

最容易理解的对卷积(convolution)的解释

项目 github 地址：[bitcarmanlee easy-algorithm-interview-and-practice](https://github.com/bitcarmanlee/easy-algorithm-interview-and-practice)

欢迎大家 star，留言，一起学习进步

啰嗦开场白

读本科期间，信号与系统里面经常讲到卷积(convolution)，自动控制原理里面也会经常有提到卷积。硕士期间又学了线性系统理论与数字信号处理，里面也是各种大把大把卷积的概念。至于最近大火的深度学习，更有专门的卷积神经网络(Convolutional Neural Network, CNN)，在图像领域取得了非常好的实际效果，已经把传统的图像处理的方法快干趴下了。啰啰嗦嗦说了这么多卷积，惭愧的是，好像一直以来对卷积的物理意义并不是那么清晰。一是上学时候只是简单考试，没有仔细思考过具体前后的来龙去脉。二是本身天资比较愚钝，理解能力没有到位。三则工作以后也没有做过强相关的工作，没有机会得以加深理解。趁着年前稍微有点时间，查阅了一些相关资料，力争将卷积的前世今生能搞明白。

##1. 知乎上排名最高的解释

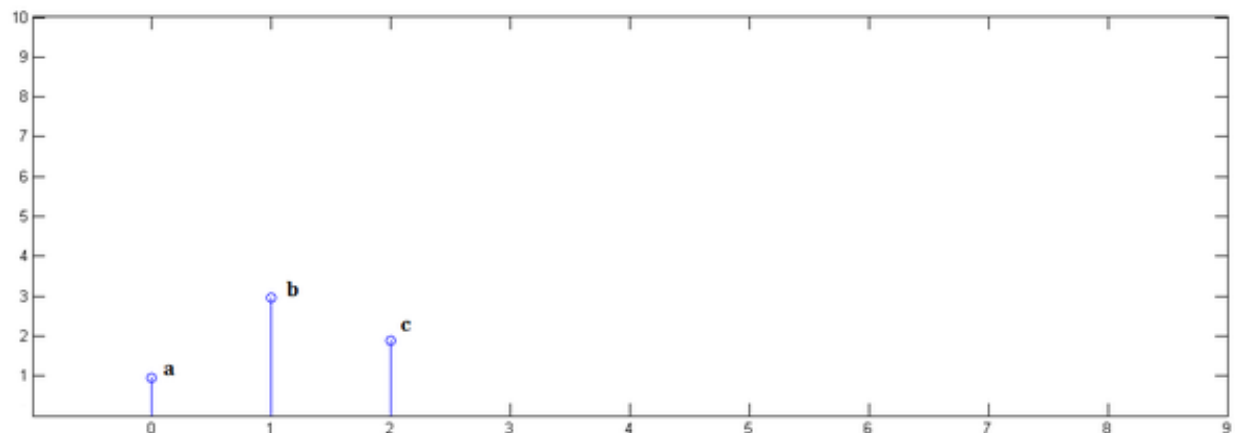
首先选取知乎上对卷积物理意义解答排名最靠前的回答。

不推荐用“反转/翻转/反褶/对称”等解释卷积。好好的信号为什么要翻转？导致学生难以理解卷积的物理意义。

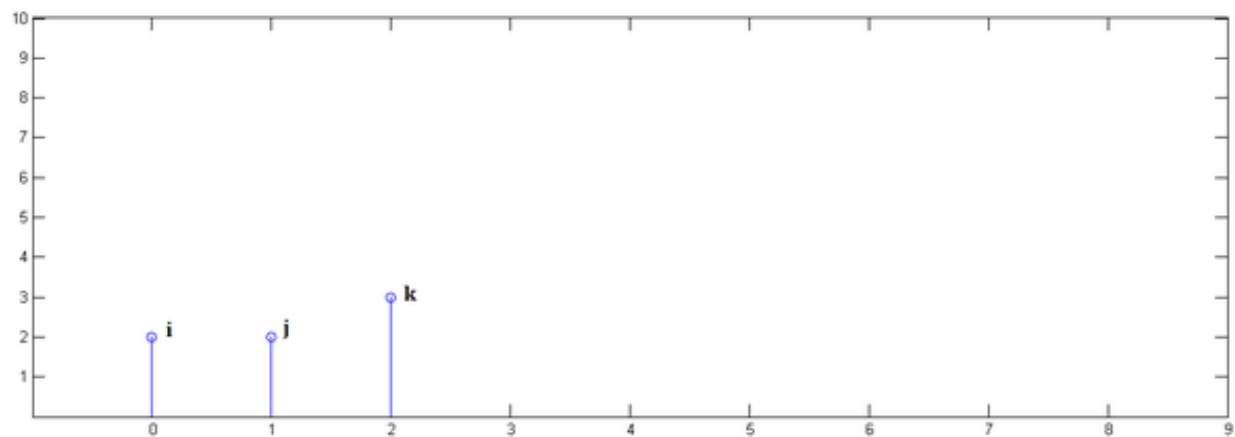
这个其实非常简单的概念，国内的大多数教材却没有讲透。

直接看图，不信看不懂。以离散信号为例，连续信号同理。

已知 $x[0] = a$, $x[1] = b$, $x[2]=c$

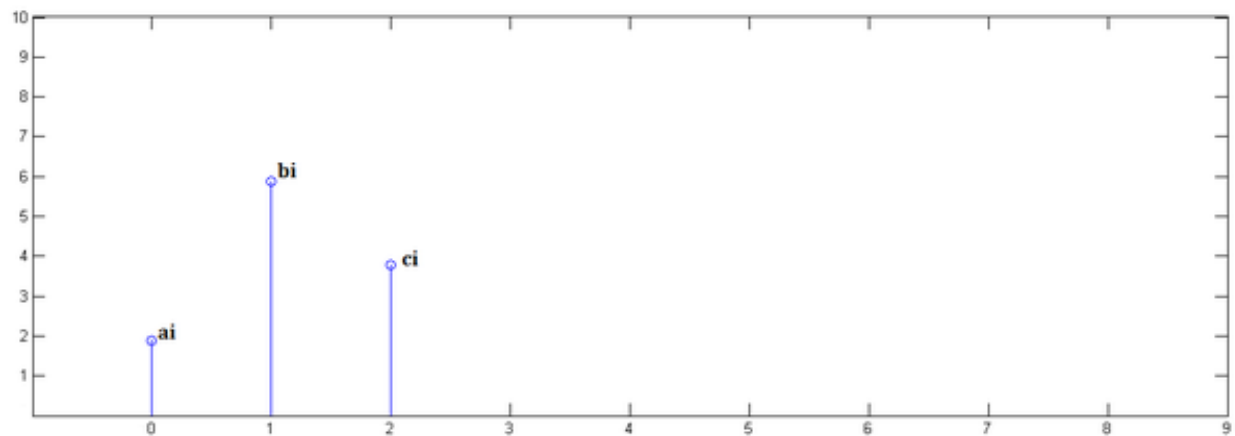


已知 $y[0] = i$, $y[1] = j$, $y[2]=k$

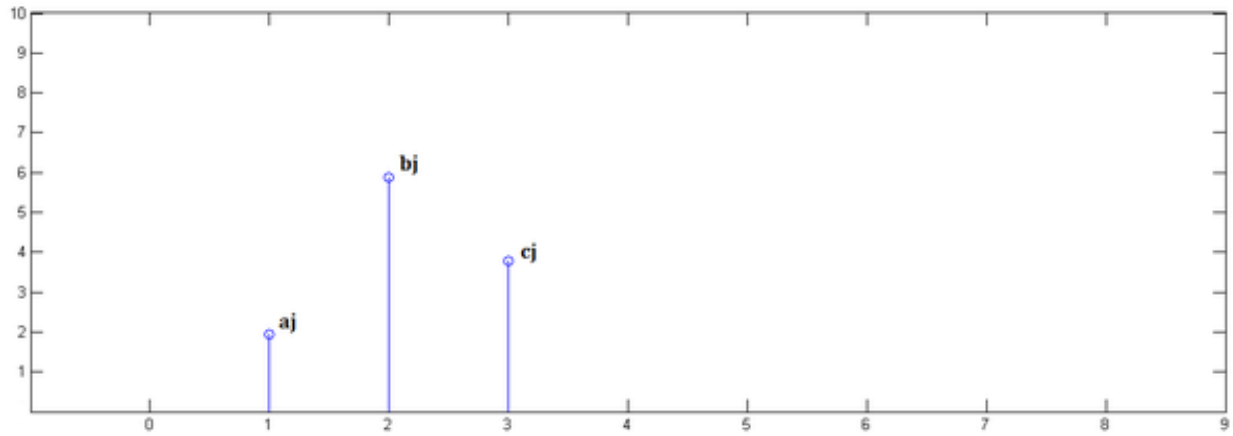


下面通过演示求 $x[n] * y[n]$ 的过程，揭示卷积的物理意义。

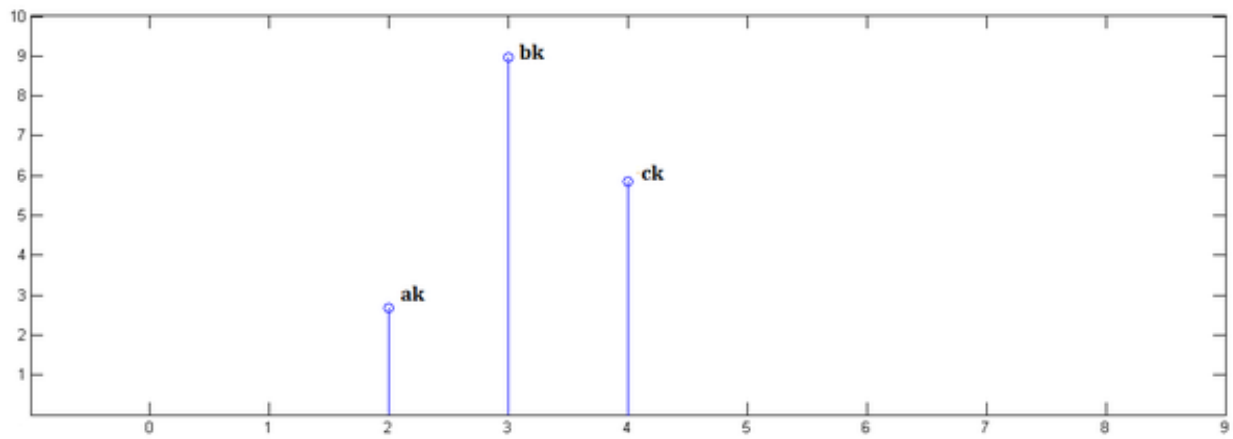
第一步， $x[n]$ 乘以 $y[0]$ 并平移到位置 0：



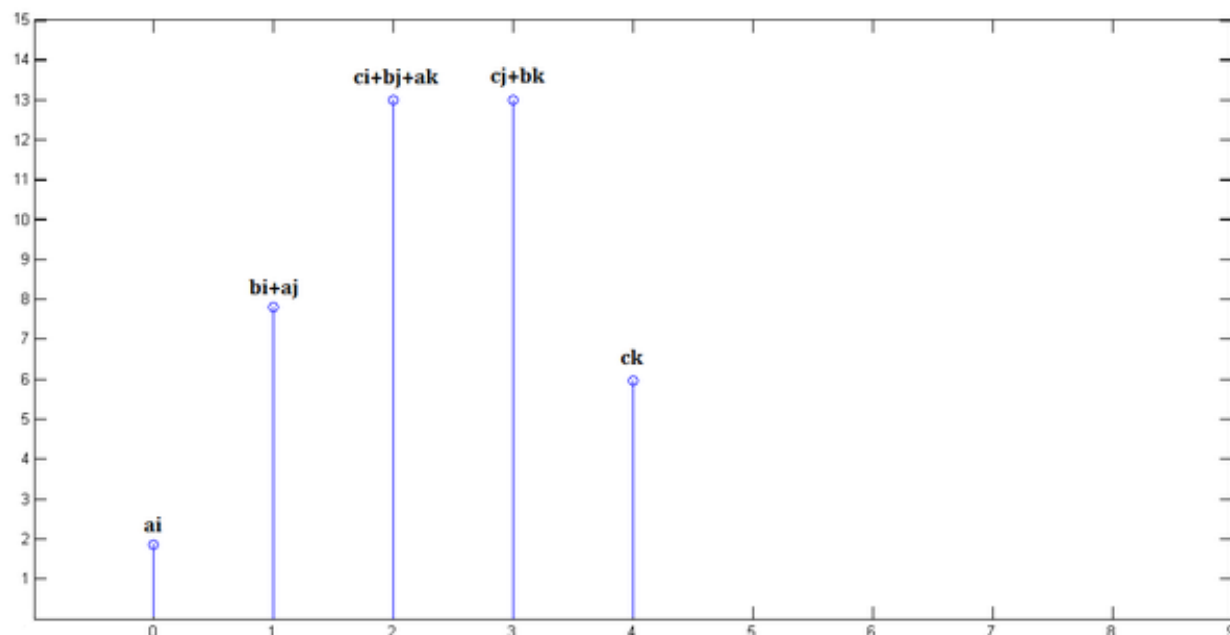
第二步， $x[n]$ 乘以 $y[1]$ 并平移到位置 1



第三步， $x[n]$ 乘以 $y[2]$ 并平移到位置 2：



最后，把上面三个图叠加，就得到了 $x[n] * y[n]$ ：



简单吧？无非是平移（没有反褶！）、叠加。

从这里，可以看到卷积的重要的物理意义是：一个函数（如：单位响应）在另一个函数（如：输入信号）上的加权叠加。

重复一遍，这就是卷积的意义：加权叠加。

对于线性时不变系统，如果知道该系统的单位响应，那么将单位响应和输入信号求卷积，就相当于把输入信号的各个时间点的单位响应 加权叠加，就直接得到了输出信号。

通俗的说：

在输入信号的每个位置，叠加一个单位响应，就得到了输出信号。

这正是单位响应是如此重要的原因。

在输入信号的每个位置，叠加一个单位响应，就得到了输出信号。

这正是单位响应是如此重要的原因。

在输入信号的每个位置，叠加一个单位响应，就得到了输出信号。
这正是单位响应是如此重要的原因。

以上是知乎上排名最高的回答。比较简单易懂。

有个回复也可以参考：

楼主这种做法和通常教材上的区别在于：书上先反褶再平移，把输入信号当作一个整体，一次算出一个时间点的响应值；而楼主把信号拆开，一次算出一个信号在所有时间的响应值，再把各个信号相加。两者本质上是相同的。

##2.卷积的另外解释

卷积表示为 $y(n) = x(n) * h(n)$

使用离散数列来理解卷积会更形象一点，我们把 $y(n)$ 的序列表示成 $y(0), y(1), y(2), \dots$ ，这是系统响应出来的信号。

同理， $x(n)$ 的对应时刻的序列为 $x(0), x(1), x(2), \dots$

其实我们如果没有学过信号与系统，就常识来讲，系统的响应不仅与当前时刻系统的输入有关，也跟之前若干时刻的输入有关，因为我们可以理解为这是之前时刻的输入信号经过一种过程（这种过程可以是递减，削弱，或其他）对现在时刻系统输出的影响，那么显然，我们计算系统输出时就必须考虑现在时刻的信号输入的响应以及之前若干时刻信号输入的响应之“残留”影响的一个叠加效果。

假设 0 时刻系统响应为 $y(0)$ ，若其在 1 时刻时，此种响应未改变，则 1 时刻的响应就变成了 $y(0) + y(1)$ ，叫序列的累加和（与序列的和不一样）。但常常系统中不是这样的，因为 0 时刻的响应不太可能在 1 时刻仍旧未变化，那么怎么表述这种变化呢，就通过 $h(t)$ 这个响应函数与 $x(0)$ 相乘来表述，表述为 $x(m) \times h(m-n)$ ，具体表达式不用多管，只要记着有大概这种关系，引入这

个函数就能够表述 $y(0)$ 在 1 时刻究竟削弱了多少，然后削弱后的值才是 $y(0)$ 在 1 时刻的真实值，再通过累加和运算，才得到真实的系统响应。

再拓展点，某时刻的系统响应往往不一定是由当前时刻和前一时刻这两个响应决定的，也可能是再加上前前时刻，前前前时刻，前前前前时刻，等等，那么怎么约束这个范围呢，就是通过对 $h(n)$ 这个函数在表达式中变化后的 $h(m-n)$ 中的 m 的范围来约束的。即说白了，就是当前时刻的系统响应与多少个之前时刻的响应的“残留影响”有关。

当考虑这些因素后，就可以描述成一个系统响应了，而这些因素通过一个表达式（卷积）即描述出来不得不说是数学的巧妙和迷人之处了。

##3.卷积的数学定义

前面讲了这么多，我们看看教科书上对卷积的数学定义。

Convolution

Several important optical effects can be described in terms of convolutions.

Let us examine the concepts using 1D continuous functions.

The convolution of two functions $f(x)$ and $g(x)$, written $f(x)*g(x)$, is defined by the integral

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha)g(x - \alpha) d\alpha. \quad (10)$$

##4.卷积的应用

用一个模板和一幅图像进行卷积，对于图像上的一个点，让模板的原点和该点重合，然后模板上的点和图像上对应的点相乘，然后各点的积相加，就得到了该点的卷积值。对图像上的每个点都这样处理。由于大多数模板都是对称的，所以模板不旋转。卷积是一种积分运算，用来求两个曲线重叠区域面积。可以看作加权求和，可以用来消除噪声、特征增强。

把一个点的像素值用它周围的点的像素值的加权平均代替。

卷积是一种线性运算,图像处理中常见的 mask 运算都是卷积，广泛应用于图像滤波。

卷积关系最重要的一种情况，就是在信号与线性系统或数字信号处理中的卷积定理。利用该定理，可以将时间域或空间域中的卷积运算等价于频率域的相乘运算，从而利用 FFT 等快速算法，实现有效的计算，节省运算代价。

##5.补充

另外在知乎上看到非常好也非常生动形象的解释，特意复制粘贴过来。(知乎马同学的解释)

从数学上讲，卷积就是一种运算。

某种运算，能被定义出来，至少有以下特征：

- 1.首先是抽象的、符号化的
- 2.其次，在生活、科研中，有着广泛的作用

比如加法：

1. $a+b$ ，是抽象的，本身只是一个数学符号
- 2.在现实中，有非常多的意义，比如增加、合成、旋转等等

卷积，是我们学习高等数学之后，新接触的一种运算，因为涉及到积分、级数，所以看起来觉得很复杂。

1 卷积的定义

我们称 $(f * g)(n)$ 为 f, g 的卷积

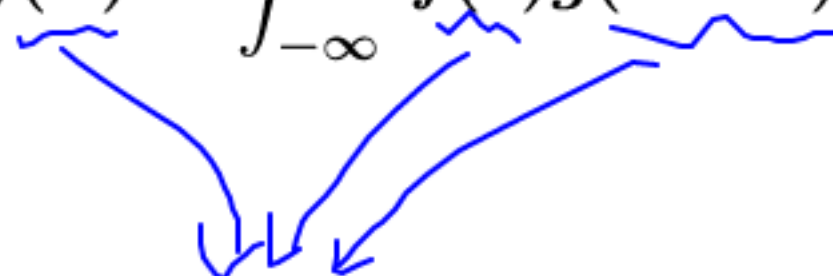
其连续的定义为：

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n - \tau)d\tau$$

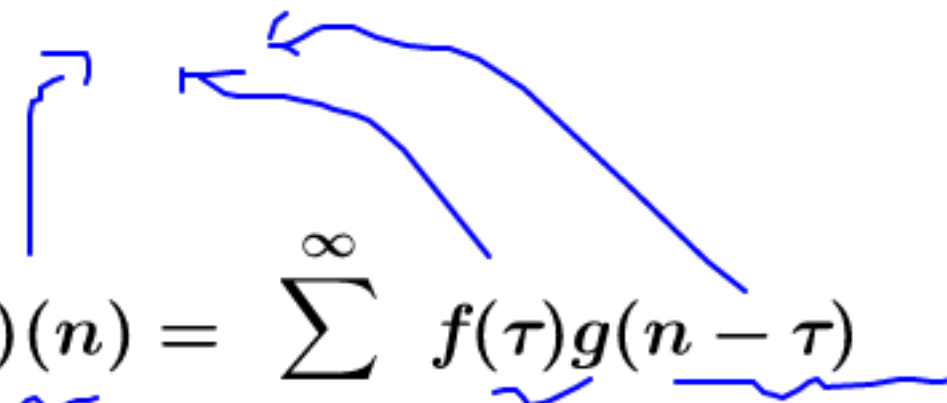
其离散的定义为：

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n - \tau)$$

这两个式子有一个共同的特征：

$$(f * g)(n) = \int_{-\infty}^{\infty} f(\tau)g(n - \tau)d\tau$$


$$n = \tau + (n - \tau)$$

$$(f * g)(n) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(n - \tau)$$


这个特征有什么意义？

只看数学符号，卷积是抽象的，不好理解的，但是，我们可以通过现实中的意义，来习惯卷积这种运算，正如我们小学的时候，学习加减乘除需要各种苹果、糖果来帮助我们习惯一样。

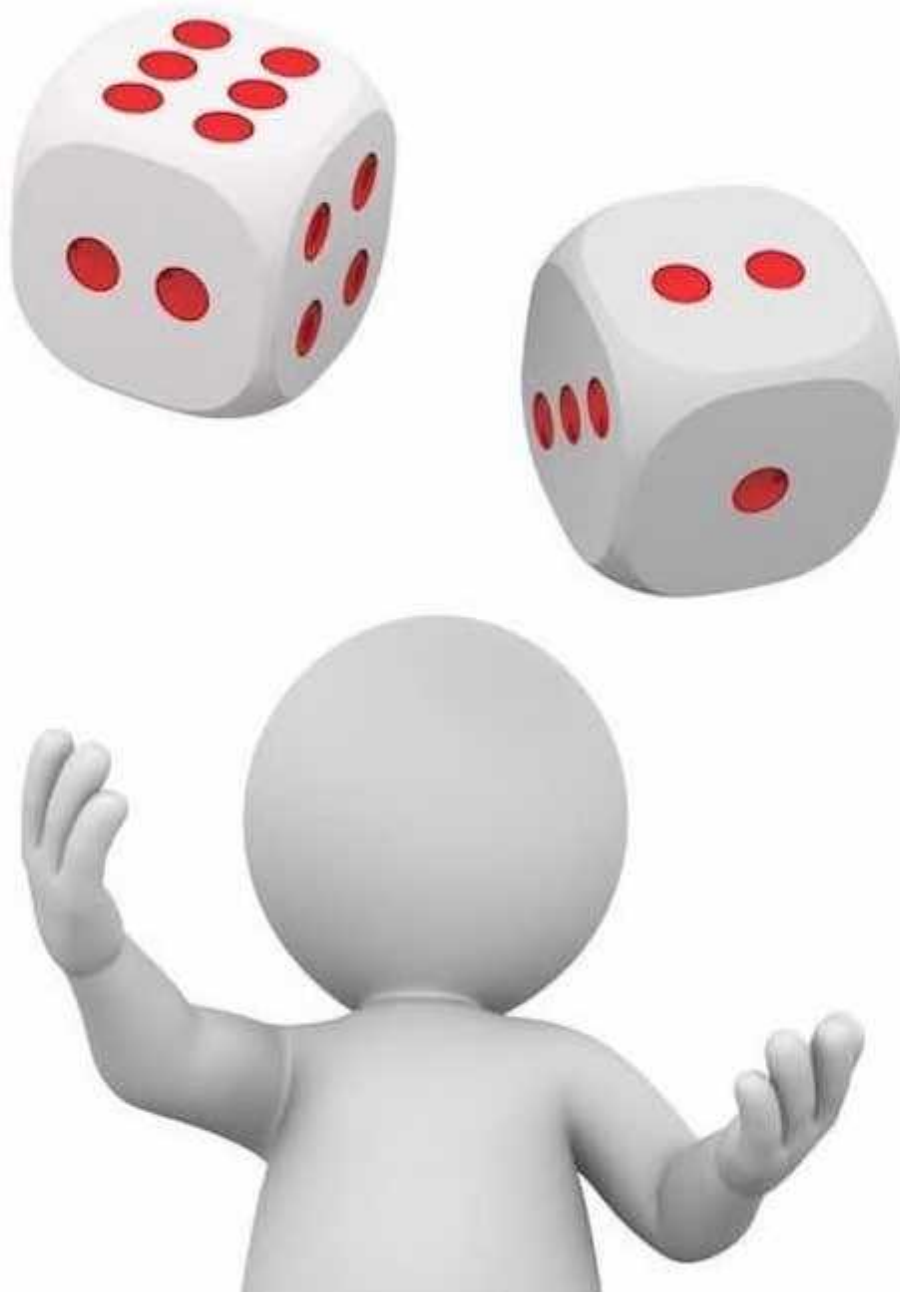
我们来看看现实中，这样的定义有什么意义。

2 离散卷积的例子：丢骰子

我有两枚骰子：



把这两枚骰子都抛出去：



求：两枚骰子点数加起来为 4 的概率是多少？

这里问题的关键是，两个骰子加起来要等于 4，这正是卷积的应用场景。

我们把骰子各个点数出现的概率表示出来：

f

1	2	3	4	5	6
---	---	---	---	---	---

f 表示第一枚骰子

$f(1)$ 表示投出1的概率

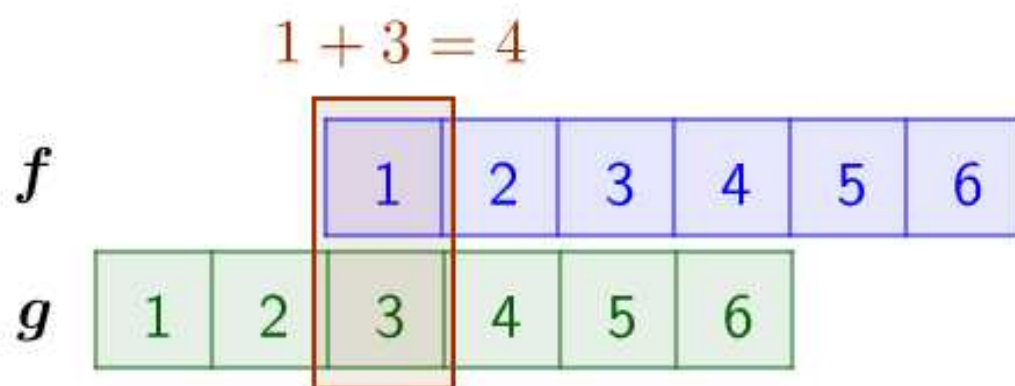
$f(2)$ 、 $f(3)$ 、 \dots 以此类推

g

1	2	3	4	5	6
---	---	---	---	---	---

g 表示第二枚骰子

那么，两枚骰子点数加起来为 4 的情况有：

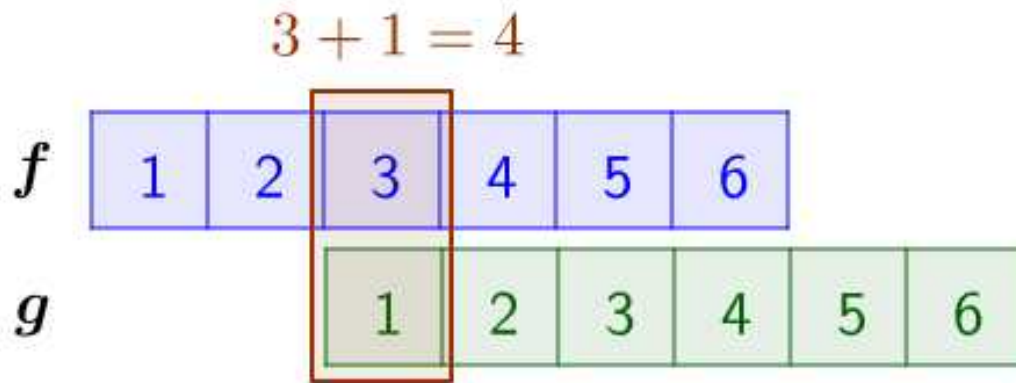


出现概率为： $f(1)g(3)$

$$2 + 2 = 4$$

f	1	2	3	4	5	6
g	1	2	3	4	5	6

出现概率为： $f(2)g(2)$



出现概率为： $f(3)g(1)$

因此，两枚骰子点数加起来为 4 的概率为：

$$f(1)g(3)+f(2)g(2)+f(3)g(1)$$

符合卷积的定义，把它写成标准的形式就是：

$$(f * g)(4) = \sum_{m=1}^3 f(4-m)g(m) \quad \text{\displaystyle}$$

$$(f * g)(4) = \sum_{m=1}^3 f(4-m)g(m) \quad (f * g)(4) = \sum_{m=1}^3 f(4-m)g(m)$$

3 连续卷积的例子：做馒头

楼下早点铺子生意太好了，供不应求，就买了一台机器，不断的生产馒头。

假设馒头的生产速度是 $f(t)$ ，那么一天后生产出来的馒头总量为：

$$\int_0^{24} f(t) dt \quad \int_0^{24} f(t) dt \quad \int_0^{24} f(t) dt$$

馒头生产出来之后，就会慢慢腐败，假设腐败函数为 $g(t)$ ，比如，10 个馒头，

24 小时会腐败：

$$10 * g(t) \quad 10 * g(t) \quad 10 * g(t)$$

想想就知道，第一个小时生产出来的馒头，一天后会经历 24 小时的腐败，第二个小时生产出来的馒头，一天后会经历 23 小时的腐败。

如此，我们可以知道，一天后，馒头总共腐败了：

$$\int_0^{24} f(t) g(24-t) dt \quad \int_0^{24} f(t) g(24-t) dt \quad \int_0^{24} f(t) g(24-t) dt$$

这就是连续的卷积。

4 图像处理

4.1 原理

有这么一副图像，可以看到，图像上有很多噪点：



这些噪点，属于高频信号

高频信号，就好像平地耸立的山峰：



看起来很显眼。

平滑这座山峰的办法之一就是，把山峰刨掉一些土，填到山峰周围去。用数学的话来说，就是把山峰周围的高度平均一下。

平滑后得到：



4.2 计算

卷积可以帮助实现这个平滑算法。

有噪点的原图，可以把它转为一个矩阵：



$$\Rightarrow \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m,0} & a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

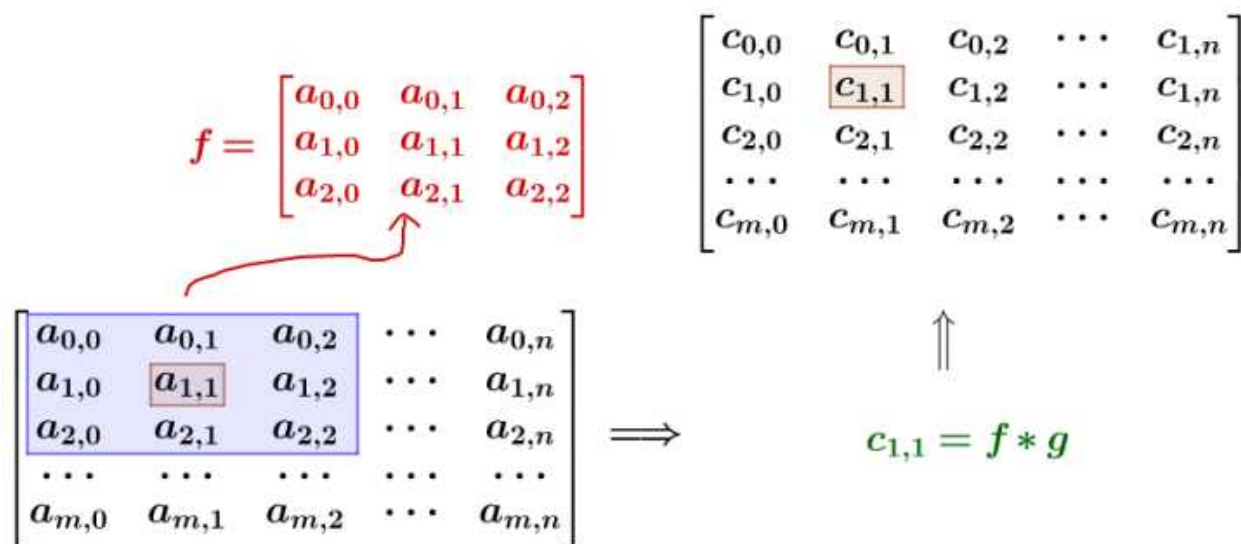
然后用下面这个平均矩阵（说明下，原图的处理实际上用的是正态分布矩阵，这里为了简单，就用了算术平均矩阵）来平滑图像：

$$g = [1919191919191919191919] g =$$

$$g = \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} \begin{bmatrix} 1919191919191919191919 \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \begin{bmatrix} 1919191919191919191919 \end{bmatrix}$$

记得刚才说过的算法，把高频信号与周围的数值平均一下就可以平滑山峰。

比如我要平滑 $a_{1,1}$ 点，就在矩阵中，取出 $a_{1,1}$ 点附近的点组成矩阵 f ，和 g 进行卷积计算后，再填回去



要注意一点，为了运用卷积， g 虽然和 f 同维度，但下标有点不一样：

注意两者的下标不一样

$$f = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \quad g = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

我用一个动图来说明下计算过程：

a, b 的下标相加都为1, 1

$$f = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \quad g = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}$$

$$c_{1,1} = a_{0,0}b_{1,1} + a_{0,1}b_{1,0} + a_{0,2}b_{1,-1} + a_{1,0}b_{0,1} \\ + a_{1,1}b_{0,0} + a_{1,2}b_{0,-1} + a_{2,0}b_{-1,1} \\ + a_{2,1}b_{-1,0} + a_{2,2}b_{-1,-1}$$

写成卷积公式就是：

$$(f * g)(1, 1) = \sum_{k=0}^2 \sum_{h=0}^2 f(h, k) g(1-h, 1-k)$$

$$\displaystyle (f * g)(1, 1) = \sum_{k=0}^2 \sum_{h=0}^2 f(h, k) g(1-h, 1-k)$$

要求 $c_{4,5}$ ，一样可以套用上面的卷积公式。

这样相当于实现了 g 这个矩阵在原来图像上的划动（准确来说，下面这幅图把 g 矩阵旋转了 180° ）：

##6. 另外一个关于卷积的有意思的解释

看了好多关于卷积的答案，看到这个例子才彻底地理解了这个过程～

关于卷积的一个血腥的讲解

比如说你的老板命令你干活，你却到楼下打台球去了，后来被老板发现，他非常气愤，扇了你一巴掌（注意，这就是输入信号，脉冲），于是你的脸上会渐渐地（贱贱地）鼓起来一个包，你的脸就是一个系统，而鼓起来的包就是你的脸对巴掌的响应，好，这样就和信号系统建立起来意义对应的联系。下面还需要一些假设来保证论证的严谨：假定你的脸是线性时不变系统，也就是说，无

论什么时候老板打你一巴掌，打在你脸的另一位置（这似乎要求你的脸足够光滑，如果你说你长了很多青春痘，甚至整个脸皮处处连续处处不可导，那难度太大了，我就无话可说了哈哈），你的脸上总是会在相同的时间间隔内鼓起来一个相同高度的包来，并且假定以鼓起来的包的大小作为系统输出。好了，那么，下面可以进入核心内容——卷积了！

如果你每天都到地下去打台球，那么老板每天都要扇你一巴掌，不过当老板打你一巴掌后，你 5 分钟就消肿了，所以时间长了，你甚至就适应这种生活了……如果有一天，老板忍无可忍，以 0.5 秒的间隔开始不间断的扇你的过程，这样问题就来了，第一次扇你鼓起来的包还没消肿，第二个巴掌就来了，你脸上的包就可能鼓起来两倍高，老板不断扇你，脉冲不断作用在你脸上，效果不断叠加了，这样这些效果就可以求和了，结果就是你脸上的包的高度随时间变化的一个函数了（注意理解）；如果老板再狠一点，频率越来越高，以至于你都辨别不清时间间隔了，那么，求和就变成积分了。可以这样理解，在这个过程中某一固定的时刻，你的脸上的包的鼓起程度和什么有关呢？和之前每次打你都有关系！但是各次的贡献是不一样的，越早打的巴掌，贡献越小，所以这就是说，某一时刻的输出是之前很多次输入乘以各自的衰减系数之后的叠加而形成某一点的输出，然后再把不同时刻的输出点放在一起，形成一个函数，这就是卷积，卷积之后的函数就是你脸上的包的大小随时间变化的函数。本来你的包几分钟就可以消肿，可是如果连续打，几个小时也消不了肿了，这难道不是一种平滑过程么？反映到剑桥大学的公式上， $f(a)$ 就是第 a 个巴掌， $g(x-a)$ 就是第 a 个巴掌在 x 时刻的作用程度，乘起来再叠加就 ok 了，大家说是不是这个道理呢？我想这个例子已经非常形象了，你对卷积有了更加具体深刻的了解了吗？

参考资料：

1. <https://www.zhihu.com/question/22298352>
2. <http://blog.csdn.net/yeeman/article/details/6325693>
3. <http://muchong.com/html/201001/1773707.html>
4. <https://www.zhihu.com/question/39753115>

5. <https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C>

6. <http://blog.csdn.net/tiandijun/article/details/40080823>

7. <https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF%E5%AE%9A%E7%90%86>

8. <https://www.zhihu.com/question/19714540/answer/14738630> 如何理解傅里叶变换公式？