忒不靠谱吧，这样你只提取了一种特征啊？对了，真聪明，我们需要提取多种特征对不？假如一种滤波器，也就是一种卷积核就是提出图像的一种特征，例如某个方向的边缘。

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！（又来了，受不了了！）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。

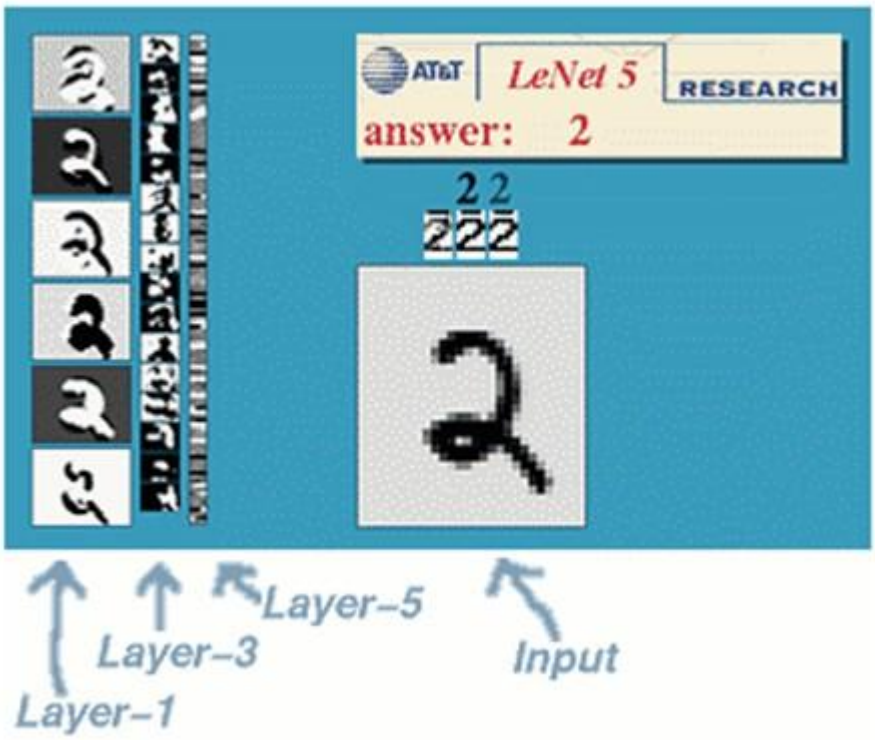


需要注意的一点是，上面的讨论都没有考虑每个神经元的偏置部分。所以权值个数需要加 1 。这个也是同一种滤波器共享的。

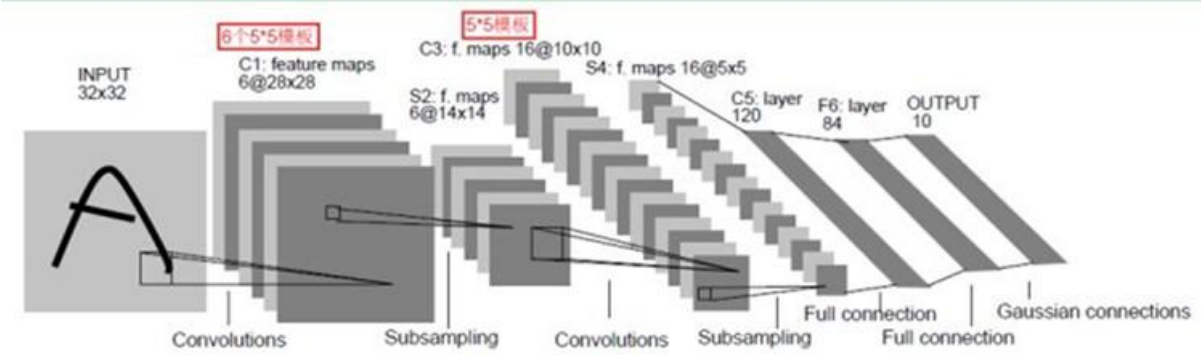
总之，卷积网络的核心思想是将：局部感受野、权值共享（或者权值复制）以及时间或空间亚采样这三种结构思想结合起来获得了某种程度的位移、尺度、形变不变性。

4）一个典型的例子说明

一种典型的用来识别数字的卷积网络是 LeNet-5（效果和 paper 等见这）。当年美国大多数银行就是用它来识别支票上面的手写数字的。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。

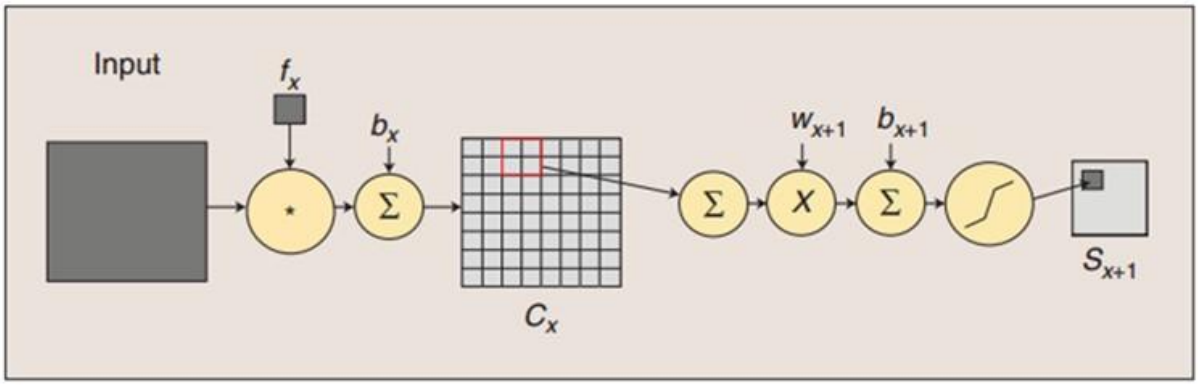


LeNet-5 共有 7 层，不包含输入，每层都包含可训练参数（连接权重）。输入图像为 32*32 大小。这要比 Mnist 数据库（一个公认的手写数据库）中最大的字母还大。这样做的原因是希望潜在的明显特征如笔画断电或角点能够出现在最高层特征监测子感受野的中心。

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。

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。

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 f_x 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 b_x，得到卷积层 C_x。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 W_{x+1} 加权，再增加偏置 b_{x+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 S_{x+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。

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。

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。

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。

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。

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。

使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。

5）训练过程

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,Y_p)，将 X 输入网络；

b）计算相应的实际输出 O_p。

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）：

$$O_p=F_n（…（F_2（F_1（X_pW^{(1)}）W^{(2)}）…）W^{(n)}）$$

第二阶段，向后传播阶段

a）算实际输出 O_p 与相应的理想输出 Y_p 的差；

b）按极小化误差的方法反向传播调整权矩阵。

6）卷积神经网络的优点

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。

7）小结

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。

呵呵，这部分讲得太啰嗦了，又没讲到点上。没办法了，先这样的，这样这个过程我还没有走过，所以自己水平有限啊，望各位明察。需要后面再改了，呵呵。

转自：http://blog.csdn.net/zouxy09/article/details/8781543

Deep Learning 论文笔记之（四）CNN 卷积神经网络推导和实现

Deep Learning 论文笔记之（四）CNN 卷积神经网络推导和实现

zouxy09@qq.com
http://blog.csdn.net/zouxy09

自己平时看了一些论文，但老感觉看完过后就会慢慢的淡忘，某一天重新拾起来的时候又好像没有看过一样。所以想习惯地把一些感觉有用的论文中的知识点总结整理一下，一方面在整理过程中，自己的理解也会更深，另一方面也方便未来自己的勘察。更好的还可以放到博客上面与大家交流。因为基础有限，所以对论文的一些理解可能不太正确，还望大家不吝指正交流，谢谢。

本文的论文来自：

Notes on Convolutional Neural Networks, Jake Bouvrie。

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

[1] Deep Learning（深度学习）学习笔记整理系列之（七）

[2] LeNet-5, convolutional neural networks

[3]卷积神经网络

[4] Neural Network for Recognition of Handwritten Digits

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

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

[7]Imagenet classification with deep convolutional neural networks.

[8] UFLDL 中的“卷积特征提取”和“池化”。

另外，这里有个 matlab 的 Deep Learning 的 toolbox，里面包含了 CNN 的代码，在下一个博文中，我将会详细注释这个代码。这个笔记对这个代码的理解非常重要。

下面是自己对其中的一些知识点的理解：

《Notes on Convolutional Neural Networks》

一、介绍

这个文档讨论的是 CNNs 的推导和实现。CNN 架构的连接比权值要多很多，这实际上就隐含着实现了某种形式的规则化。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，

我们转向讨论如何自动地学习组合前一层的特征 maps，特别地，我们还学习特征 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）

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

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

$$\Delta \mathbf{W}^{\ell} = -\eta \frac{\partial E}{\partial \mathbf{W}^{\ell}}$$ 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：

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

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。

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

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）。

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

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：

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

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：

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

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

神经网络：卷积神经网络

一、前言

这篇卷积神经网络是前面介绍的多层神经网络的进一步深入，它将深度学习的思想引入到了神经网络当中，通过卷积运算来由浅入深的提取图像的不同层次的特征，而利用神经网络的训练过程让整个网络自动调节卷积核的参数，从而无监督的产生了最适合的分类特征。这个概括可能有点抽象，我尽量在下面描述细致一些，但如果要更深入了解整个过程的原理，需要去了解 DeepLearning。

这篇文章会涉及到卷积的原理与图像特征提取的一般概念，并详细描述卷积神经网络的实现。但是由于精力有限，没有对人类视觉的分层以及机器学习等原理有进一步介绍，后面会在深度学习相关文章中展开描述。

二、卷积

卷积是分析数学中一种很重要的运算，其实是一个很简单的概念，但是很多做图像处理的人对这个概念都解释不清，为了简单起见，这里面我们只介绍离散形式的卷积，那么在图像上，对图像用一个卷积核进行卷积运算，实际上是一个滤波的过程。我们先看一下卷积的基本数学表示：

$$f(x,y)\circ w(x,y)=\sum_{s=-a}^{a}\sum_{t=-b}^{b}w(s,t)f(x-s,y-t)$$

其中 $I=f(x,y)$ 是一个图像，$f(x,y)$ 是图像 I 上面 x 行 y 列上点的灰度值。而 $w(x,y)$ 有太多名字叫了，滤波器、卷积核、响应函数等等，而 a 和 b 定义了卷积核即 $w(x,y)$ 的大小。

从上面的式子中，可以很明显的看到，卷积实际上是提供了一个权重模板，这个模板在图像上滑动，并将中心依次与图像中每一个像素对齐，然后对这个模板覆盖的所有像素进行加权，并将结果作为这个卷积核在图像上该点的响应。所以从整个卷积运算我们可以看到以下几点：

1）卷积是一种线性运算

2）卷积核的大小，定义了图像中任何一点参与运算的邻域的大小。

3）卷积核上的权值大小说明了对应的邻域点对最后结果的贡献能力，权重越大，贡献能力越大。

4）卷积核沿着图像所有像素移动并计算响应，会得到一个和原图像等大图像。

5）在处理边缘上点时，卷积核会覆盖到图像外层没有定义的点，这时候有几种方法设定这些没有定义的点，可以用内层像素镜像复制，也可以全设置为 0。

35	124	211
248	195	143
82	20	206

★

w1	w2	w3
w4	w5	w6
w7	w8	w9

$$35 * w_1 + 124 * w_2 + 211 * w_3 + 248 * w_4 + 195 * w_5 + 143 * w_6 + 82 * w_7 + 20 * w_8 + 206 * w_9$$

三、卷积特征层

其实图像特征提取在博客里的一些其他文章都有提过，这里我想说一下图像特征提取与卷积的关系。其实大部分的图像特征提取都依赖于卷积运算，比如显著的边缘特征就是用各种梯度卷积算子对图像进行滤波的结果。一个图像里目标特征主要体现在像素与周围像素之间形成的关系，这些邻域像素关系形成了线条、角点、轮廓等。而卷积运算正是这种用邻域点按一定权重去重新定义该点值的运算。

水平梯度的卷积算子：

-1	-2	-1
0	0	0
1	2	1

竖直梯度的卷积算子：

-1	0	1
-2	0	2
-1	0	1

根据深度学习关于人的视觉分层的理论，人的视觉对目标的辨识是分层的，低层会提取一些边缘特征，然后高一些层次进行形状或目标的认知，更高层的会分析一些运动和行为。也就是说高层的特征是低层特征的组合，从低层到高层的特征表示越来越抽象，越来越能表现语义或者意图。而抽象层面越高，存在的可能猜测就越少，就越利于分类。

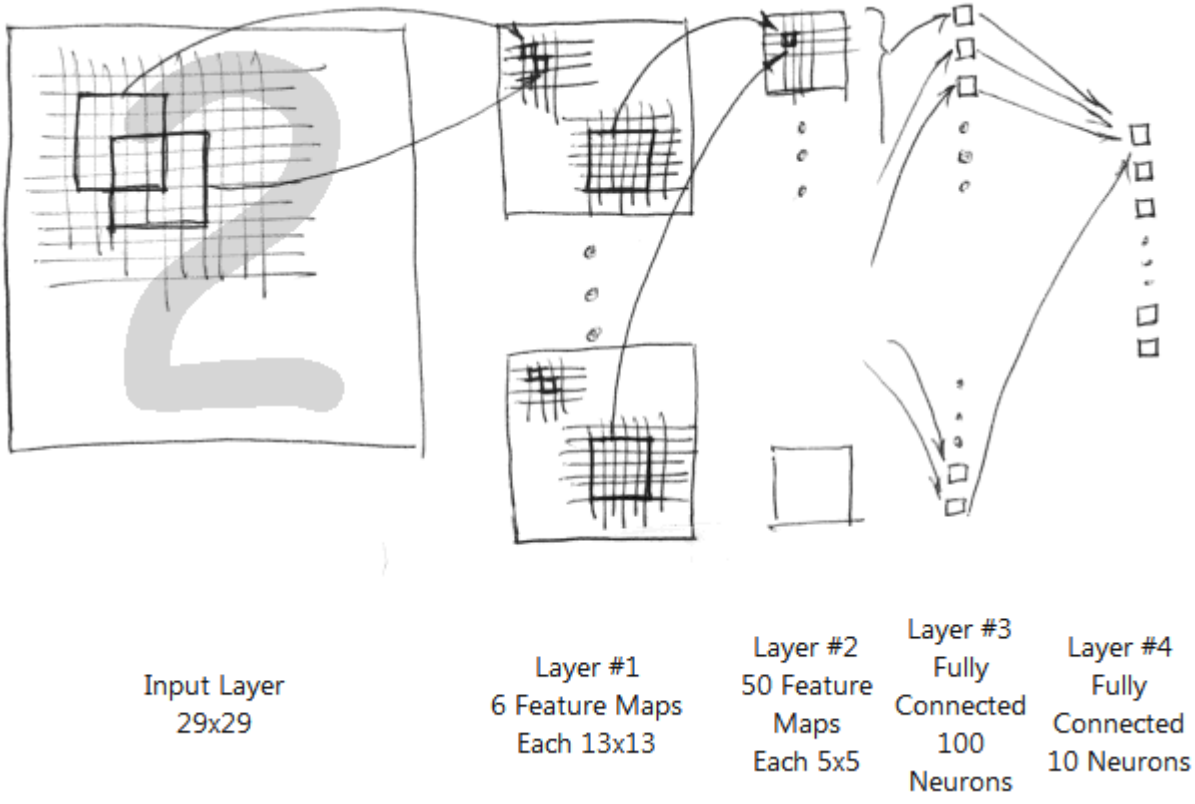
而深度学习就是通过这种分层的自动特征提取来达到目标分类，先构建一些基本的特征层，然后用这些基础特征去构建更高层的抽象，更精准的分类特征。

那整个卷积神经网络的结构也就很容易理解了，它在普通的多层神经网络的前面加了 2 层特征层，这两层特征层是通过权重可调整的卷积运算实现的。

四、卷积神经网络

在原来的多层神经网络结构中，每一层的所有结点会按照连续线的权重向前计算，成为下一层结点的输出。而每一条权重连结线都彼此不同，互不共享。每一个下一层结点的值与前一层所有结

点都相关。
与普通多层神经网络不同的是，卷积神经网络里，有特征抽取层与降维层，这些层的结点连结是部分连接且，一幅特征图由一个卷积核生成，这一幅特征图上的所有结点共享这一组卷积核的参数。
这里我们设计一个 5 层的卷积神经网络，一个输入层，一个输出层，2 个特征提取层，一个全连接的隐藏层。下面详细说明每一层的设计思路。



在一般的介绍卷积神经网络的文章中你可能会看到在特征层之间还有 2 层降维层，在这里我们将卷积与降维同步进行，只用在卷积运算时，遍历图像中像素时按步长间隔遍历即可。

输入层：普通的多层神经网络，第一层就是特征向量。一般图像经过人为的特征挑选，通过特征函数计算得到特征向量，并作为神经网络的输入。而卷积神经网络的输出层则为整个图像，如上图示例 29*29,那么我们可以将图像按列展开，形成 841 个结点。而第一层的结点向前没有任何的连结线。

第 1 层： 第 1 层是特征层，它是由 6 个卷积模板与第一层的图像做卷积形成的 6 幅 13*13 的图像。也就是说这一层中我们算上一个偏置权重，一共有只有（5*5+1）*6=156 权重参数，但是，第二层的结点是将 6 张图像按列展开，即有 6*13*13=1014 个结点。
而第 2 层构建的真正难点在于，连结线的设定，当前层的结点并不是与前一层的所有结点相连，而是只与 25 邻域点连接，所以第二层一定有（25+1）*13*13*6=26364 条连线，但是这些连结线共享了 156 个权值。按前面多层网络 C++的设计中，每个连线对象有 2 个成员，一个是权重的索引，一个是上一层结点的索引，所以这里面要正确的设置好每个连线的索引值，这也是卷积神经网络与一般全连结层的区别。

第 2 层： 第 2 层也是特征层，它是由 50 个特征图像组成，每个特征图像是 5*5 的大小，这

个特征图像上的每一点都是由前一层 6 张特征图像中每个图像取 25 个邻域点最后在一起加权而成，所以每个点也就是一个结点有（5*5+1）*6=156 个连结线，那么当前层一共有 5*5*50=1250 个结点，所以一共有 195000 个权重连结线，但是只有(5*5+1)*6*50=7800 个权重参数，每个权重连结线的值都可以在 7800 个权重参数数组中找到索引。
所以第 3 层的关键也是，如何建立好每根连结线的权重索引与与前一层连结的结点的索引。

第 3 层： 和普通多神经网络没有区别了，是一个隐藏层，有 100 个结点构成，每个结点与上一层中 1250 个结点相联，共有 125100 个权重与 125100 个连结线。

第 4 层： 输出层，它个结点个数与分类数目有关，假设这里我们设置为 10 类，则输出层为 10 个结点，相应的期望值的设置在多层神经网络里已经介绍过了，每个输出结点与上面隐藏层的 100 个结点相连，共有（100+1）*10=1010 条连结线，1010 个权重。
从上面可以看出，卷积神经网络的核心在于卷积层的创建，所以在设计 CNNLayer 类的时候，需要两种创建网络层的成员函数，一个用于创建普通的全连接层，一个用于创建卷积层。

```
1class CNNlayer
2 {
3private:
4        CNNlayer* preLayer;
5        vector<CNNneural> m_neurals;
6        vector<double> m_weights;
7public:
8        CNNlayer(){ preLayer = nullptr; }
9// 创建卷积层
10void createConvLayer(unsigned curNumberOfNeurals, unsigned preNumberOfNeurals, unsigned preNumberOfFeatMaps, unsigned curNumberOfFeatMaps);
11// 创建普通层
12void createLayer(unsigned curNumberOfNeurals,unsigned preNumberOfNeurals);
13void backPropagate(vector<double>& dErrWrtDxn, vector<double>& dErrWrtDxnm, double eta);
14 };
```

创建普通层，在前面介绍的多层神经网络中已经给出过代码，它接收两个参数，一个是前面一层结点数，一个是当前层结点数。
而卷积层的创建则复杂的多，所有连结线的索引值的确定需要对整个网络有较清楚的了解。这里设计的 createConvLayer 函数，它接收 4 个参数，分别对应，当前层结点数，前一层结点数，前一层特征图的个数和当前层特征图像的个数。
下面是 C++代码，要理解这一部分可以会稍有点难度，因为特征图实际中都被按列展开了，所以邻域这个概念会比较抽象，我们考虑把特征图像还原，从图像的角度去考虑。

```
1void CNNlayer::createConvLayer
2    (unsigned curNumberOfNeurals, unsigned preNumberOfNeurals, unsigned preNumberOfFeatMaps, unsigned curNumberOfFeatMaps)
3 {
4// 前一层和当前层特征图的结点数
5        unsigned preImgSize = preNumberOfNeurals / preNumberOfFeatMaps;
```

```
 6        unsigned curImgSize = curNumberOfNeurals / curNumberOfFeatMaps;
 7
 8// 初始化权重
 9        unsigned numberOfWeights = preNumberOfFeatMaps*curNumberOfFeatMaps*(5 * 5 +
1);
10for (unsigned i = 0; i != numberOfWeights; i++)
11        {
12              m_weights.push_back(0.05*rand() / RAND_MAX);
13        }
14// 建立所有连结线
15
16for (unsigned i = 0; i != curNumberOfFeatMaps; i++)
17        {
18                    unsigned  imgRow  =  sqrt(preImgSize);    // 上一层特征图像的大
小 ,imgRow=imgCol
19//  间隙 2 进行取样，邻域周围 25 个点
20for (int c = 2; c < imgRow-2; c= c+ 2)
21              {
22for (int r = 2; r < imgRow-2; r =r + 2)
23                    {
24                          CNNneural neural;
25for (unsigned k = 0; k != preNumberOfNeurals; k++)
26                          {
27for (int kk = 0; kk < (5*5+1); kk ++)
28                                {
29                                      CNNconnection connection;
30// 权重的索引
31                                      connection.weightIdx = i*(curNumberOfFeatMaps*(5 * 5 + 1)) +
k*(5 * 5 + 1) + kk;
32// 结点的索引
33                                      connection.neuralIdx = k*preImgSize + c*imgRow + r;
34                                      neural.m_connections.push_back(connection);
35                                }
36                                m_neurals.push_back(neural);
37                          }
38                    }
39              }
40        }
41 }
```

五、训练与识别

整个网络结构搭建好以后，训练只用按照多层神经网络那样训练即可，其中的权值更新策略都是一致的。所以总体来说，卷积神经网络与普通的多层神经网络，就是结构上不同。卷积神经网络多了特征提取层与降维层，他们之间结点的连结方式是部分连结，多个连结线共享权重。而多层

神经网络前后两层之间结点是全连结。除了这以外，权值更新、训练、识别都是一致的。
训练得到一组权值，也就是说在这组权值下网络可以更好的提取图像特征用于分类识别。
关于源码的问题：个人非常不推荐直接用别人的源码，所以我的博客里所有文章不会给出整个工程的源码，但是会给出一些核心函数的代码，如果你仔细阅读文章，一定能够很好的理解算法的核心思想。尝试着去自己实现，会对你的理解更有帮助。有什么疑问可以直接在下面留言。
转自：

http://www.cnblogs.com/ronny/p/ann_03.html

CNN 卷积神经网络代码理解

Deep Learning 论文笔记之（五）CNN 卷积神经网络代码理解

zouxy09@qq.com
http://www.csdn123.com/link.php?url=http://blog.csdn.net/zouxy09

自己平时看了一些论文，但老感觉看完过后就会慢慢的淡忘，某一天重新拾起来的时候又好像没有看过一样。所以想习惯地把一些感觉有用的论文中的知识点总结整理一下，一方面在整理过程中，自己的理解也会更深，另一方面也方便未来自己的勘察。更好的还可以放到博客上面与大家交流。因为基础有限，所以对论文的一些理解可能不太正确，还望大家不吝指正交流，谢谢。

本文的代码来自 githup 的 Deep Learning 的 toolbox，（在这里，先感谢该 toolbox 的作者）里面包含了很多 Deep Learning 方法的代码。是用 Matlab 编写的（另外，有人翻译成了 C++和 python 的版本了）。本文中我们主要解读下 CNN 的代码。详细的注释见代码。

在读代码之前，最好先阅读下我的上一个博文：

Deep Learning 论文笔记之（四）CNN 卷积神经网络推导和实现

http://www.csdn123.com/link.php?url=http://blog.csdn.net/zouxy09

/article/details/9993371

里面包含的是我对一个作者的 CNN 笔记的翻译性的理解，对 CNN 的推导和实现做了详细的介绍，看明白这个笔记对代码的理解非常重要，所以强烈建议先看懂上面这篇文章。

下面是自己对代码的注释：

cnnexamples.m

```
clear all; close all; clc;
addpath('../data');
addpath('../util');
load mnist_uint8;

train_x = double(reshape(train_x',28,28,60000))/255;
test_x = double(reshape(test_x',28,28,10000))/255;
train_y = double(train_y');
test_y = double(test_y');

%% ex1
%will run 1 epoch in about 200 second and get around 11% error.
%With 100 epochs you'll get around 1.2% error

cnn.layers = {
    struct('type', 'i') %input layer
    struct('type', 'c', 'outputmaps', 6, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %sub sampling layer
    struct('type', 'c', 'outputmaps', 12, 'kernelsize', 5) %convolution layer
    struct('type', 's', 'scale', 2) %subsampling layer
};

% 这里把 cnn 的设置给 cnnsetup，它会据此构建一个完整的 CNN 网络，并返回
cnn = cnnsetup(cnn, train_x, train_y);

% 学习率
opts.alpha = 1;
% 每次挑出一个 batchsize 的 batch 来训练,也就是每用 batchsize 个样本就调整一次权值，而不是
% 把所有样本都输入了，计算所有样本的误差了才调整一次权值
```

```
opts.batchsize = 50;
% 训练次数，用同样的样本集。我训练的时候：
% 1 的时候 11.41% error
% 5 的时候 4.2% error
% 10 的时候 2.73% error
opts.numepochs = 10;

% 然后开始把训练样本给它，开始训练这个 CNN 网络
cnn = cnntrain(cnn, train_x, train_y, opts);

% 然后就用测试样本来测试
[er, bad] = cnntest(cnn, test_x, test_y);

%plot mean squared error
plot(cnn.rL);
%show test error
disp([num2str(er*100) '% error']);
```

cnnsetup.m

```
function net = cnnsetup(net, x, y)
    inputmaps = 1;
    % B=squeeze(A) 返回和矩阵 A 相同元素但所有单一维都移除的矩阵 B，单一维是满足 size(A,dim)=1 的维。
    % train_x 中图像的存放方式是三维的 reshape(train_x',28,28,60000)，前面两维表示图像的行与列，
    % 第三维就表示有多少个图像。这样 squeeze(x(:, :, 1))就相当于取第一个图像样本后，再把第三维
    % 移除，就变成了 28x28 的矩阵，也就是得到一幅图像，再 size 一下就得到了训练样本图像的行数与列数了
    mapsize = size(squeeze(x(:, :, 1)));

    % 下面通过传入 net 这个结构体来逐层构建 CNN 网络
    % n = numel(A)返回数组 A 中元素个数
    % net.layers 中有五个 struct 类型的元素，实际上就表示 CNN 共有五层，这里范围的是 5
    for l = 1 : numel(net.layers)   %  layer
        if strcmp(net.layers{l}.type, 's') % 如果这层是 子采样层
            % subsampling 层的 mapsize，最开始 mapsize 是每张图的大小 28*28
            % 这里除以 scale=2,就是 pooling 之后图的大小,pooling 域之间没有重叠，所以 pooling 后的图像为 14*14
            % 注意这里的右边的 mapsize 保存的都是上一层每张特征 map 的大小,它会随着循环进行不断更新
            mapsize = floor(mapsize / net.layers{l}.scale);
```

```
            for j = 1 : inputmaps % inputmap 就是上一层有多少张特征图
                net.layers{l}.b{j} = 0; % 将偏置初始化为 0
            end
        end
        if strcmp(net.layers{l}.type, 'c') % 如果这层是 卷积层
            % 旧的 mapsize 保存的是上一层的特征 map 的大小,那么如果卷积核的移动步长是 1，那用
            % kernelsize*kernelsize 大小的卷积核卷积上一层的特征 map 后，得到的新的 map 的大小就是下面这样
            mapsize = mapsize - net.layers{l}.kernelsize + 1;
            % 该层需要学习的参数个数。每张特征 map 是一个(后层特征图数量)*(用来卷积的 patch 图的大小)
            % 因为是通过用一个核窗口在上一个特征 map 层中移动（核窗口每次移动 1 个像素），遍历上一个特征 map
            % 层的每个神经元。核窗口由 kernelsize*kernelsize 个元素组成，每个元素是一个独立的权值，所以
            % 就有 kernelsize*kernelsize 个需要学习的权值，再加一个偏置值。另外，由于是权值共享，也就是
            % 说同一个特征 map 层是用同一个具有相同权值元素的 kernelsize*kernelsize 的核窗口去感受输入上一
            % 个特征 map 层的每个神经元得到的，所以同一个特征 map，它的权值是一样的，共享的，权值只取决于
            % 核窗口。然后，不同的特征 map 提取输入上一个特征 map 层不同的特征，所以采用的核窗口不一样，也
            % 就是权值不一样，所以 outputmaps 个特征 map 就有（kernelsize*kernelsize+1）* outputmaps 那么多的权值了
            % 但这里 fan_out 只保存卷积核的权值 W，偏置 b 在下面独立保存
            fan_out = net.layers{l}.outputmaps * net.layers{l}.kernelsize ^ 2;
            for j = 1 : net.layers{l}.outputmaps  %  output map
                % fan_out 保存的是对于上一层的一张特征 map，我在这一层需要对这一张特征 map 提取 outputmaps 种特征，
                % 提取每种特征用到的卷积核不同,所以 fan_out 保存的是这一层输出新的特征需要学习的参数个数
                % 而,fan_in 保存的是,我在这一层,要连接到上一层中所有的特征 map，然后用 fan_out 保存的提取特征
                % 的权值来提取他们的特征。也即是对于每一个当前层特征图，有多少个参数链到前层
                fan_in = inputmaps * net.layers{l}.kernelsize ^ 2;
                for i = 1 : inputmaps  %  input map
                    % 随机初始化权值，也就是共有 outputmaps 个卷积核，对上层的每个特征 map，都需要用这么多个卷积核
                    % 去卷积提取特征。
                    % rand(n)是产生 n×n 的 0-1 之间均匀取值的数值的矩阵，再减去 0.5 就相当于产生-0.5 到 0.5 之间的随机数
```

```
                    % 再 *2 就放大到 [-1, 1]。然后再乘以后面那一数，why?
                    % 反正就是将卷积核每个元素初始化为[-sqrt(6 / (fan_in +
fan_out)), sqrt(6 / (fan_in + fan_out))]
                    % 之间的随机数。因为这里是权值共享的，也就是对于一张特征 map，
所有感受野位置的卷积核都是一样的
                    % 所以只需要保存的是 inputmaps * outputmaps 个卷积核。
                    net.layers{l}.k{i}{j} = (rand(net.layers{l}.kernelsize) - 0.5)
* 2 * sqrt(6 / (fan_in + fan_out));
                end
                net.layers{l}.b{j} = 0; % 将偏置初始化为 0
            end
            % 只有在卷积层的时候才会改变特征 map 的个数，pooling 的时候不会改变个
数。这层输出的特征 map 个数就是
            % 输入到下一层的特征 map 个数
            inputmaps = net.layers{l}.outputmaps;
        end
    end

    % fvnum 是输出层的前面一层的神经元个数。
    % 这一层的上一层是经过 pooling 后的层，包含有 inputmaps 个特征 map。每个特征 map
的大小是 mapsize。
    % 所以，该层的神经元个数是 inputmaps * （每个特征 map 的大小）
    % prod: Product of elements.
    % For vectors, prod(X) is the product of the elements of X
    % 在这里 mapsize = [特征 map 的行数 特征 map 的列数]，所以 prod 后就是 特征 map
的行*列
    fvnum = prod(mapsize) * inputmaps;
    % onum 是标签的个数，也就是输出层神经元的个数。你要分多少个类，自然就有多少
个输出神经元
    onum = size(y, 1);

    % 这里是最后一层神经网络的设定
    % ffb 是输出层每个神经元对应的基 biases
    net.ffb = zeros(onum, 1);
    % ffW 输出层前一层 与 输出层 连接的权值，这两层之间是全连接的
    net.ffW = (rand(onum, fvnum) - 0.5) * 2 * sqrt(6 / (onum + fvnum));
end
```

cnntrain.m

```
function net = cnntrain(net, x, y, opts)
    m = size(x, 3); % m 保存的是 训练样本个数
    numbatches = m / opts.batchsize;
    % rem: Remainder after division. rem(x,y) is x - n.*y 相当于求余
```

```
    % rem(numbatches, 1) 就相当于取其小数部分，如果为0，就是整数
    if rem(numbatches, 1) ~= 0
        error('numbatches not integer');
    end

    net.rL = [];
    for i = 1 : opts.numepochs
        % disp(X) 打印数组元素。如果X是个字符串，那就打印这个字符串
        disp(['epoch ' num2str(i) '/' num2str(opts.numepochs)]);
        % tic 和 toc 是用来计时的，计算这两条语句之间所耗的时间
        tic;
        % P = randperm(N) 返回[1, N]之间所有整数的一个随机的序列，例如
        % randperm(6) 可能会返回 [2 4 5 6 1 3]
        % 这样就相当于把原来的样本排列打乱，再挑出一些样本来训练
        kk = randperm(m);
        for l = 1 : numbatches
            % 取出打乱顺序后的batchsize个样本和对应的标签
            batch_x = x(:, :, kk((l - 1) * opts.batchsize + 1 : l * opts.batchsize));
            batch_y = y(:,     kk((l - 1) * opts.batchsize + 1 : l * opts.batchsize));

            % 在当前的网络权值和网络输入下计算网络的输出
            net = cnnff(net, batch_x); % Feedforward
            % 得到上面的网络输出后，通过对应的样本标签用bp算法来得到误差对网络
权值
            %（也就是那些卷积核的元素）的导数
            net = cnnbp(net, batch_y); % Backpropagation
            % 得到误差对权值的导数后，就通过权值更新方法去更新权值
            net = cnnapplygrads(net, opts);
            if isempty(net.rL)
                net.rL(1) = net.L; % 代价函数值，也就是误差值
            end
            net.rL(end + 1) = 0.99 * net.rL(end) + 0.01 * net.L; % 保存历史的误
差值，以便画图分析
        end
        toc;
    end

end
```

cnnff.m

```
function net = cnnff(net, x)
    n = numel(net.layers); % 层数
    net.layers{1}.a{1} = x; % 网络的第一层就是输入，但这里的输入包含了多个训练图
```

像
```
    inputmaps = 1; % 输入层只有一个特征 map，也就是原始的输入图像

    for l = 2 : n   %  for each layer
        if strcmp(net.layers{l}.type, 'c') % 卷积层
            %  !!below can probably be handled by insane matrix operations
            % 对每一个输入 map，或者说我们需要用 outputmaps 个不同的卷积核去卷积图像
            for j = 1 : net.layers{l}.outputmaps   %  for each output map
                %  create temp output map
                % 对上一层的每一张特征 map，卷积后的特征 map 的大小就是
                % （输入 map 宽 - 卷积核的宽 + 1）* （输入 map 高 - 卷积核高 + 1）
                % 对于这里的层，因为每层都包含多张特征 map，对应的索引保存在每层 map 的第三维
                % 所以，这里的 z 保存的就是该层中所有的特征 map 了
                z = zeros(size(net.layers{l - 1}.a{1}) - [net.layers{l}.kernelsize - 1 net.layers{l}.kernelsize - 1 0]);
                for i = 1 : inputmaps   %  for each input map
                    %  convolve with corresponding kernel and add to temp output map
                    % 将上一层的每一个特征 map（也就是这层的输入 map）与该层的卷积核进行卷积
                    % 然后将对上一层特征 map 的所有结果加起来。也就是说，当前层的一张特征 map，是
                    % 用一种卷积核去卷积上一层中所有的特征 map，然后所有特征 map 对应位置的卷积值的和
                    % 另外,有些论文或者实际应用中,并不是与全部的特征 map 链接的,有可能只与其中的某几个连接
                    z = z + convn(net.layers{l - 1}.a{i}, net.layers{l}.k{i}{j}, 'valid');
                end
                %  add bias, pass through nonlinearity
                % 加上对应位置的基 b，然后再用 sigmoid 函数算出特征 map 中每个位置的激活值，作为该层输出特征 map
                net.layers{l}.a{j} = sigm(z + net.layers{l}.b{j});
            end
            %  set number of input maps to this layers number of outputmaps
            inputmaps = net.layers{l}.outputmaps;
        elseif strcmp(net.layers{l}.type, 's') % 下采样层
            %  downsample
            for j = 1 : inputmaps
                %  !! replace with variable
                % 例如我们要在 scale=2 的域上面执行 mean pooling，那么可以卷积大小为 2*2，每个元素都是 1/4 的卷积核
```

```
                z = convn(net.layers{l - 1}.a{j}, ones(net.layers{l}.scale) /
(net.layers{l}.scale ^ 2), 'valid');
                % 因为 convn 函数的默认卷积步长为 1，而 pooling 操作的域是没有重叠
的，所以对于上面的卷积结果
                % 最终 pooling 的结果需要从上面得到的卷积结果中以 scale=2 为步长，
跳着把 mean pooling 的值读出来
                net.layers{l}.a{j} = z(1 : net.layers{l}.scale : end, 1 :
net.layers{l}.scale : end, :);
            end
        end
    end

    %  concatenate all end layer feature maps into vector
    % 把最后一层得到的特征 map 拉成一条向量，作为最终提取到的特征向量
    net.fv = [];
    for j = 1 : numel(net.layers{n}.a) % 最后一层的特征 map 的个数
        sa = size(net.layers{n}.a{j}); % 第 j 个特征 map 的大小
        % 将所有的特征 map 拉成一条列向量。还有一维就是对应的样本索引。每个样本一
列，每列为对应的特征向量
        net.fv = [net.fv; reshape(net.layers{n}.a{j}, sa(1) * sa(2), sa(3))];
    end
    %  feedforward into output perceptrons
    % 计算网络的最终输出值。sigmoid(W*X + b)，注意是同时计算了 batchsize 个样本的
输出值
    net.o = sigm(net.ffW * net.fv + repmat(net.ffb, 1, size(net.fv, 2)));

end
```

cnnbp.m

```
function net = cnnbp(net, y)
    n = numel(net.layers); % 网络层数

    %  error
    net.e = net.o - y;
    %  loss function
    % 代价函数是 均方误差
    net.L = 1/2* sum(net.e(:) .^ 2) / size(net.e, 2);

    %%  backprop deltas
    % 这里可以参考 UFLDL 的 反向传导算法 的说明
    % 输出层的 灵敏度 或者 残差
    net.od = net.e .* (net.o .* (1 - net.o));   %  output delta
    % 残差 反向传播回 前一层
```

```
    net.fvd = (net.ffW' * net.od);                  %  feature vector delta
    if strcmp(net.layers{n}.type, 'c')              %  only conv layers has sigm function
        net.fvd = net.fvd .* (net.fv .* (1 - net.fv));
    end

    %  reshape feature vector deltas into output map style
    sa = size(net.layers{n}.a{1}); % 最后一层特征 map 的大小。这里的最后一层都是指输出层的前一层
    fvnum = sa(1) * sa(2); % 因为是将最后一层特征 map 拉成一条向量，所以对于一个样本来说，特征维数是这样
    for j = 1 : numel(net.layers{n}.a) % 最后一层的特征 map 的个数
      % 在 fvd 里面保存的是所有样本的特征向量(在 cnnff.m 函数中用特征 map 拉成的)，所以这里需要重新
        % 变换回来特征 map 的形式。d 保存的是 delta，也就是 灵敏度 或者 残差
        net.layers{n}.d{j} = reshape(net.fvd(((j - 1) * fvnum + 1) : j * fvnum, :), sa(1), sa(2), sa(3));
    end

    % 对于 输出层前面的层（与输出层计算残差的方式不同）
    for l = (n - 1) : -1 : 1
        if strcmp(net.layers{l}.type, 'c')
            for j = 1 : numel(net.layers{l}.a) % 该层特征 map 的个数
                % net.layers{l}.d{j} 保存的是 第 l 层 的 第 j 个 map 的 灵敏度 map。也就是每个神经元节点的 delta 的值
                % expand 的操作相当于对 l+1 层的灵敏度 map 进行上采样。然后前面的操作相当于对该层的输入 a 进行 sigmoid 求导
                % 这条公式请参考 Notes on Convolutional Neural Networks
                % for k = 1:size(net.layers{l + 1}.d{j}, 3)
                    % net.layers{l}.d{j}(:, :, k) = net.layers{l}.a{j}(:, :, k) .* (1 - net.layers{l}.a{j}(:, :, k)) .* kron(net.layers{l + 1}.d{j}(:, :, k), ones(net.layers{l + 1}.scale)) / net.layers{l + 1}.scale ^ 2;
                % end
                net.layers{l}.d{j} = net.layers{l}.a{j} .* (1 - net.layers{l}.a{j}) .* (expand(net.layers{l + 1}.d{j}, [net.layers{l + 1}.scale net.layers{l + 1}.scale 1]) / net.layers{l + 1}.scale ^ 2);
            end
        elseif strcmp(net.layers{l}.type, 's')
            for i = 1 : numel(net.layers{l}.a) % 第 l 层特征 map 的个数
                z = zeros(size(net.layers{l}.a{1}));
                for j = 1 : numel(net.layers{l + 1}.a) % 第 l+1 层特征 map 的个数
                    z = z + convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'full');
                end
```

```
                net.layers{l}.d{i} = z;
            end
        end
    end

    %%  calc gradients
    % 这里与 Notes on Convolutional Neural Networks 中不同，这里的 子采样 层没有参数，也没有
    % 激活函数，所以在子采样层是没有需要求解的参数的
    for l = 2 : n
        if strcmp(net.layers{l}.type, 'c')
            for j = 1 : numel(net.layers{l}.a)
                for i = 1 : numel(net.layers{l - 1}.a)
                    % dk 保存的是 误差对卷积核 的导数
                    net.layers{l}.dk{i}{j}   =   convn(flipall(net.layers{l   -
1}.a{i}), net.layers{l}.d{j}, 'valid') / size(net.layers{l}.d{j}, 3);
                end
                % db 保存的是 误差对于 bias 基 的导数
                net.layers{l}.db{j}      =      sum(net.layers{l}.d{j}(:))      /
size(net.layers{l}.d{j}, 3);
            end
        end
    end
    % 最后一层 perceptron 的 gradient 的计算
    net.dffW = net.od * (net.fv)' / size(net.od, 2);
    net.dffb = mean(net.od, 2);

    function X = rot180(X)
        X = flipdim(flipdim(X, 1), 2);
    end
end
```

cnnapplygrads.m

```
function net = cnnapplygrads(net, opts)
    for l = 2 : numel(net.layers)
        if strcmp(net.layers{l}.type, 'c')
            for j = 1 : numel(net.layers{l}.a)
                for ii = 1 : numel(net.layers{l - 1}.a)
                    % 这里没什么好说的，就是普通的权值更新的公式：W_new = W_old - alpha * de/dW（误差对权值导数）
                    net.layers{l}.k{ii}{j} = net.layers{l}.k{ii}{j} - opts.alpha
* net.layers{l}.dk{ii}{j};
                end
```

```
            end
            net.layers{l}.b{j}    =    net.layers{l}.b{j}    -    opts.alpha    *
net.layers{l}.db{j};
        end
    end

    net.ffW = net.ffW - opts.alpha * net.dffW;
    net.ffb = net.ffb - opts.alpha * net.dffb;
end
```

cnntest.m

```
function [er, bad] = cnntest(net, x, y)
    %  feedforward
    net = cnnff(net, x); % 前向传播得到输出
    % [Y,I] = max(X) returns the indices of the maximum values in vector I
    [~, h] = max(net.o); % 找到最大的输出对应的标签
    [~, a] = max(y);     % 找到最大的期望输出对应的索引
    bad = find(h ~= a);  % 找到他们不相同的个数，也就是错误的次数

    er = numel(bad) / size(y, 2); % 计算错误率
end
```