计算机视觉综述（深度学习篇）

琚烈

712100，陕西，杨凌，西北农林科技大学，信息工程学院

1. CNN

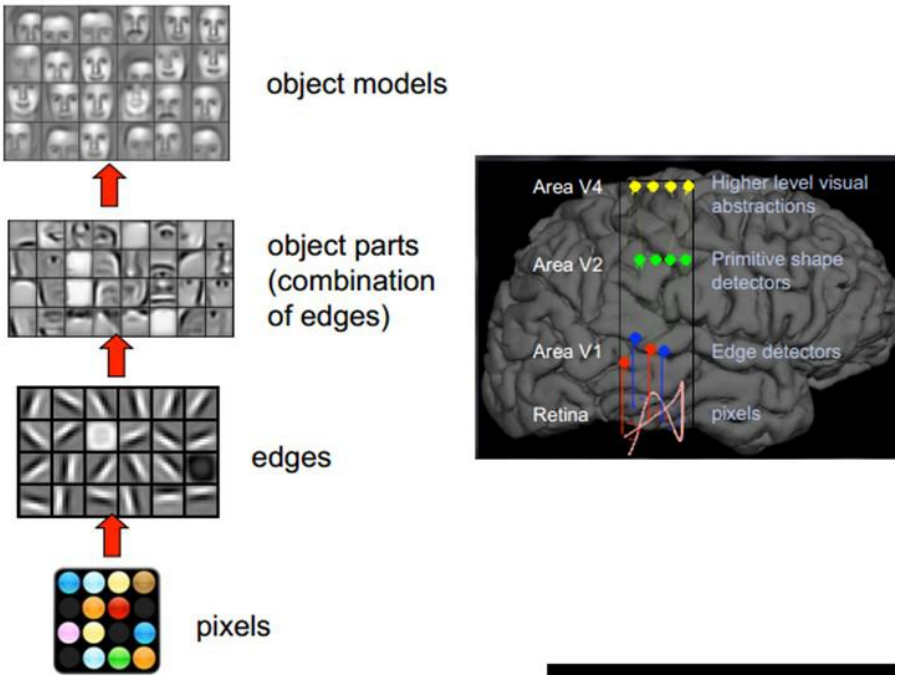
卷积神经网络（Convolutional Neural Networks, CNN）是一类包含卷积或相关计算且具有深度结构的前馈神经网络（Feedforward Neural Networks），是深度学习（deep learning）的代表算法之一。虽然它似乎既包含生物学概念，又充斥着数学理论，最后应用在计算机领域，但近几年的实验证明，它是最适用于计算机视觉领域的技术，从最初的可解释性差因此不被传统的机器学习学派认可，到如今的各种大赛上表现出的低错误率，CNN 已经成为了每个计算机视觉研究人员的必修课。

2. 人类视觉

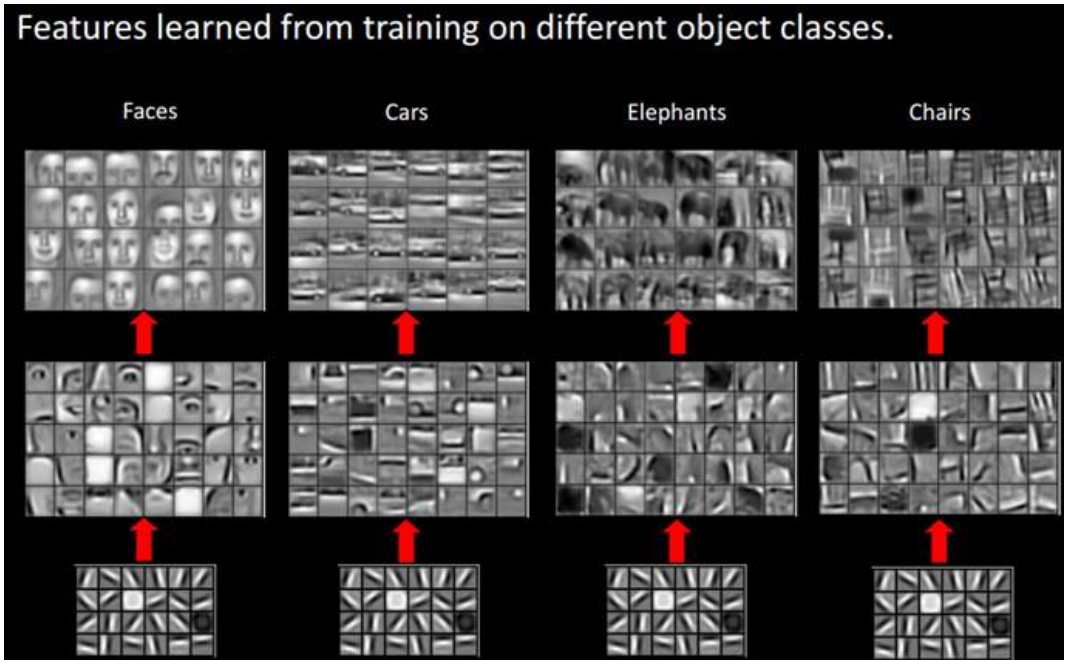
深度学习的许多研究成果，离不开对大脑认知原理的研究，尤其是视觉原理的研究。

1981 年的诺贝尔医学奖，颁发给了 David Hubel（出生于加拿大的美国神经生物学家） 和 TorstenWiesel，以及 Roger Sperry。前两位的主要贡献，是“发现了视觉系统的信息处理”，可视皮层是分级的。

人类的视觉原理如下：从原始信号摄入开始（瞳孔摄入像素 Pixels），接着做初步处理（大脑皮层某些细胞发现边缘和方向），然后抽象（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）。下面是人脑进行人脸识别的一个示例：



对于不同的物体，人类视觉也是通过这样逐层分级，来进行认知的：

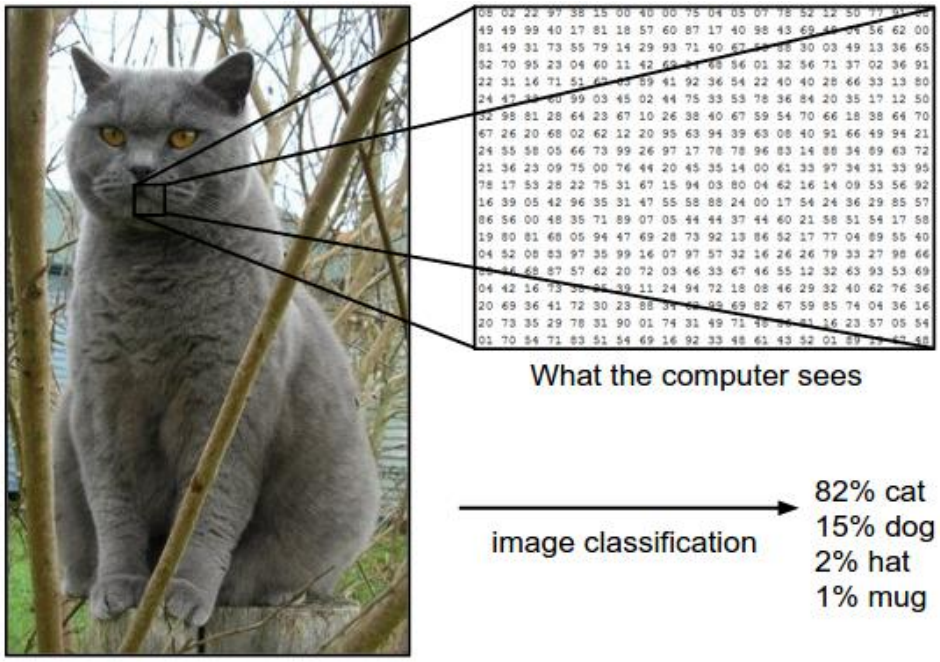


我们可以看到，在最底层特征基本上是类似的，就是各种边缘，越往上，越能提取出此类物体的一些特征（轮子、眼睛、躯干等），到最上层，不同的高级特征最终组合成相应的图像，从而能够让人类准确的区分不同的物体。

那么我们可以很自然的想到：可以不可以模仿人类大脑的这个特点，构造多层的神经网络，较低层的识别初级的图像特征，若干底层特征组成更上一层特征，最终通过多个层级的组合，最终在顶层做出分类呢？答案是肯定的，这也是许多深度学习算法（包括 CNN）的灵感来源。

图像分类主要指将输入图像进行硬分类或模糊分类（例如猫图、狗图等）。人类可以通过成长过程中的学习，凭借经验来分辨物体，但机器却无法拥有这种天赋，但通过“训练”，可以使其模拟我们学习分辨物体的过程，在一定程度上，机器的识别率还会超过人类。

当我们的电脑看到一副图片时，机器只是看到一个由像素值组成的矩阵，例如说 32*32*3，其中 32 表示其分辨率或图像大小，3 表示 RGB 三原色。为了把问题阐述清楚，我们这里定义一个 JPG 格式的彩色图像，大小为 480*480，那么表示的矩阵就是 480*480*3。矩阵里每一点取值范围 0-255，表示为该点的像素强度（灰度值）。在我们人类进行图像识别的时候，这些像素点一点儿意义也没有，它们只是作为机器进行图像识别的输入而已。机器的输出呢，可以是一组概率值，这组概率值表明了当前的图像是某一类图像的可能性有多大。（比如 0.8 可能性是猫，0.15 可能性是狗，0.05 可能性是鸟等）



我们知道了问题的输入和输出，现在要考虑该怎么解决它。我们想要计算机做的是分辨出所有给出的图中所具有的独特特征，例如说狗图或猫图的独特特征，这些特征是在某一分类图中一致，而跟其它类型图不同。这件事在我们自己的脑中同样也是自动完成的。例如，当我们看一副狗图时，我们能够根据图中物体的爪子或 4 条腿分辨其是小狗。类似地，计算机也可以通过寻找一些低等级特征，例如边缘或纹理等，并由此通过一系列卷积层来建立更抽象的概念，来实现分类识别。大体上这就是 CNN 所做的事。接下来让我们更具体地展开说。

3. 数据输入层

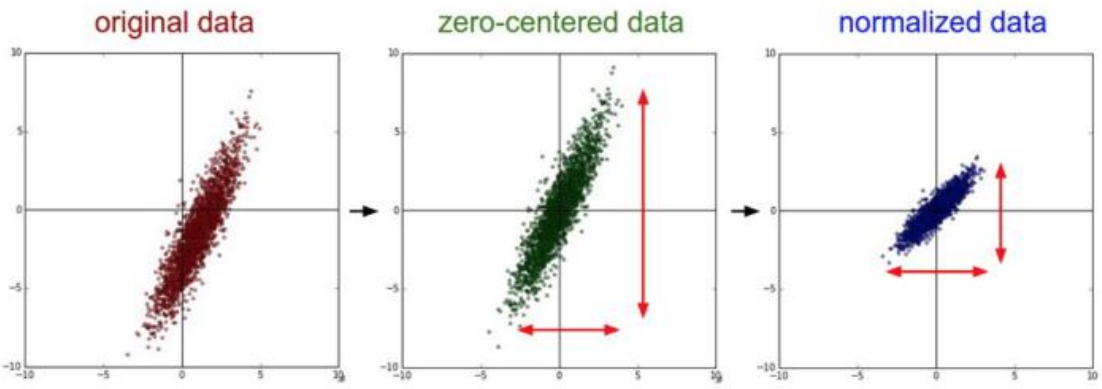
严格意义上来说，数据输入层并不是 CNN 的第一层，但对于深度学习模型的训练来说，一个合理的模型也包括它是怎样对数据进行预处理的，“数据决定了机器学习的上限，而算法只是尽可能逼近这个上限”。深度学习亦然，这也是为什么近些年特征工程大火，因为当人们发现无法在 CNN 的深度与设计继续做改进时，一般人都会将目光移向对数据的处理。

该层要做的处理主要是对原始图像数据进行预处理，其中包括：

• 去均值：把输入数据各个维度都中心化为 0，如下图所示，它对数据中每个独立特征减去平均值，从几何上可以理解为在每个维度上都将数据云的中心都迁移到原点。

• 归一化：幅度归一化到同样的范围，如下所示，即减少各维度数据取值范围的差异而带来的干扰，比如，我们有两个维度的特征 A 和 B，A 范围是 0 到 10，而 B 范围是 0 到 10000，如果直接使用这两个特征是有问题的，好的做法就是归一化，即 A 和 B 的数据都变为 0 到 1 的范围。

• PCA/白化：PCA 的思想是通过坐标轴转换，寻找数据分布的最优子空间，从而达到降维、去相关的目的。具体原理留给读者自己去查阅，在此不多加概述。



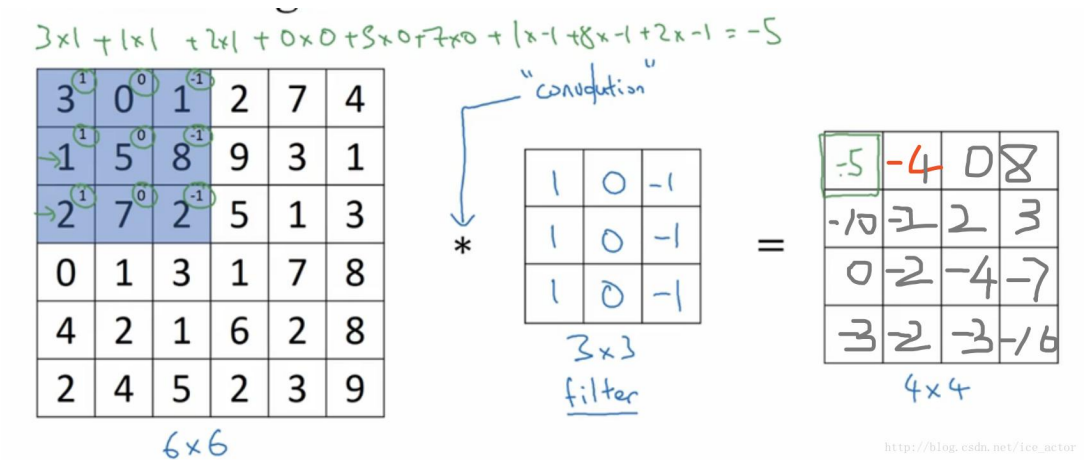
4. 卷积层（一）

CNN 的第一层通常都是卷积层。你必须记住的第一件事应当是卷积层(conv layer)的输入是什么。就像之前说的，输入是一个 32*32*3 的像素数列。

要解释这个卷积层，最好的办法就是想象一下下面场景：你举着手电筒将光束打在一幅图像的左上角。我们假定这个光束覆盖的范围是 5*5。那么现在，想象光束逐渐滑动平移，经过整幅图像。在机器学习的语言里，这个手电筒就被称作滤波器 filter（或神经元 neuron，或内核 kernel），而它照过的区域叫做感知区(receptive field)。这个滤波器也可以表示为一个数组（数组中的数字可称为加权 weight 或参数 parameter）。一个重要的点在于，滤波器的深度必须跟输入图像深度一致（才能保证计算不出错）。

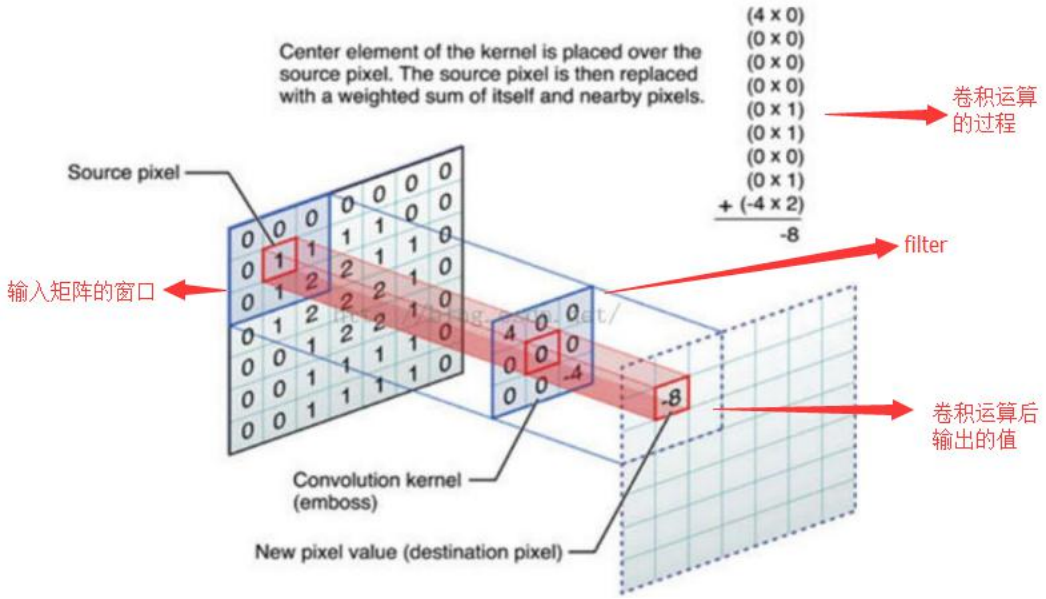
那么，这个滤波器的维度就变成了：5*5*3。现在，以这个滤波器最初的位置为例，它应当处于图像的左上角。随着滤波器的平移（或称卷积 convolving），经过整幅图像。它将自身数组中的值与图像对应位置的像素值进行相乘（即点乘），将点乘结果加起来（数学上一共有 75 次乘加），就有了一个数。记住，这个数只代表滤波器在图像左上角初始位置时的卷积值。现在，我们一边移动滤波器一边重复这个计算（顺序是从左到右，然后从上到下。下一步是往右移一个像素）。这样，图像上每个单独的像素位置都会产生一个滤波（卷积）后的数值。在遍历整个图像后，你会得到一个 28*28*1 的数列，我们称之为激活图 activation map 或特征图 feature map。28 的原因是 32-5+1=28。这样一共就有 784 个值。

如下是一个 6*6 的灰度图像，构造一个 3*3 的矩阵，在卷积神经网络中通常称之为 filter，对这个 6*6 的图像进行卷积运算，以左上角的-5 计算为例 ：



3*1+0*0+1*-1+1*1+5*0+8*-1+2*1+7*0+2*-1 = -5

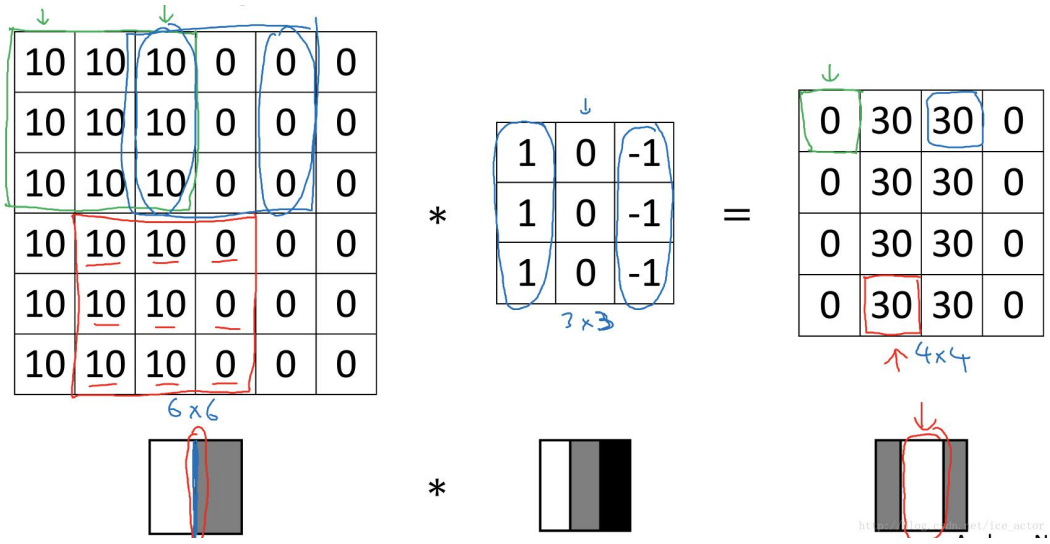
其它的以此类推，让过滤器在图像上逐步滑动，对整个图像进行卷积计算得到一幅 4*4 的图像。



很多人可能到这里就遇到了入门的难点，为什么要这么做？为什么这么做可以识别图像？这也是我们需要理解的最重要的部分，为什么 CNN 适用于计算机视觉。

首先让我们来看看吴恩达先生在 DeepLearning.AI 课程中给出的解释：

下图 0 表示图像暗色区域，10 为图像比较亮的区域，同样用一个 3*3 过滤器，对图像进行卷积，得到的图像中间亮，两边暗，亮色区域就对应图像边缘。



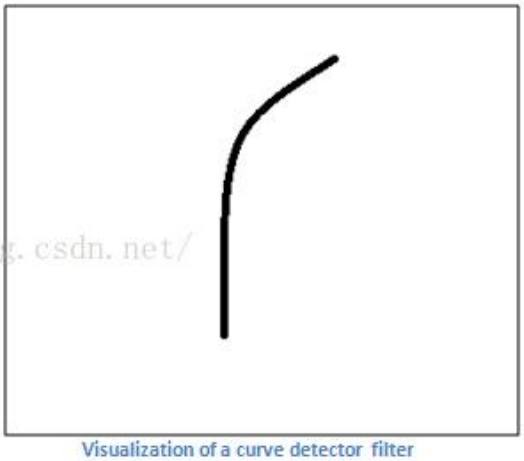
是不是还是不好理解，我们将给出另一个具体的例子。

让我们讨论一下，从更高的层面上，该如何看待这个卷积层所做的事。事实上，这

些滤波器可以看做是特征识别器。这里的特征，指的是那些直线边缘、颜色、纹理等。考虑所有图像中都共通的一些最简单的特征。让我们把第一个滤波器改成大小为 7*7*3 的纹理识别器。（在这个章节，为了降低复杂度，我们忽略滤波器与图像的深度，仅考虑其顶层的灰度数列及其计算）。作为一个纹理识别器，它的像素结构将会在纹理形状的对应区域内具有较高的数值。（记住，所有的滤波器都只是数值和数值的组合）

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

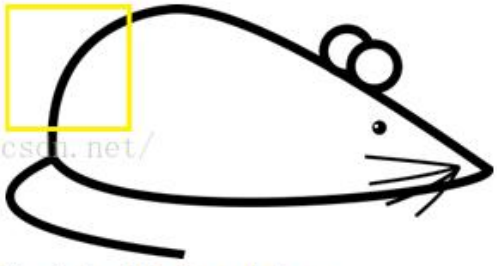
Pixel representation of filter



Visualization of a curve detector filter

现在，让我们回到数学上。当我们把滤波器放到图像左上角时，它开始进行 7*7 范围的内积计算。这里我们用给出的特征提取滤波器举一个例子，见下图。

Original image



Visualization of the filter on the image

记住，我们做的事是矩阵对应的像素值相乘再相加。

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

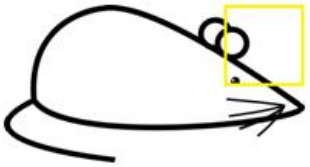
*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

基本上，在输入图像中，如果当前形状跟我们的滤波器大体相像，卷积后的计算结果将会是一个**很大的值**！那么让我们看看如果移动滤波器到其它位置会发生什么。



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

计算结果很小！这是因为滤波器和当前形状完全不同。记住，这个卷积层的输出是一个激活图层（activation map）。那么，针对这个单边缘滤波器卷积的简单例子来说，它的激活图层（activation map）将会展示出当前图像中最有可能是边缘特征的区域。在上图中，左上区域的卷积结果为 6600，这个大数值表明图中对应位置有这样的边缘特征使得滤波器被激活了。而右上区域的值为 0，表示当前区域没有任何能够激活滤波器的特征。（或更简单地说，图中该区域没有对应的图像特征）。记住，这只是一个滤波器。这只是一个检测竖直向右偏线状特征的滤波器。我们可以再添加其它滤波器用于检测竖直左偏或垂直特征等。滤波器越多，激活图层 activation map 的深度就越深，对于输入图像我们就能够获取到越多的信息。

声明：文中所述的滤波器是为了卷积计算上的简化而定义的。下图给出了一些用于第一卷积层的实际滤波器示例，但主要参数（原理）还是类似的。第一层的滤波器通过对输入图像的卷积和“激活”（或计算极值点），来寻找输入图像中特定的特征。

一个传统的 CNN 结构包含以下层次：
数据输入层/ Input layer
卷积计算层/ CONV layer
ReLU 激励层 / ReLU layer
池化层 / Pooling layer
全连接层 / FC layer

然而，最后一层是非常重要的一层，我们会提前做介绍。

现在让我们回过头来看看到目前已经学了哪些东西。我们讨论了第一卷积层的滤波器的设计与检测方法。它们用来检测边缘、纹理这类低阶特征。

正如大家想象的，要想预测图像中的事物，我们需要一个网络结构用于识别像手掌、爪子、耳朵这样的高阶特征。然后让我们想一下第一卷积层的输出应当是什么：一个 28*28*3 的数列结构（假设我们用的是三个 5*5*3 的滤波器）。

当我们进入第二个卷积层的时候，第一卷积层的输出就变成了第二卷积层的输入。这就比较抽象，比较难以想象了。要知道，当我们讨论第一层时，输入就是未经处理的原始图像；但是，当我们讨论第二卷积层时，输入已经变成了第一层的卷积结果，也就是一个激活图层 activation map。这个输入的每一层基本上描述了某些低阶特征在原始

图像中出现的位置。现在，当你把它输入给第二层，也就是再使用一系列滤波器对其卷积时，输出就应当是表示更高阶特征的激活图层，例如说半圆（结合了一个曲线和直线边缘）、方形（结合了一些直线边缘）等。当你将数据继续经过更多的卷积层时，你就会得到更加高阶特征的激活图层。

在整个网络的最后，你也许会用一些滤波器用以激活图像中的手写特征 handwriting、粉红色 物体 pink object 等。如果你想知道更多关于滤波器可视化的信息，Matt Zeiler 与 Rob Fergus 在他们的 ConvNets 模型与其研究论文 research paper 中有精彩的阐述。同样，在 YouTube 上，Jason Yosinski 提供了一个非常棒的讲解视频 video。另外一件有趣的事是，当你的网络结构越深，滤波器的感知区域范围就越大，这意味着它们在处理时，能够将对应原始图像中更大区域的信息都揽括在内（能够对更大的像素空间进行反应和感知）。

5. 全连接层

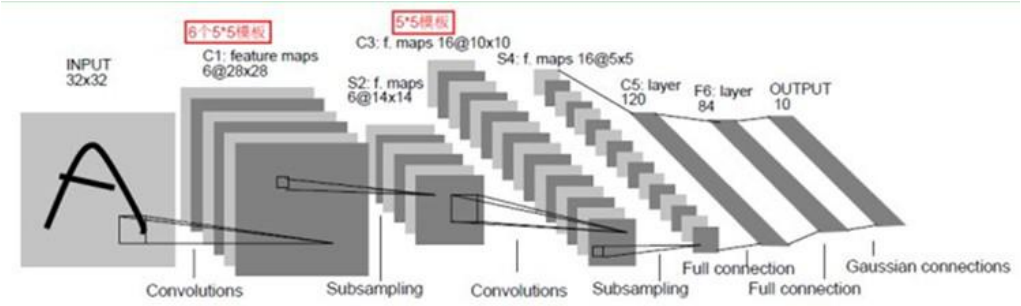
检测出高阶特征后，我们可以锦上添花地在网络结构的最后添加一个全连接层 fully connected layer。

全连接层输入一个数列（无论这个输入是来自卷积层 conv、线性整流 ReLU 层还是池化层 pool），输出一个 N 维向量，N 是由程序指定的**分类类别数量**。例如，对于一个数字分类程序，N 就应该取 10（0~9 共 10 个数字）。这个 N 维向量中的每一个数字表示被分到该类的几率。例如，还是针对数字分类程序的分类结果为[0 .1 .1 .75 0 0 0 0 0 .05]，这就表示这个输入的图像为 1 的概率有 10%,为 2 的概率 10%，为 3 的概率 75%，为 9 的概率 5%。（注：输出的表示方法不止这一种，这里只是展示了这种小数概率表示的 softmax 算法）。

全连接层的工作原理是，根据之前其它层的输出（表示为高阶特征的激活图层），检测**哪些特征与特定的类别相匹配**。

例如，程序若判定图像是一只狗，那么激活图层中表示狗的高阶特征，像是爪子、四条腿等特征将会具有很高的数值。再比如，程序若判定图像是一只鸟，那么激活图层中表示鸟的高阶特征，像是翅膀、鸟喙等就会具有很高的数值。

基本上，全连接层的工作就是寻找与特定类别匹配的高阶特征，并设定相应的权重值。这样当我们把之前层的结果通过权重进行计算后，就能够正确估计其属于某一类别的可能性了。



6. 训练

事实上，我们不需要把每个层都介绍完再说训练的问题，事实上，很多教程文章都

这么做了，但我认为这是十分愚蠢的！

训练其实是神经网络中最重要的部分，也许你在之前的阅读时会有很多问题。比如，第一卷积层的滤波器怎么知道该如何查找边缘和纹理？而全连通层又是怎样知道该如何查找激活图层的？每一层的滤波器都是怎样设定其值的？如果你对训练的过程有一个详细的了解，你在学习其他一些抽象层的时候，对一些问题就会有更好的体会。

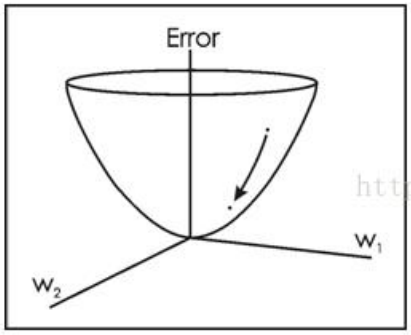
计算机在解决这些问题时主要依靠一个训练过程 training process，称作**反向传播算法 backpropagation（BP 算法）**。

讨论反向传播之前，我们必须先回过头来讲讲让神经网络（生物学意义上的）工作起来需要什么。当我们出生时，我们的大脑是一片空白的。我们并不知道什么是猫、狗、鸟儿。类似地，在 CNN 运行之前，滤波器的数值和权重等参数都是随机的。滤波器并不知道该如何寻找边缘和纹理。高阶层的滤波器同样也不知道该如何寻找爪子、鸟喙这类高阶特征。当我们逐渐长大，父母以及老师会利用照片、影像与之对应的标签来教会我们这些动物的辨别方法。这种利用标签的图片同样也是 CNN 所使用的训练过程。在深入研究之前，我们先简单地打个比方：我们有一个训练集，其中有几千张图片，绘制着猫、狗、鸟儿，同时每张图片上都有一个标明这是哪种动物的标签 label。好的，现在回到我们的反向传播算法。

反向传播算法可以分为 4 个部分：前向传播 forward pass，损失函数 loss function，反向传播 backward pass，权重更新 weight update。在前向传播的过程中，你将一个训练数据（32*32*3 的图像数列，表示一个数字）输入整个网络。由于所有权重或滤波器系数都是随机生成的，输出很可能是类似这样的结果[.1.1.1.1.1.1.1.1.1.1]，基本上这样的输出难以判别是一个有效分类的。系统以目前的参数/权值是很难寻找低阶特征，并且也很难以此进行可靠的分类的。于是进入算法的损失函数阶段。记得我们刚才使用的输入是训练数据，那么除了图像数列外还附带标签。比方说这个输入的训练数据是 3，那么标签就应当是[0 0 0 1 0 0 0 0 0 0]。通过其输出结果和标签的比较，就能够计算损失函数了。损失函数的定义方法很多，这里就用一个常见的均方差算法 MSE(mean squared error)，如下式：

$$E_{\text{total}} = \sum \frac{1}{2}(t\,\text{arg}\,et - output)^2$$

这里我们使用变量 L 保存这个损失函数的结果。可以想象，在训练刚开始时这个值会相当的大。现在让我们用直觉去想想，我们想要找到一个“点”使得预测的标签（也就是网络的输出）和训练标签相同（这意味着我们的网络预测的结果是正确的）。那么我们需要做的就是最小化损失函数 L。将其具象化，其实就是一个演算 calculus 优化的问题，我们想要找到哪些输入（在我们的系统中就是权值）更直接地影响到损失 L，或者说误差。

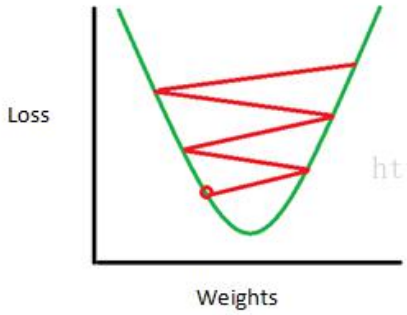


One way of visualizing this idea of minimizing the loss is to consider a 3-D graph where the weights of the neural net (there are obviously more than 2 weights, but let's go for simplicity) are the independent variables and the dependent variable is the loss. The task of minimizing the loss involves trying to adjust the weights so that the loss decreases. In visual terms, we want to get to the lowest point in our bowl shaped object. To do this, we have to take a derivative of the loss (visual terms: calculate the slope in every direction) with respect to the weights.

上图表示，针对这个问题具象化的一个方法是将其放在由一个误差（损失相关）坐标轴、两个权值坐标轴组成的 3D 图中。（当然大部分神经网络的权值显然大于 2，这只是为了简化）。想要最小化误差值就需要不断调节权值 w1 和 w2。用图上的话说，我们要找到这个碗状曲面的 z 轴最低点。要做到这一点，就需要求出误差在各个方向（权值）上的导数（斜率）。 在数学上，这等价于针对某一层的权重 W，计算导数 dL/dW。那么现在，我们想要做的就是对网络进行一次反向传播 backward pass，以判断哪些权重对损失 L 有较大的影响，并且调整这些权重以减小损失。一旦导数计算出来之后，我们就可以进行最后一个步骤：更新权重。我们将所有滤波器的权重进行更新使得它们随着其梯度方向的变化而改变（change in the direction of the gradient 梯度下降法？）

$$\mathrm{w} = \mathrm{w}_{\mathrm{i}} - \eta \frac{\mathrm{d}L}{\mathrm{d}W}$$

学习速率 learning rate 是由程序员所指定的参数。高学习速率表明在进行权重更新的跨度大，这样模型就能更快地汇集到最优的权重集上面来。但是，过高的学习速率有可能导致跨度过大而难以精确地收敛到最优解上。



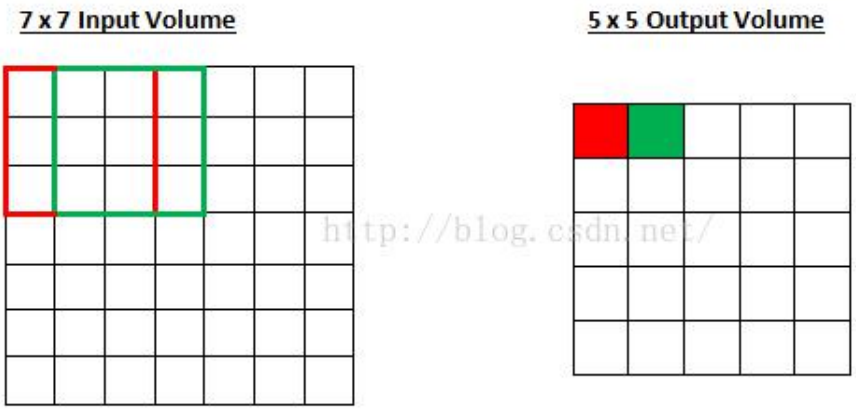
Consequence of a high learning rate where the jumps are too large and we are not able to minimize the loss.

这整套流程：前向传递 forward pass、损失函数 loss function、后向传递 backward pass、参数更新 parameter update 合起来称为一代 epoch。针对每个训练图像，程序会重复进行多代 epoch 的训练。每次迭代结束后，我们都希望能够让网络通过良好地训练，正确地调节其每层的权重。

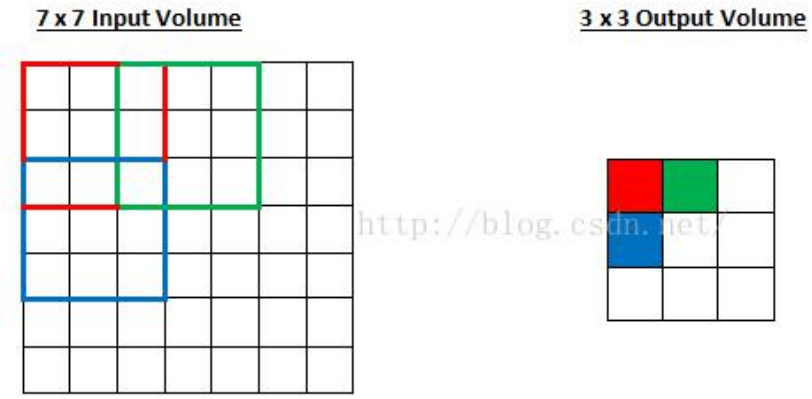
7. 卷积层（二）

好的，让我们回忆一下我们的老朋友卷积层。还记得那些滤波器、感知区、卷积吗？现在让我们介绍两个卷积层中的重要参数。步长 stride 和填充 padding。

步长 stride 主要用于控制滤波器在输入图像上的卷积行为。如下图例子所示，滤波器在图像上的卷积是每次卷积计算后平移一定距离再次计算。这个距离就是通过步长 stride 来进行控制的。在这个例子中，步长 stride 设为 1。通常来说，步长 stride 参数的选择需要考虑到这个步数能够被整除。让我们看一下图中的这个例子：设输入为 7*7 的图像，滤波器尺寸为 3*3（同样忽略了第三维），设定步长为 1，那么输出就应该是下图右边的 5*5 图像。



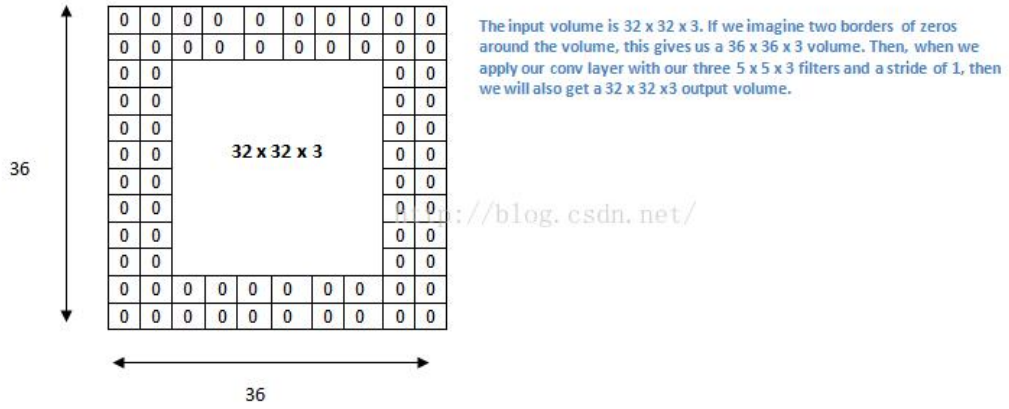
还是老样子，不是吗？想想如果我们把步长 stride 设为 2 的情形。



那么，感知区每次就会平移两个单位（像素），那么输出自然就相应的缩小为 3*3 了。如果我们想要把步长 stride 设为 3，那么在进行卷积时就会有空间上的问题了，同时也难以保证感知区在平移时是否还处于图像内（译者：原因是步数不能整除？）。通常，在设计卷积层时，步长 stride 越大，那么感知区的重叠就越小，同样地输出的图像也越小。

现在，让我们看一下填充 padding 参数。先想象一下这样的场景。当你使用 3 个 5*5*3 的滤波器对一个 32*32*3 的输入图像滤波，那么输出图像应当是 28*28*3。注意空间尺寸变小了。然后我们把这个图像再次放入卷积层，会发现尺寸更小了。事实上，在整个网络的开始几层，我们并不希望尺寸缩水得这么快。我们希望能够尽量保留下这些输入图像的信息，才能够比较好地提取底层特征。也就是说，我们想要保证输出仍然是

32*32*3。为了实现这个目的，我们在卷积层上引入了尺寸为 2 的零填充 zero padding。它能够在输入图像的边界上形成一层宽度为 2，值为 0 的边界。那么输入图像就变成了 36*36*3。如下图所示。



若设定步长 stride 为 1，同时零填充 zero padding 的尺寸为：

$$ZeroPadding = \frac{(K-1)}{2}$$

式中 K 为滤波器尺寸。那么输入和输出图像的尺寸就会永远保持一致。

卷积层的输出图像的尺寸可以通过下面公式得出：

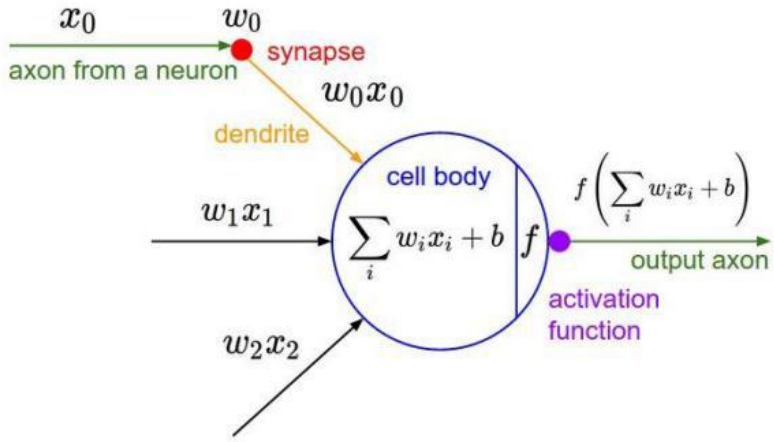
$$O = \frac{(W - K + 2P)}{S} + 1$$

式中 O 是输出宽/高，W 是输入宽/高，K 为滤波器尺寸，P 为填充 padding 尺寸，S 为步长 stride。

我们该怎样确定在一个系统中，卷积层的数量、滤波器的尺寸、或是步长 stride、填充 padding 这些参数呢？这些可不是细枝末节的问题，另外这些问题也没有一个放之四海而皆准的答案。这是因为系统参数的设定主要取决于你的数据类型。数据的大小、图像的复杂度、图像处理的目的和方式等，都会随着处理目的的不同而发生变化。选择超参数的正确做法是，在检视数据集时通过抽象概念将数据/图像以合适的尺度正确组合起来。

8. 激励层

激活函数（Activation functions）对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。它们将非线性特性引入到我们的网络中。其主要目的是将神经网络模型中一个节点的输入信号转换成一个输出信号。该输出信号现在被用作堆叠中下一个层的输入。



如果我们不运用激活函数的话，则输出信号将仅仅是一个简单的线性函数。线性函数一个一级多项式。现如今，线性方程是很容易解决的，但是它们的复杂性有限，并且从数据中学习复杂函数映射的能力更小。一个没有激活函数的神经网络将只不过是一个线性回归模型（Linear regression Model）罢了，它功率有限，并且大多数情况下执行得并不好。我们希望我们的神经网络不仅仅可以学习和计算线性函数，而且还要比这复杂得多。同样是因为没有激活函数，我们的神经网络将无法学习和模拟其他复杂类型的数据，例如图像、视频、音频、语音等。这就是为什么我们要使用人工神经网络技术，诸如深度学习（Deep learning），来理解一些复杂的事情，一些相互之间具有很多隐藏层的非线性问题，而这也可以帮助我们了解复杂的数据。

那么为什么我们需要非线性函数？

非线性函数是那些一级以上的函数，而且当绘制非线性函数时它们具有曲率。现在我们需要一个可以学习和表示几乎任何东西的神经网络模型，以及可以将输入映射到输出的任意复杂函数。神经网络被认为是通用函数近似器（Universal Function Approximators）。这意味着他们可以计算和学习任何函数。几乎我们可以想到的任何过程都可以表示为神经网络中的函数计算。

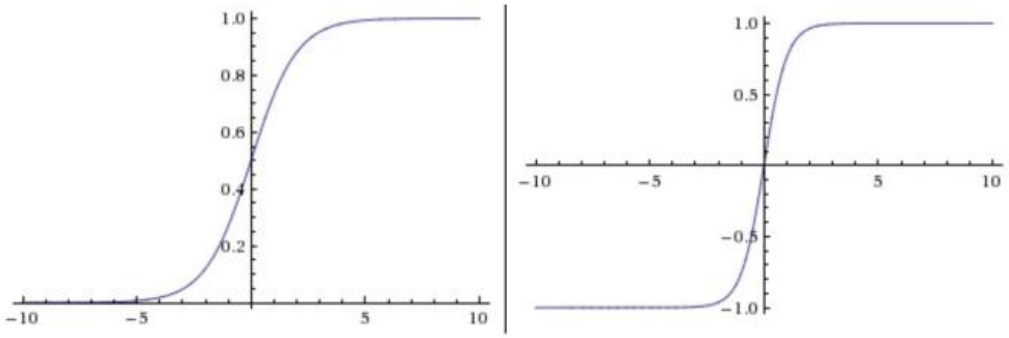
而这一切都归结于这一点，我们需要应用激活函数 f（x），以便使网络更加强大，增加它的能力，使它可以学习复杂的事物，复杂的表单数据，以及表示输入输出之间非线性的复杂的任意函数映射。因此，使用非线性激活函数，我们便能够从输入输出之间生成非线性映射。

激活函数的另一个重要特征是：它应该是可以区分的。我们需要这样做，以便在网络中向后推进以计算相对于权重的误差（丢失）梯度时执行反向优化策略，然后相应地使用梯度下降或任何其他优化技术优化权重以减少误差。

只要永远记住要做：

“输入时间权重，添加偏差和激活函数”。

每个激活函数（或非线性函数）的输入都是一个数字，然后对其进行某种固定的数学操作。下面是在实践中可能遇到的几种激活函数：



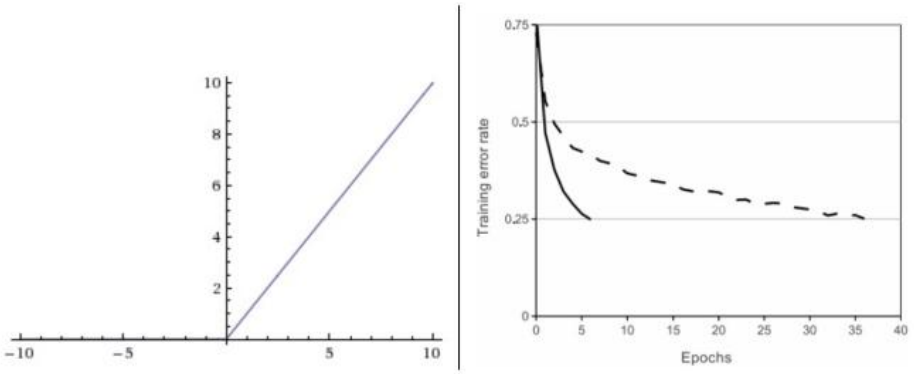
左边是 Sigmoid 非线性函数，将实数压缩到[0,1]之间。右边是 tanh 函数，将实数压缩到[-1,1]。

sigmoid 非线性函数的数学公式是 $\sigma(x) = 1/(1+e^{-x})$，函数图像如上图的左边所示。在前一节中已经提到过，它输入实数值并将其“挤压”到 0 到 1 范围内。更具体地说，很大的负数变成 0，很大的正数变成 1。在历史上，sigmoid 函数非常常用，这是因为它对于神经元的激活频率有良好的解释：从完全不激活（0）到在求和后的最大频率处的完全饱和（saturated）的激活（1）。然而现在 sigmoid 函数已经不太受欢迎，实际很少使用了，这是因为它有两个主要缺点：

1.Sigmoid 函数饱和使梯度消失。sigmoid 神经元有一个不好的特性，就是当神经元的激活在接近 0 或 1 处时会饱和：在这些区域，梯度几乎为 0。回忆一下，在反向传播的时候，这个（局部）梯度将会与整个损失函数关于该门单元输出的梯度相乘。因此，如果局部梯度非常小，那么相乘的结果也会接近零，这会有效地“杀死”梯度，几乎就有没有信号通过神经元传到权重再到数据了。还有，为了防止饱和，必须对于权重矩阵初始化特别留意。比如，如果初始化权重过大，那么大多数神经元将会饱和，导致网络就几乎不学习了。

2.Sigmoid 函数的输出不是零中心的。这个性质并不是我们想要的，因为在神经网络后面层中的神经元得到的数据将不是零中心的。这一情况将影响梯度下降的运作，因为如果输入神经元的数据总是正数（比如在 $\mathrm{f} = \mathrm{w}^{\mathrm{T}}x + b$ 中每个元素都 $\mathrm{x} > 0$），那么关于 w 的梯度在反向传播的过程中，将会要么全部是正数，要么全部是负数（具体依整个表达式 f 而定）。这将会导致梯度下降权重更新时出现 z 字型的下降。然而，可以看到整个批量的数据的梯度被加起来后，对于权重的最终更新将会有不同的正负，这样就从一定程度上减轻了这个问题。因此，该问题相对于上面的神经元饱和问题来说只是个小麻烦，没有那么严重。

Tanh 非线性函数图像如上图右边所示。它将实数值压缩到[-1,1]之间。和 sigmoid 神经元一样，它也存在饱和问题，但是和 sigmoid 神经元不同的是，它的输出是零中心的。因此，在实际操作中，tanh 非线性函数比 sigmoid 非线性函数更受欢迎。注意 tanh 神经元是一个简单放大的 sigmoid 神经元，具体说来就是：$\tanh(x) = 2\sigma(2x) - 1$。



左边是 ReLU（校正线性单元：Rectified Linear Unit）激活函数，当 x=0 时函数值为 0。当 x>0 函数的斜率为 1。右边是从 Krizhevsky 等的论文中截取的图表，指明使用 ReLU 比使用 tanh 的收敛快 6 倍。

ReLU。在近些年 ReLU 变得非常流行。它的函数公式是 $\mathrm{f}(x)=\max(0,x)$。换句话说，这个激活函数就是一个关于 0 的阈值（如上图左侧）。使用 ReLU 有以下一些优缺点：

优点：相较于 sigmoid 和 tanh 函数，ReLU 对于随机梯度下降的收敛有巨大的加速作用（ Krizhevsky 等的论文指出有 6 倍之多）。据称这是由它的线性，非饱和的公式导致的。

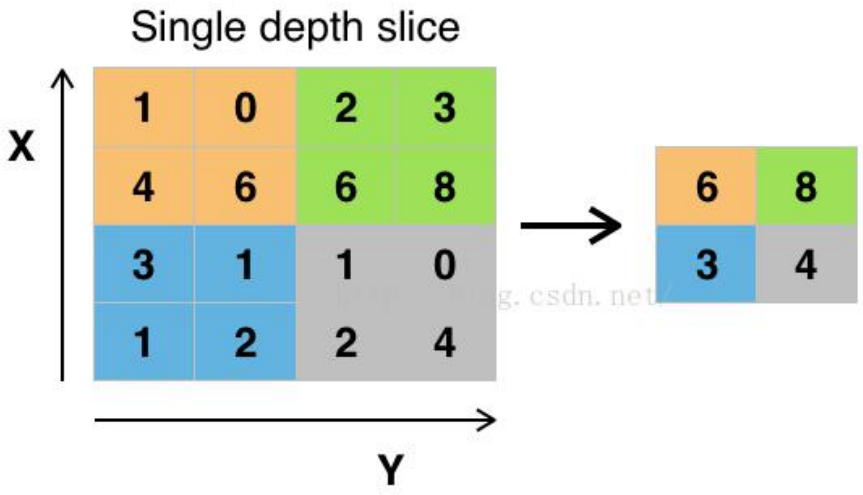
优点：sigmoid 和 tanh 神经元含有指数运算等耗费计算资源的操作，而 ReLU 可以简单地通过对一个矩阵进行阈值计算得到。

缺点：在训练的时候，ReLU 单元比较脆弱并且可能“死掉”。举例来说，当一个很大的梯度流过 ReLU 的神经元的时候，可能会导致梯度更新到一种特别的状态，在这种状态下神经元将无法被其他任何数据点再次激活。如果这种情况发生，那么从此所以流过这个神经元的梯度将都变成 0。也就是说，这个 ReLU 单元在训练中将不可逆转的死亡，因为这导致了数据多样化的丢失。例如，如果学习率设置得太高，可能会发现网络中 40%的神经元都会死掉（在整个训练集中这些神经元都不会被激活）。通过合理设置学习率，这种情况的发生概率会降低。

激励层的实践经验：
①不要用 sigmoid！不要用 sigmoid！不要用 sigmoid！
② 首先试 RELU，因为快，但要小心点
③ 如果 2 失效，请用 Leaky ReLU 或者 Maxout
④ 某些情况下 tanh 倒是有不错的结果，但是很少

9. 池化层

通过线性整流层 ReLU 处理后，研究人员可能会在其后添加一个池化 pooling 层，也叫下采样 downsampling 层。这同样有许多可选的方法，其中最常用的是最大池化 maxpooling 方法。这种方法定义一个最大值滤波器（通常 2*2）以及对应长度的步长 stride，然后对输入图像进行滤波输出滤波器经过的每个子区域的最大值。如下图。



Example of Maxpool with a 2x2 filter and a stride of 2

池化层也可以用均值或 L2 范数（L2-norm，欧氏距离）来计算。池化层的直观意义在于，从输入图像中获取到某个特征（会有较高的激活值 activation value），它与其它特征的相对位置比其自身的绝对位置更加重要。从图中不难看出，池化层大大降低了输入图像的空间维度（宽和高，不包括深度）。这就达到了两个目的。首先，参数/权重的个数降低为原来的 1/4，计算量相应减少了。第二，它能够防止过拟合 overfitting。过拟合表示为由于过度的调优导致模型过于执着满足样本的特征（可能有些是局部特征）而失去了泛用性，从而难以识别其它数据。过拟合 overfitting 的一个征兆就是，模型在训练集上能获得 99%或 100%的精确度但在测试数据上仅有 50%。

10. Dropout 层

事实上，不是所有网络都有 Dropout 层，但为了模型的健壮性，却变得越来越不可缺少了。

开篇明义，dropout 是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。注意是暂时，对于随机梯度下降来说，由于是随机丢弃，故而每一个 mini-batch 都在训练不同的网络。

dropout 是 CNN 中防止过拟合提高效果的一个大杀器，但对于其为何有效，却众说纷纭。在下读到两篇代表性的论文，代表两种不同的观点，特此分享给大家。

组合派

参考文献中第一篇中的观点，Hinton 老大爷提出来的，关于 Hinton 在深度学习界的地位我就不再赘述了，光是这地位，估计这一派的观点就是“武当少林”了。注意，派名是我自己起的，各位勿笑。

观点

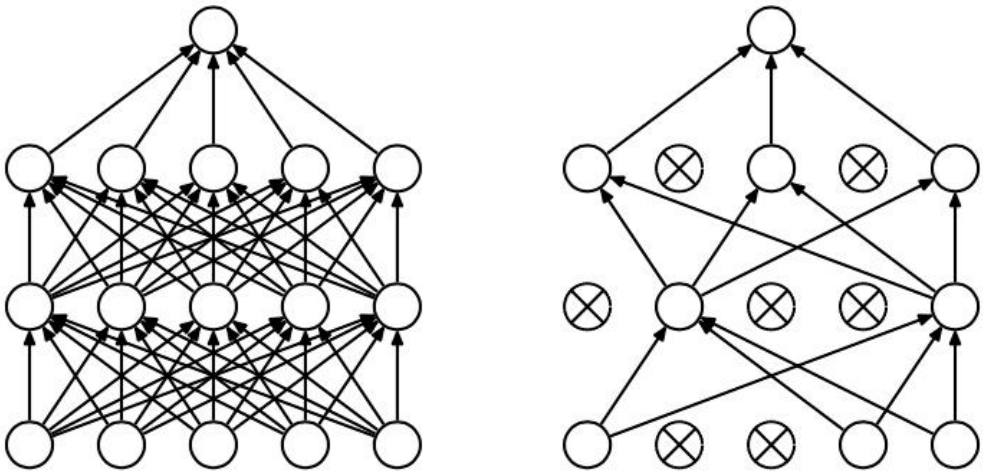
该论文从神经网络的难题出发，一步一步引出 dropout 为何有效的解释。大规模的神经网络有两个缺点：

1. 费时

2. 容易过拟合

这两个缺点真是抱在深度学习大腿上的两个大包袱，一左一右，相得益彰，额不，臭气相投。过拟合是很多机器学习的通病，过拟合了，得到的模型基本就废了。而为了解决过拟合问题，一般会采用 ensemble 方法，即训练多个模型做组合，此时，费时就成为一个大问题，不仅训练起来费时，测试起来多个模型也很费时。总之，几乎形成了一个死锁。

Dropout 的出现很好的可以解决这个问题，每次做完 dropout，相当于从原始的网络中找到一个更瘦的网络，如下图所示：



因而，对于一个有 N 个节点的神经网络，有了 dropout 后，就可以看做是 2n 个模型的集合了，但此时要训练的参数数目却是不变的，这就解脱了费时的问题。

动机论

虽然直观上看 dropout 是 ensemble 在分类性能上的一个近似,然而实际中,dropout 毕竟还是在一个神经网络上进行的，只训练出了一套模型参数。那么他到底是因何而有效呢？这就要从动机上进行分析了。论文中作者对 dropout 的动机做了一个十分精彩的类比：

在自然界中，在中大型动物中，一般是有性繁殖，有性繁殖是指后代的基因从父母两方各继承一半。但是从直观上看，似乎无性繁殖更加合理，因为无性繁殖可以保留大段大段的优秀基因。而有性繁殖则将基因随机拆了又拆，破坏了大段基因的联合适应性。

但是自然选择中毕竟没有选择无性繁殖，而选择了有性繁殖，须知物竞天择，适者生存。我们先做一个假设，那就是基因的力量在于混合的能力而非单个基因的能力。不管是有性繁殖还是无性繁殖都得遵循这个假设。为了证明有性繁殖的强大，我们先看一个概率学小知识。

比如要搞一次恐怖袭击，两种方式：

- 集中 50 人，让这 50 个人密切精准分工，搞一次大爆破。
- 将 50 人分成 10 组，每组 5 人，分头行事，去随便什么地方搞点动作，成功一次就算。

哪一个成功的概率比较大？ 显然是后者。因为将一个大团队作战变成了游击战。

那么，类比过来，有性繁殖的方式不仅仅可以将优秀的基因传下来，还可以降低基

因之间的联合适应性，使得复杂的大段大段基因联合适应性变成比较小的一个一个小段基因的联合适应性。

dropout 也能达到同样的效果，它强迫一个神经单元，和随机挑选出来的其他神经单元共同工作，达到好的效果。消除减弱了神经元节点间的联合适应性，增强了泛化能力。

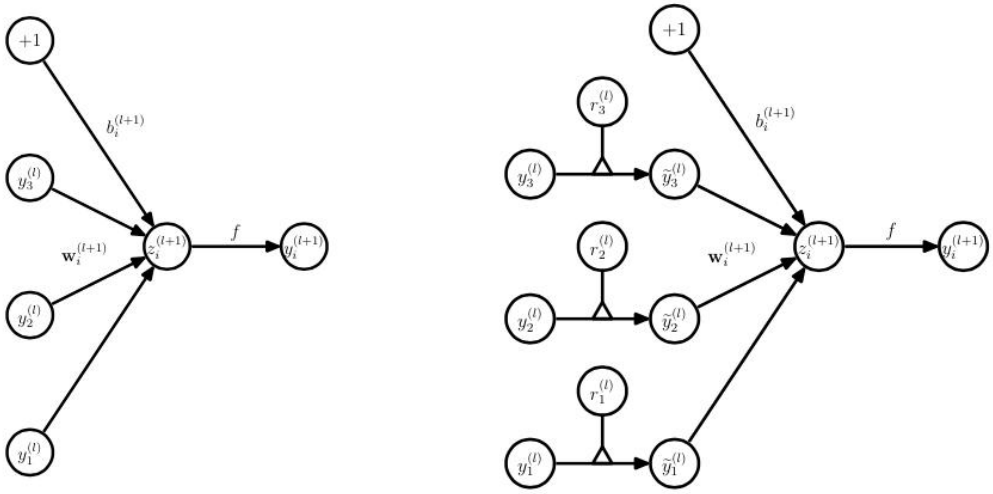
个人补充一点：那就是植物和微生物大多采用无性繁殖，因为他们的生存环境的变化很小，因而不需要太强的适应新环境的能力，所以保留大段大段优秀的基因适应当前环境就足够了。而高等动物却不一样，要准备随时适应新的环境，因而将基因之间的联合适应性变成一个一个小的，更能提高生存的概率。

dropout 带来的模型的变化

而为了达到 ensemble 的特性，有了 dropout 后，神经网络的训练和预测就会发生一些变化。

1. 训练层面

无可避免的，训练网络的每个单元要添加一道概率流程。



(a) Standard network　　(b) Dropout network

对应的公式变化如下如下：

• 没有 dropout 的神经网络

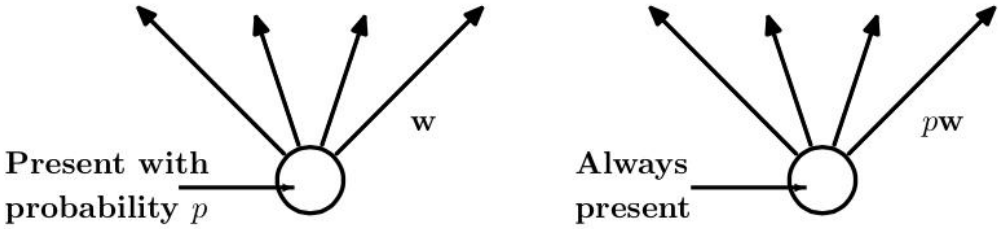
$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

• 有 dropout 的神经网络

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^{l} + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

2.测试层面

预测的时候，每一个单元的参数要预乘以 p。

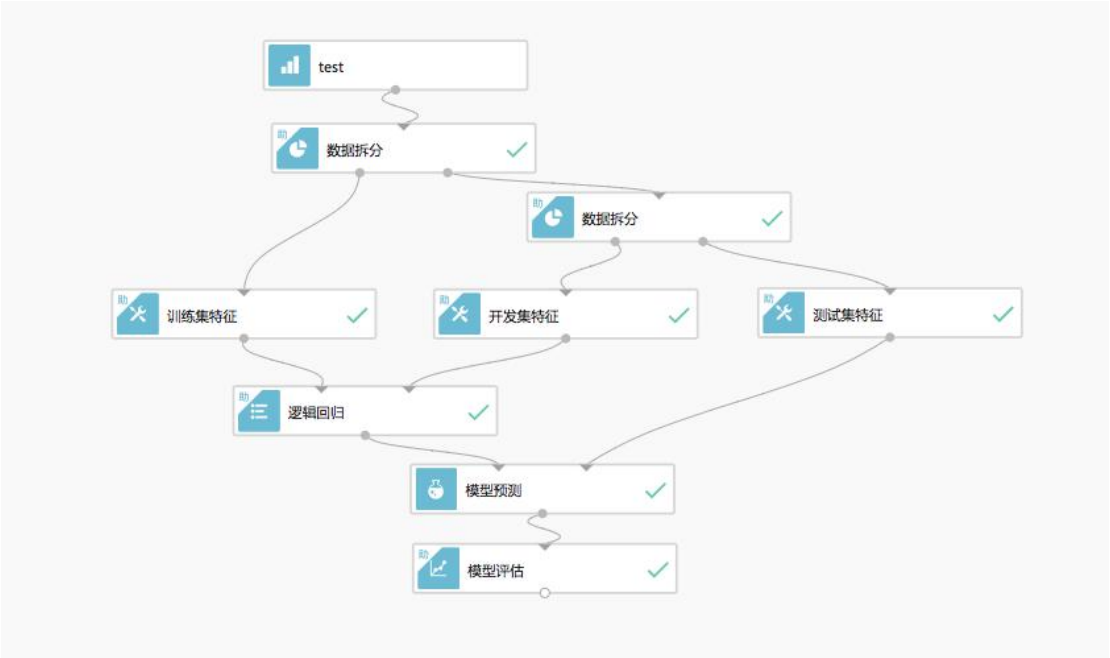


(a) At training time　　(b) At test time

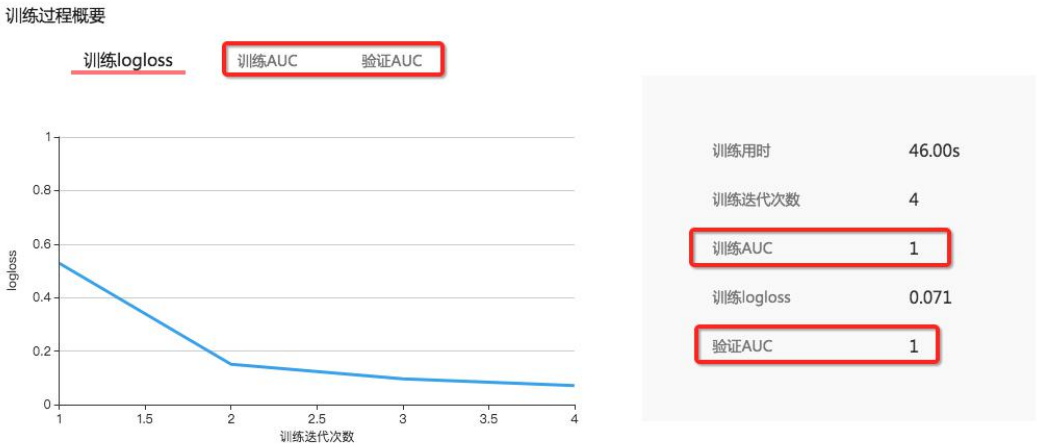
如何防止过拟合一直是深度学习中很头疼的问题，如果读者感兴趣，可以参阅参考文献中关于 Dropout 层更深层次的讨论。

11. 测试

我们在模型训练的时候通常会将我们所得到的数据分成三部分。分别是 training set， dev set(也叫 validation set)和 test set。在我们的模型调研过程中，他们分别起着不同的作用。training set 用来训练模型， dev set 用来统计单一评估指标，调节参数，选择算法。test set 则用来在最后整体评估模型的性能。

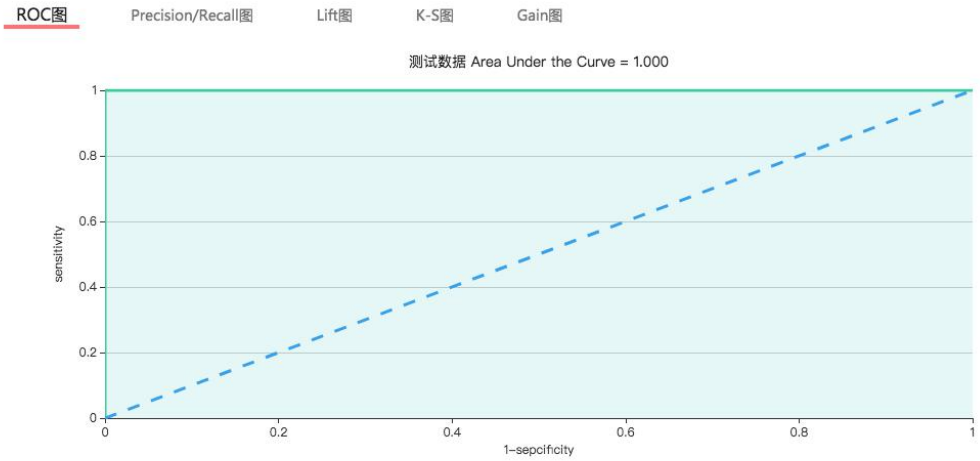


如上图，假设我们有一份数据，会将它按一定的规则进行拆分。其中 training set 和 dev set 分别输入到了逻辑回归算法中，而 test set 则是在模型训练结束后，被输入到模型中评估结果。我们可以根据 report 来看一下他们各自的作用。尤其是 dev set 和 test set，在很多文章中对他们的介绍很模棱两可，让人搞不明白他们之间到底有什么区别。给我的感觉就是写文章的人也不懂，在那里随便写写罢了。我们先看 training set 和 dev set，因为他们都被输入到了模型训练算法中。



上图是模型训练的 report。 我们可以从中看到 training set 和 dev set(图中叫验证集) 的 auc 指标。这里便引入了 dev set 的作用，training set 很好理解，训练模型用的。而 dev set 的作用就是在这里很方便的评估算法的单一评估指标，在这里也就是 auc，通过这个指标在 training set 和 dev set 上的对比我们可以用来调整算法参数。例如，如果 training set 的 auc 是 0.9 而 dev set 的 auc 确是 0.6. 根据之前学到的我们很可能发生了过拟合的情况。需要减少迭代次数或者设置 L2 正则来减少拟合。 又或者说我们发现训练 AUC 和验证 AUC 的数值都很低，例如只有 0.6，而 loss(损失函数) 仍然没有收敛，我们可能要增加迭代次数或者是扩大数据集或者改变特征。这就是 dev set 的作用，它是与 training set 一起被输入到模型算法中但又不参与模型训练，我们

一边训练一边在 dev set 上看 auc。这样相比于在 test set 上进行评估可以节省大量的时间。 我们在最开始的图中可以看到，使用 test set 会增加额外的步骤。在我们调参阶段完全没有必要。只有当我们觉得当前的模型在 dev set 的效果已经差不多的时候，才会使用 test set 进一步验证模型性能。那么 dev set 和 training set 有什么区别么？看上去好像都是用来评估模型好坏的，特意在 test set 上进行验证又有什么用呢？ 因为 test set 能够提供更多的评估模型的指标。我们之前说过评估一个分类器有混淆矩阵，有 ROC，有召回，精准，F1 Scroe 等。这些都是在 dev set 上不做统计的，dev set 上只统单一评估指标，也就是 AUC。例如下面是 test set 的评估报告：



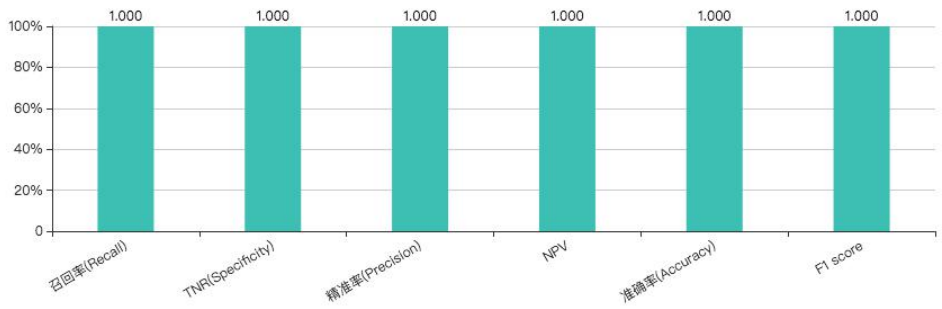
评估指标

测试集AUC	1.000	测试数据样本数	13

测试集混淆矩阵和评估指标

正负例判定阈值 0.50

	模型预测（正）	模型预测（负）	总数
实际值（正）	2 15.38%	0 0	2 15.38%
实际值（负）	0 0	11 84.62%	11 84.62%
总数	2 15.38%	11 84.62%	



总结一下：

我们用 training set 做训练，dev set 来初步评估结果，这里的评估结果是单一评估指标，也就是对模型来说最重要的一个指标，在这里我们使用 AUC。这么做的优点是 dev set 跟随 training set 一起被输入到模型算法中但又不参与模型训练。只是用来快速评估 AUC 的， 在调参阶段我们会不停的改变参数值来调整模型，而 dev set 就能帮助我们快速的查看结果。 相反的 test set 的作用并不是快速查看结果的，它提供一个模型的完整评估报告。可以更好的从多个维度评价模型的性能，但缺点是要做很多其他的操作,比较费时。我们一般在 dev set 上把参数调整的差不多后,才会使用到 test set。

数据拆分的规则：

数据量的拆分

现在我们知道要把数据拆分成 3 份，那么我们需要有一定的规则拆分数据集。 在老的规则里，也就是大数据时代之前。可能的规则是数据按照 8:1:1 来拆分。dev set 和 test set 不能太小，小了不能很好的评估模型效果。但这个规则是建立在数据集比较小的情况下，例如只有 1w 条数据或者更小。在大数据时代小，我们一般面临的都是百万级甚至亿级的数据量。这时候的拆分规则就会变化一下。比如我们有一百万行的数据，那么这时候的拆分就不能按照 8:1:1 来了。可能是 98%:1%:1%的规则比较合适，因为百万级的数据即便只有 1%也有一万行，用来评估模型效果已经有了比较不错的效果。把更多的数据留给 training set 来获得更好的模型是比较好的选择。

数据分布的拆分

我们的数据都是有业务含义的。所以我们拆分的时候出了要考虑数据量以外可能还要考虑一些场景。首先说 dev set 和 test set，他们都是用来评估模型性能的，所以一定要保证他们的数据处于同一分布。举个例子说，如果我们统计的是 8 个国家的数据，如果我们选择 4 个国家的数据作为 dev set，另外 4 个国家作为 test set 可以么？这是明显不可以的，他们的数据处于不同的分布上。 如果我们在这样的 dev set 上进行调参并训练模型。 那到了 test set 上的时候你会发现模型效果极其的差。所以我们要牢记于心的规则就是 dev set 和 test set 一定要处于同一分布。在这里例子，我们要让 dev set 和 test set 随机的从所有国家中平均的抽取数据。 例如说每个国家有 100 份数据。那么我们要保证 dev set 和 test set 能在每个国家中都随机抽取 50 份数据。

training set 的抽取

那么 traning set 呢？ 理论上 traning set 最好也是要跟 dev set 和 test set 处于同一分布的。但有些时候我们不能这么做，比如说我们的数据是有时序性的。例如预测点击率这种场景中，数据的时序性是很重要的。因为新闻，视频或者音频等资源的时效性比较强。这时候如果让 traning set 与 dev set，test set 处于同一分布就会有问题。 例如如果我们从数据中随机的拆分就会有问题。这样拆分甚至会出现用未来的数据预测过去的行为。 而我们希望的是用过去的数据预测未来的数据。所以这时候我们应该做的是首先按时间把数据分成两类----过去的数据和未来的数据， 或者说是按一个时间点拆分成两种数据，处于这个时间点之前的数据作为 traning set。这个时间点之后的数据作为 dev set 和 test set。这时候我们再按照上面的规则，保证 dev set 和 test set 处于同一分布就好了。所以关于 training set 的选择要根据数据的业务场景来定。

说到底：

验证集是一定需要的；

如果验证集具有足够泛化代表性，是不需要再整出什么测试集的；

整个测试集往往就是为了在验证集只是非训练集一个小子集的情况下，好奇一下那个靠训练集（训练）和验证集（调参）多次接力训练出来的模型是不是具有了泛化性能，因而加试一下图个确定。

12. 数据增强

我们现在已经对卷积网络中数据的重要性非常清楚了，那么就谈谈该怎样通过一些简单的转换，进一步扩大我们的现有数据集。就像我们之前所说，当电脑输入一组数据（图像）时，输入的形式通常为灰度值数组。假设我们把整幅图像平移 1 个像素，从我们的角度来说，这种变化是几乎难以察觉的。但是，对电脑来说，若要保证不影响结果，这种变化就相当明显了。这种在保持结果不变的前提下修改训练数据表现形式的方法称为数据扩容方法 data augmentation techniques。这是人工扩展数据集的方法。一些常用的扩容方法包括灰度扩展 grayscales，水平翻转 horizontal flips，垂直翻转 vertical flips，随机裁剪 random crops，颜色抖动 color jitters，翻译（转化？）translations，旋转 rotations 等等。应用这些变换方法，我们可以很轻易地成倍扩展我们的数据集。

13. 欠拟合和过拟合

训练误差和泛化误差

机器/深度学习模型在训练数据集上表现出的误差叫做训练误差，在任意一个测试数据样本上表现出的误差的期望值叫做泛化误差。

统计学习理论的一个假设是：训练数据集和测试数据集里的每一个数据样本都是从同一个概率分布中相互独立地生成出的（独立同分布假设）。

一个重要结论是：训练误差的降低不一定意味着泛化误差的降低。机器学习既需要降低训练误差，又需要降低泛化误差。

欠拟合和过拟合

欠拟合：机器/深度学习模型无法得到较低训练误差。

过拟合：机器/深度学习模型的训练误差远小于其在测试数据集上的误差。

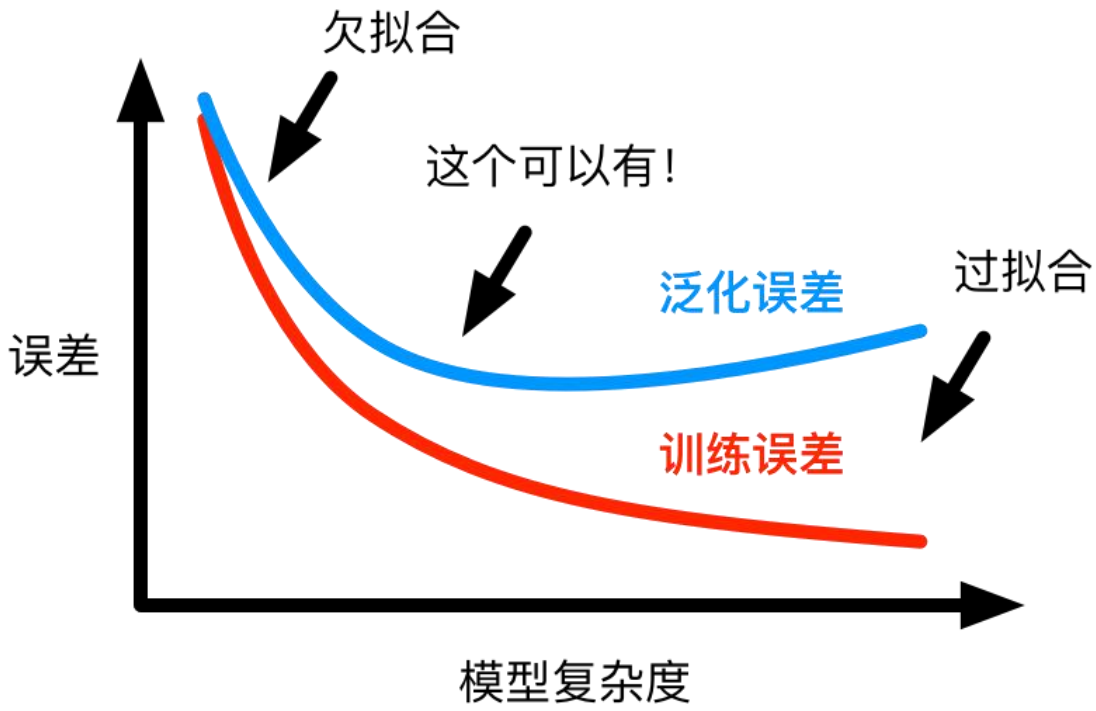
举例来谈，所谓过拟合（over-fitting）其实就是所建的机器学习模型或者是深度学习模型在训练样本中表现得过于优越，导致在验证数据集以及测试数据集中表现不佳。打个比喻就是当我需要建立好一个模型之后，比如是识别一只狗狗的模型，我需要对这个模型进行训练。恰好，我训练样本中的所有训练图片都是二哈，那么经过多次迭代训练之后，模型训练好了，并且在训练集中表现得很好。基本上二哈身上的所有特点都涵括进去，那么问题来了！假如我的测试样本是一只金毛呢？将一只金毛的测试样本放进这个识别狗狗的模型中，很有可能模型最后输出的结果就是金毛不是一条狗（因为这个模型基本上是按照二哈的特征去打造的）。所以这样就造成了模型过拟合，虽然在训练集上表现得很好，但是在测试集中表现得恰好相反，在性能的角度上讲就是协方差

过大（variance is large），同样在测试集上的损失函数（cost function）会表现得很大。

所谓欠拟合呢（under-fitting）？相对过拟合欠拟合还是比较容易理解。还是拿刚才的模型来说，可能二哈被提取的特征比较少，导致训练出来的模型不能很好地匹配，表现得很差，甚至二哈都无法识别。

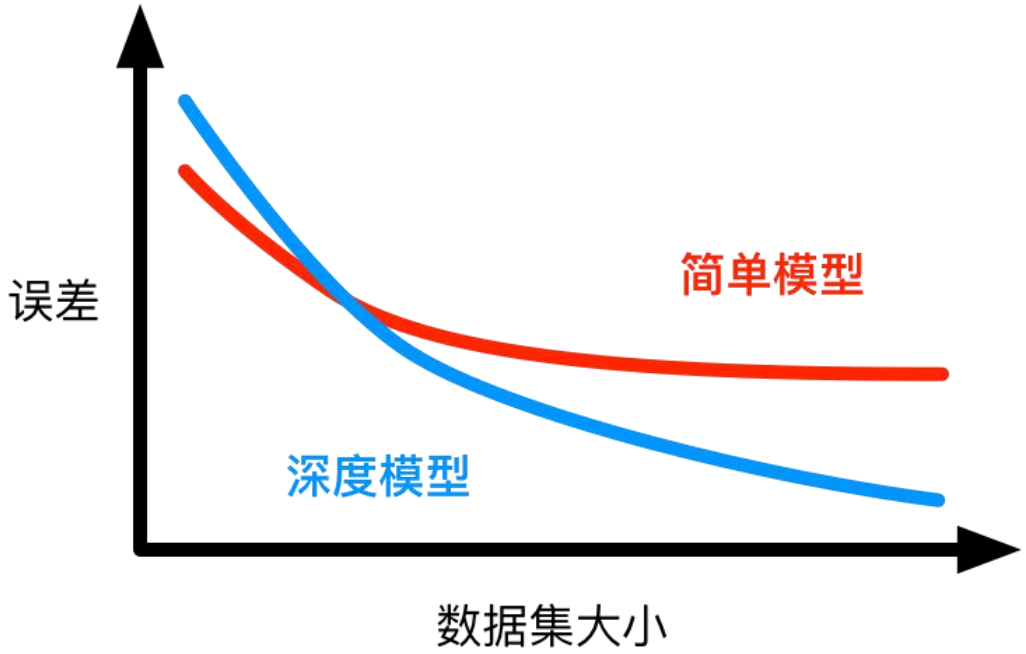
模型的选择

模型拟合能力和误差之间的关系如下图：



训练数据集的大小

一般来说，如果训练数据集过小，特别是比模型参数数量更小时，过拟合更容易发生。除此之外，泛化误差不会随训练数据集里样本数量增加而增大。



欠拟合问题

欠拟合问题易于解决，其基本方式有：

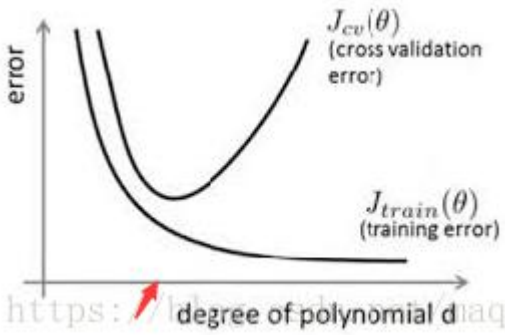
[1]增加迭代次数，使用更多的数据喂养模型，使得模型有更强的拟合能力。

[2]增加网络的深度和广度，增大神经网络的'容量'，使得模型有更好的空间表示能力。

过拟合问题

在 DNN 中常用的解决过拟合方法有：

[1]早停策略。早停是指在使用交叉检验策略，每隔一定的训练次数观察训练集和验证集上数据的准确率，从而可以比较观察找到合适的训练次数，及时在下图红点出停止，防止网络过度拟合训练集。



[2]集成学习策略。用 bagging 的思路进行正则化，对原始的 m 个训练样本进行又放回的随机采样，从而可以可以使用有放回的方式构建 N 组 m 个样本（里面可以重复）数据集，然后让这 N 组数据去训练 DNN，这样可以 DNN 往一个样本模式过度拟合，而能

学习综合的样本特征，但是这样的方式会导致数据量增加，训练更耗时。

[3]Dropout 策略。方法是在前后向传播算法每次迭代时随机隐藏一部分神经元不参与计算，并使用这些隐藏一部分数据后的网络去拟合一批数据，通过这种随机隐藏，来提升拟合限度，防止生成太拟合数据的函数式，但是这样方式需要有大量数据集来喂养，否则会导致欠拟合。

[4]正则化方法是指在进行目标函数或代价函数优化时，在目标函数或代价函数后面加上一个正则项（代表模型复杂度，又可称为惩罚项），这样我们令损失函数尽量小时，也会同时会让后面正则项代表的模型复杂度尽量小，通过这样的限制来防止模型结果过于复杂，从而有更好的普适效果。

14. 参考文献

[1]卷积神经网络 CNN 总结(https://www.cnblogs.com/skyfsm/p/6790245.html)

[2]深度学习简介(一)卷积神经网络
(https://www.cnblogs.com/alexcai/p/5506806.html)

[3]深度学习大神都推荐入门必须读完这 9 篇论文
(https://blog.csdn.net/meyh0x5vDTk48P2/article/details/79072666)

[4]深度学习 - 对过拟合和欠拟合问题的处理
(https://blog.csdn.net/maqunfi/article/details/82634450)

[5]大白话给你说清楚什么是过拟合、欠拟合以及对应措施
(https://blog.csdn.net/qq_18254385/article/details/78428887)

[6]动手学深度学习(二)——欠拟合和过拟合
(https://www.jianshu.com/p/6841b8f69ea4)

[7]CNN 中的 dropout 理解
(https://blog.csdn.net/dod_jdi/article/details/78379781)

[8]什么是激活函数？它有什么作用？
(https://blog.csdn.net/zouzhen_id/article/details/79701002)

[9]吴恩达 deeplearning 之 CNN—卷积神经网络入门
(https://blog.csdn.net/ice_actor/article/details/78648780)

[10]智能单元 CS231n 翻译笔记(https://zhuanlan.zhihu.com/intelligentunit)

[11]AI 测试 深度学习基础 (十)--训练集，验证集和测试集
(https://testerhome.com/topics/11390)