1. **神经网络**

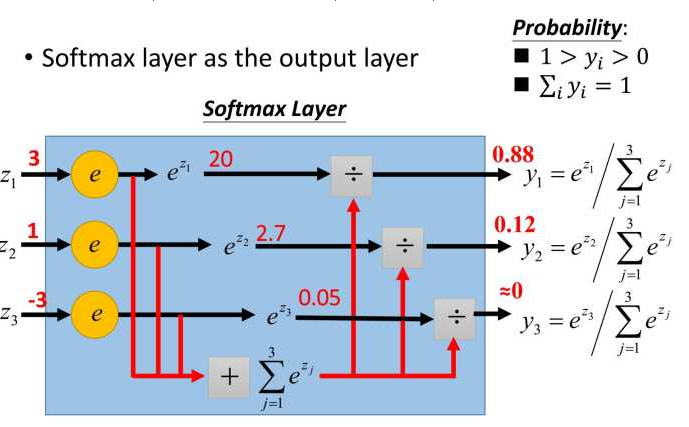
**1.1 Softmax**

<https://blog.csdn.net/bitcarmanlee/article/details/82320853>

<http://www.cnblogs.com/yjmyzz/p/7822990.html>

在多分类问题中，如果希望得到目标对象属于每个类别的概率，而我们通过分类器得到的一般是一系列数值，这里就可以通过Softmax将它们转换成0-1直接的概率，并且概率和为1

Softmax的计算：



Softmax分类和k个二元分类的比较

如果希望将样本分为k类，选择Softmax还是k个二元分类器，取决于是否类别之间是否互斥。如果互斥，则应该选择Softmax，如果不互斥，则可以选择k个二元分类器。

**1.2 极大似然估计(maximum-likelihood)**

<https://blog.csdn.net/zengxiantao1994/article/details/72787849>

先看贝叶斯分类：



其中，为先验概率，表示某种类别的概率分布。为类条件概率，表示属于某个类别的前提下，事件发生的概率。为后验概率，表示事件发生的前提下，属于某个类别的概率。

例子：

已知：在夏季，某公园男性穿凉鞋的概率为1/2，女性穿凉鞋的概率为2/3，并且该公园中男女比例通常为2:1，问题：若你在公园中随机遇到一个穿凉鞋的人，请问他的性别为男性或女性的概率分别为多少？

男性，为女性，为穿凉鞋。

先验概率：

，

类条件概率：

，

穿凉鞋的概率：



则：





在实际问题中，我们往往只能获取有限数目的样本，也无法知晓先验概率和类条件概率。

先验概率可以通过以下方式获取：1、每个样本所属的自然状态都是已知的（有监督学习）；2、依靠经验；3、用训练样本中各类出现的频率估计。

类条件概率：将估计类条件概率转换为参数估计，并采用极大似然估计来解决。

极大似然估计的重要前提：

1. 训练样本的分布能代表样本的真实分布。
2. 每个样本集中的样本都是所谓独立同分布的随机变量。
3. 有充分的训练样本。

极大似然估计：利用已知的样本结果，反推最有可能（最大概率）出现这种结果的参数值。

设样本集为：



我们来估计参数。

似然函数（likelihood function）：联合概率密度函数称为相对于的的似然函数。



其中，最后一个=之所以成立，是因为样本是独立同分布的。

如果能使似然函数最大，则就是最可能的值。



为了便于分析，定义对数似然函数：



对似然函数求导，极大似然估计量为：



极大似然估计的一般步骤：

1. 写出似然对数
2. 对似然对数求导
3. 解似然方程，求出。

极大似然估计的优点：

1. 简单易实现
2. 收敛性好，如果样本趋近总体，则收敛性会变好

极大似然估计的缺点：

1. 如果假设的概率模型正确，则结果会较好；假如模型出现偏差，则效果可能会比较差。
2. **深度神经网络**

**2.1 Keras**

1. 序列模型

from keras.models import Sequential

#Create the Sequential model

model = Sequential()

1. 层

**from** keras.models **import** Sequential

**from** keras.layers.core **import** Dense, Activation, Flatten

*#创建序列模型*

model = Sequential()

*#第一层 - 添加有128个节点的全连接层以及32个节点的输入层*

model.add(Dense(128, input\_dim=32))

*#第二层 - 添加 softmax 激活层*

model.add(Activation('softmax'))

*#第三层 - 添加全连接层*

model.add(Dense(10))

*#第四层 - 添加 Sigmoid 激活层*

model.add(Activation('sigmoid'))

创建32x128x10的神经网络，输入为32结点，中间层为128结点，激活函数为softmax，输出为10个结点，激活函数为sigmoid。

1. 模型编译：损失函数，优化器，评估指标

model.compile(loss="categorical\_crossentropy", optimizer="adam", metrics = ['accuracy'])

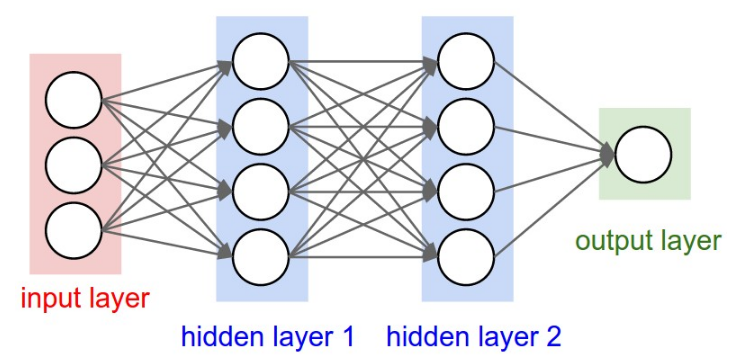
1. **卷积神经网络(CNN)**

CNN最擅长解决图像识别的问题，其输入默认就是图像。

MLP(多层感知器)神经网络可以用于图像识别，但它的缺点较为明显：

1. 参数非常多，Udacity中一个28x28图像使用的神经网络的参数高达60万个，如果使用更大的图片，会非常消耗计算资源。之所以有这么多参数，原因是MLP神经网络使用全连接层，每个输入结点都会与所有隐藏层结点产生权重参数，隐藏层之间也是如此。
2. MLP处理图像时，将图片转换为向量输入，会丢失所有的二维信息。

传统神经网络：



CNN可以接受矩阵输入（二维或三维的），

CNN通常包含以下几层：

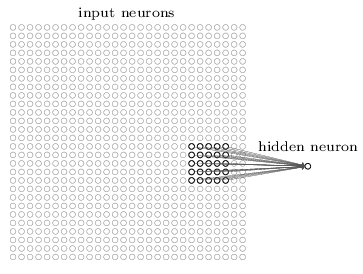
1. 卷积层（Convolutional Layer）
2. 池化层（Pooling Layer）
3. 全连接层（Fully-Connected Layer）



**3.1 卷积层(特征提取层)**

传统神经网络如果输入图片是1000x1000，则需要10^6个输入单元，如果隐藏层也有10^6个结点，则权重系数就会有10^12个，根本无法计算。

卷积层采用局部感知的方法来解决这一问题，每个隐藏层的结点都只与部分输入结点相连。



具体的方式是采用一个filter或kernel对像素矩阵进行卷积来实现。

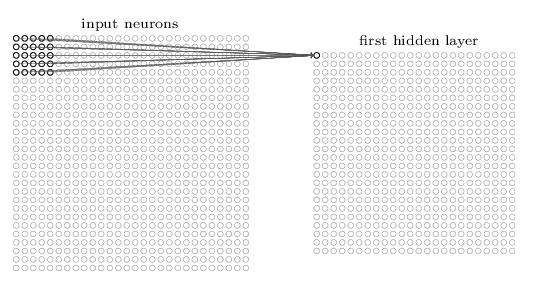
卷积的概念：



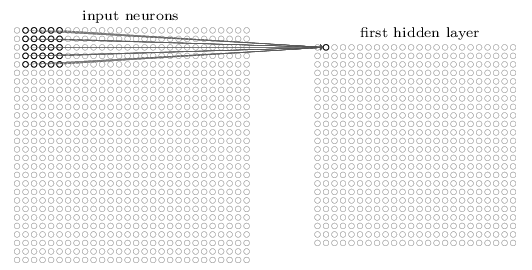
其中称为卷积核，为输入矩阵。

我们移动这样一个filter使其扫描输入矩阵，每次移动1步(stride)都有一个不同的隐藏层结点与之对应。

第1个：



第2个：



以此类推，可以形成第1个隐藏层，如上图所示，输入为28x28，filter为5x5，得到的隐藏层为24x24(24=28-5+1)。也可以设置每次移动多步。

权值共享（Shared weights and biases）：

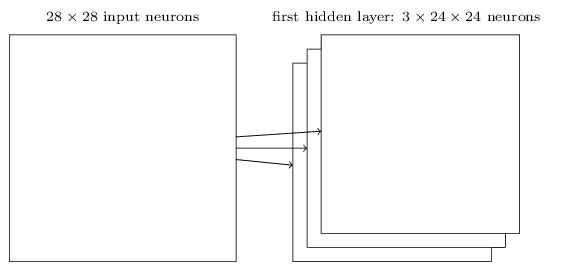
经过局部感知后，隐藏层有24\*24个结点，每个结点对应5\*5个权重系数。



其中代表输入层的矩阵，代表filter中的参数，表示激活函数，一般是Sigmoid或ReLU等，公式表示隐藏层第j行第k列的结点的输入。

虽然比全连接有所降低，但仍然很多，所谓权值共享就是认为隐藏层24\*24个对应的5\*5的权重系数和偏移值都是相同的。这样隐藏层的参数就变成了5\*5.这样认为的原因是：我们认为隐藏层从图像某一部分提取出来的特征也适用于另一部分，这也意味着一个隐藏层只能提取出一种特征。

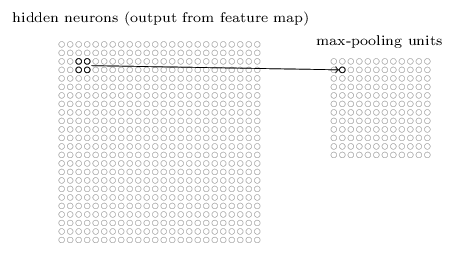
一般CNN都会有多个卷积核(filter)。例，如果使用3个filter:

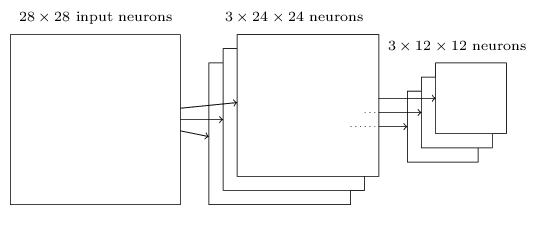


* 1. **池化层**

在卷积层后添加一个池化层，用来简化卷积层的输出，也是使用一个filter，但不进行卷积操作，而是通过最大值，最小值、平均值来进行。

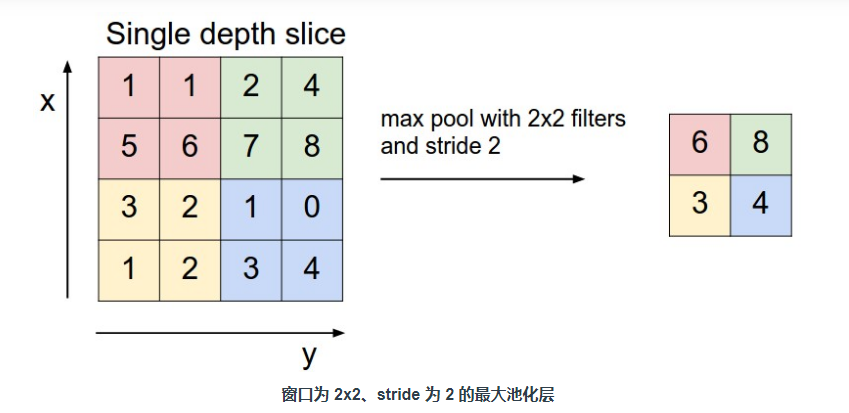
例，2\*2的max filter，选4个参数中最大的一个，从而压缩了卷积层的输出。





经过池化层后，会大大减少我们学到的特征。

Keras中的最大池化层：



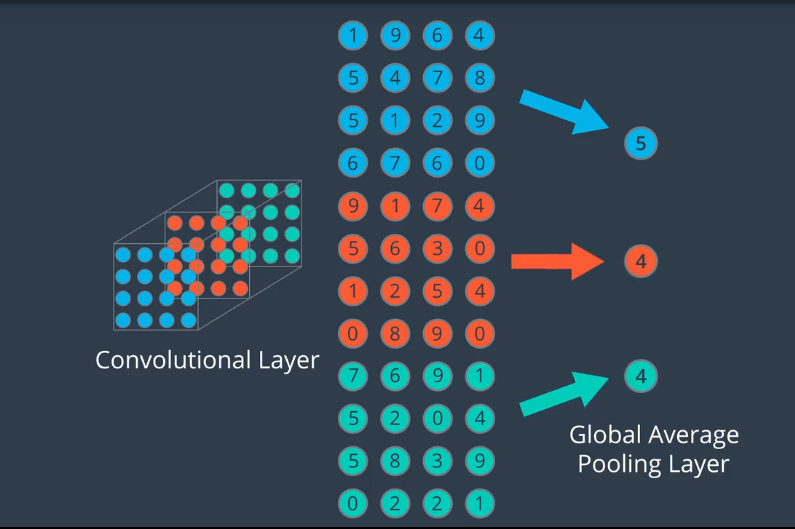
**from** keras.layers **import** MaxPooling2D

MaxPooling2D(pool\_size, strides, padding)

pool\_size：池化层filter的宽和高

stride和padding与卷积层的类似。

全局平均池化层：更为极端的特征压缩

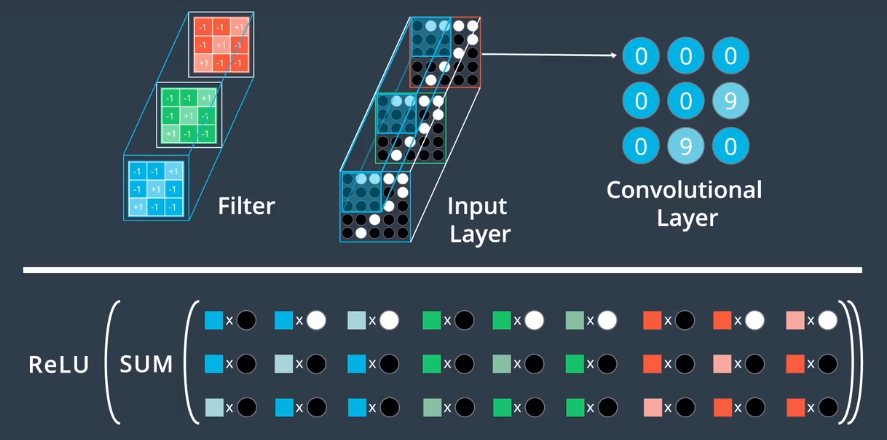


将卷积层每个窗口的权重系数求全局平均，然后作为池化层结点。

* 1. **全连接层（输出层）**

经过池化层的输出与输出层（有可能有多层，全是全连接层）进行全连接，然后得到神经网络的输出。

在Udacity中，处理彩色图像，是将其看成3个（RGB）二维矩阵输入。



在CNN中，第1卷积层与输入层局部相连，第2卷积层与第1卷积层也是局部相连，以此类推，只有最后输出层才是全连接层。

在CNN中，我们可以指定卷尺层的结点数目，filter的大小，但不能指定filter中的权重系数，初始是随机生成的，CNN中的反向传播其实是更新filter中的权重系数。

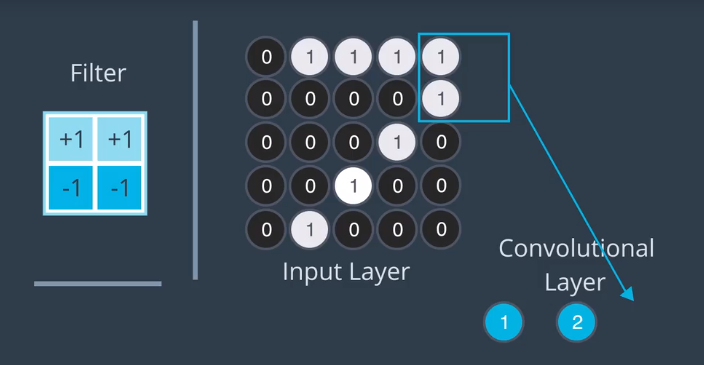
CNN中的超参数：除了结点树，层数，激活函数等之外的其它超参数

1. stride 步幅

卷积时每次移动的像素数目，两个相邻的卷积层结点，与它们相连的输入区域的距离。

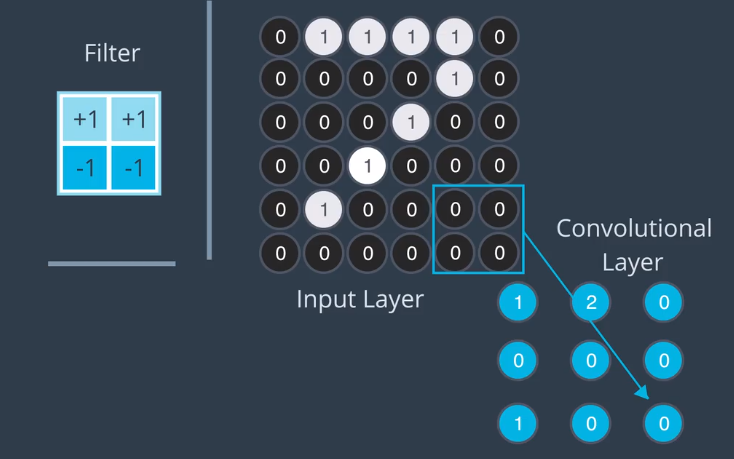
1. padding 填充选择

filter窗卷积时，会出现再图像边缘部分超出图片的情况，如下图所示。



一种处理方法是直接删除掉这些卷积层的结点，但这样做会丢失掉图像的边缘信息。padding=’valid’

另一种处理方法是对图像进行填充，一般是填充0，padding=’same’



Keras中的卷积层：

**from** keras.layers **import** Conv2D

Conv2D(filters, kernel\_size, strides, padding, activation='relu', input\_shape)

必须参数：

filters：filter数目

kernel\_size：（方形）卷积窗口的高和宽。

可选参数：

strides：步幅

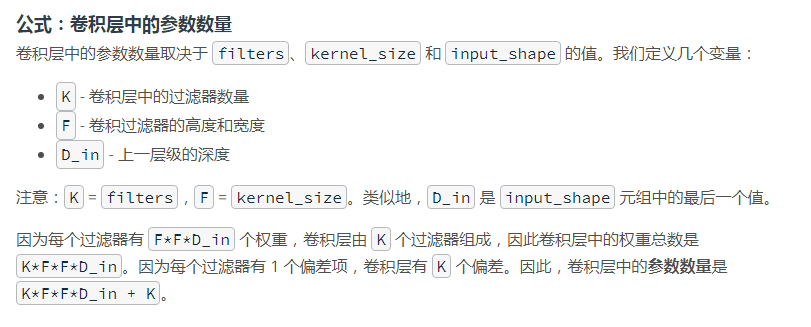
padding：填充规则

activation：激活函数

input\_shape：元组类型，如果卷积层是第一层级，则需要指定输入的高度，宽度和深度

注意：如果卷积层不是网络的第一个层级，请勿包含 input\_shape 参数。

卷积层的参数数量：



卷积层的深度就是过滤器的数量。



**3.4 CNN的设计**

PythonStudy\udacity\_test\machine\_learn\aind2-cnn-master\cifar10-classification

CNN模型：



模型参数为：

