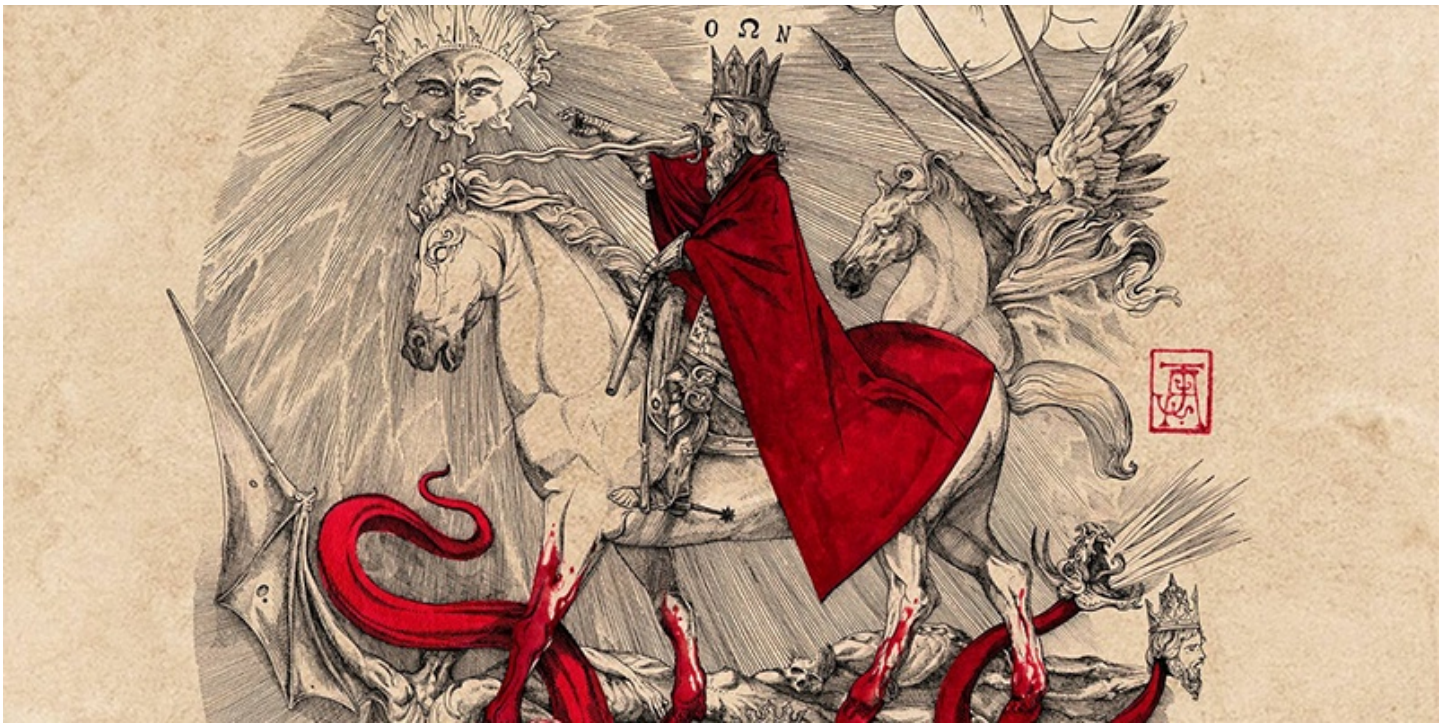


卷积神经网络简介

张觉非
All IS ONE

> 一、卷积

我们在 2 维上说话。有两个 $\mathcal{R}^2 \to \mathcal{R}$ 的函数 f(x, y) 和 g(x, y) 。所谓 f 和 g 的卷积就是一个新的 $\mathcal{R}^2 \to \mathcal{R}$ 的函数 c(x, y) 。通过下式得到：

$$c(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(s,t) \times g(x-s, y-t)\, ds\, dt$$

这式子的含义是：遍览从负无穷到正无穷的全部 s 和 t 值，把 g 在 (x-s, y-t) 上的值乘以 f 在 (s, t) 上的值之后再“加和”到一起（*积分意义上*），得到 c 在 (x, y) 上的值。说白了卷积就是一种“加权求和”：以 f 为权，以 (x, y) 为中心，把 g 距离中心 (-s, -t) 位置上的值乘上 f 在 (s, t) 的值，最后加到一起。把卷积公式写成离散形式就更清楚了：

$$C(x,y) = \sum_{t=-\infty}^{\infty}\sum_{s=-\infty}^{\infty} F(s,t) \times G(x-s, y-t)\, \Delta s\, \Delta t = \sum_{t=-\infty}^{\infty}\sum_{s=-\infty}^{\infty} F(s,t) \times G(x-s, y-t)$$

第二个等号成立是因为在这里我们每隔单位长度 1 一采样，Δs 和 Δt 都是 1 。可以令 G 表示一幅 100 x 100 大小的灰度图像。G(x, y) 取值 [0,255] 区间内的整数，是图像在 (x, y) 的灰度值。x 和 y 坐标取 [0, 99]，其它位置上 G 值全取 0 。令 F 在 s 和 t 取 {-1, 0, 1} 的位置为特定值，其他位置全取 0 。F 可以看作是一个 3 x 3 的网格。如图 1.1

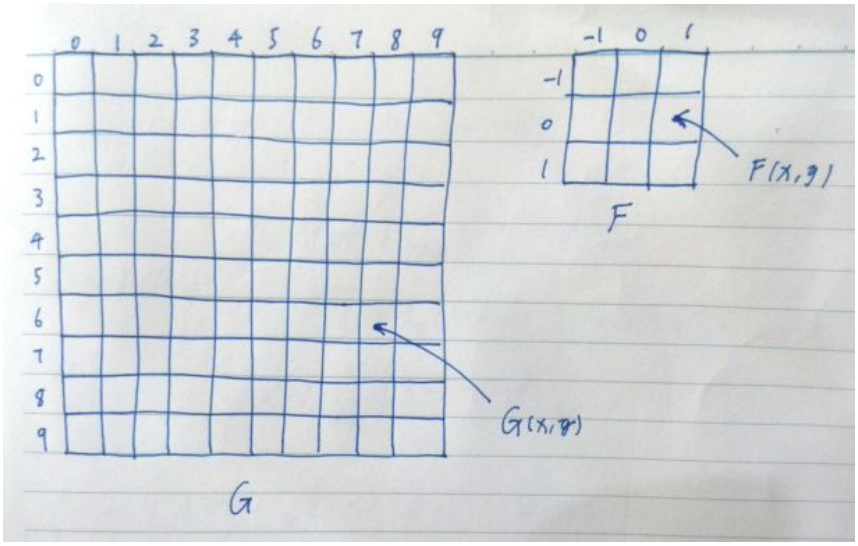


图 1.1

图 1.1 中 G 每个小格子里的值就是图像在 (x, y) 的灰度值。F 每个小格子里的值就是 F 在 (s, t) 的取值。

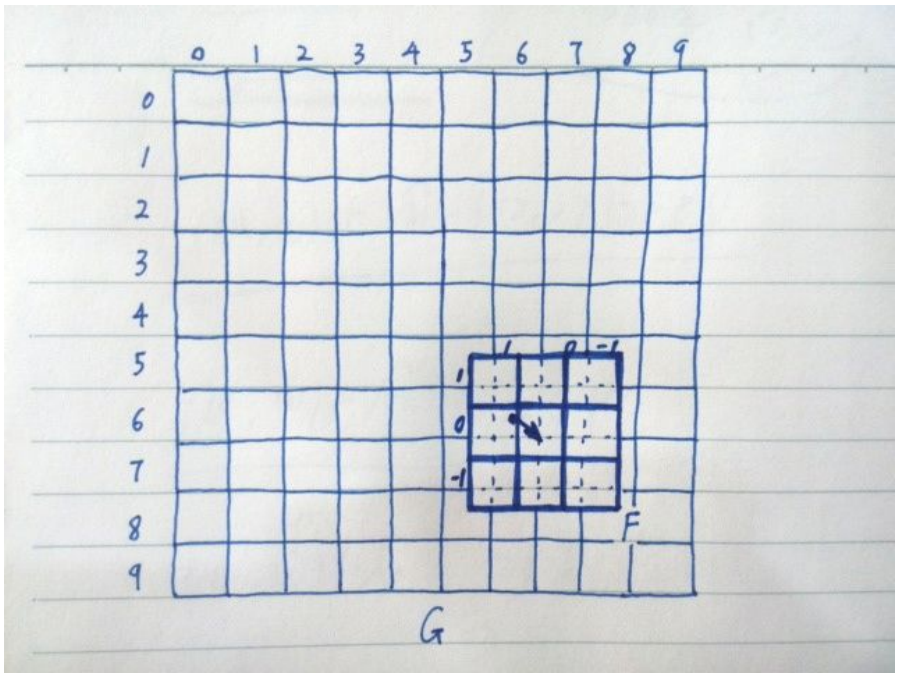


图 1.2

如图 1.2 所示，将 F 的中心 (0, 0) 对准 G 的 (6, 6) 。把 F 和 G 对应的 9 个位置上各自的值相乘，再将 9 个乘积加在一起，就得到了卷积值 C(6, 6) 。对 G 的每一个位置求 C 值，就得到了一幅新的图像。其中注意三点：

1. F 是上下左右翻转后再与 G 对准的。因为卷积公式中 F(s, t) 乘上的是 G(x-**s**, y-**t**) 。比如 F(-1, -1) 乘上的是 G(7, 7) ；
2. 如果 F 的所有值之和不等于 1.0，则 C 值有可能不落在 [0, 255] 区间内，那就不是一个合法的图像灰度值。所以如果需要让结果是一幅图像，就得将 F 归一化——令它的所有位置之和等于 1.0 ；
3. 对于 G 边缘

中是小了一圈，如果 F 覆盖范围更大，那么小的圈数更多。

上述操作其实就是对数字图像进行**离散卷积**操作，又叫**滤波**。F 称作**卷积核**或**滤波器**。不同的滤波器起不同的作用。想象一下，如果 F 的大小是 3 x 3 ，每个格子里的值都是 1/9 。那么滤波就相当于对原图像每一个点计算它周围 3 x 3 范围内 9 个图像点的灰度平均值。这应该是一种模糊。看看效果。

图 1.3

左图是 lena 灰度原图。中图用 3 x 3 值都为 1/9 的滤波器去滤，得到一个轻微模糊的图像。模糊程度不高是因为滤波器覆盖范围小。右图选取了 9 x 9 值为 1/81 的滤波器，模糊效果就较明显了。滤波器还有许多其他用处。例如下面这个滤波器：

```
+----+----+----+
| -1 |  0 |  1 |
+----+----+----+
| -2 |  0 |  2 |
+----+----+----+
| -1 |  0 |  1 |
+----+----+----+
```

注意该滤波器没有归一化（*和不是 1.0*），故滤出来的值可能不在 [0, 255] 之内。通过减去最小值、除以最大 / 最小值之差、再乘以 255 并取整，把结果值归一到 [0, 255] 之内，使之成为一幅灰度图像。现在尝试用它来滤 lena 图。

图 1.4

该滤波器把图像的边缘检测出来了。它就是 Sobel 算子。边缘检测、图像模糊等等都是人们设计出来的、有专门用途的滤波器。如果搞一个 9 x 9 的**随机**滤波器，会是什么效果呢?

图 1.5

如上图，效果也类似于模糊。因为把一个像素点的值用它周围 9 x 9 范围的值**随机**加权求和，相当于“捣浆糊”。但可以看出模糊得并不润滑。

这时我们不禁要想，如果不是由人来设计一个滤波器，而是从一个随机滤波器开始，根据某种目标、用某种方法去逐渐调整它，直到它接近我们想要的样子，可行么？这就是卷积神经网络（Convolutional Neural Network, CNN）的思想了。可调整的滤波器是 CNN 的“卷积”那部分；如何调整滤波器则是 CNN 的“神经网络”那部分。

二、神经网络

人工神经网络（Neural Network, NN）作为一个计算模型，其历史甚至要早于计算机。 W.S. McCulloch 和 W. Pitts 在四十年代就提出了人工神经元模型。但是单个人工神经元甚至无法计算异或。多个人工神经元连接成网络就可以克服无法计算异或的问题。但是对于这样的网络——多层感知机网络，当时的人们没有发现训练它的方法。人工智能领域的巨擘马文.明斯基认为这个计算模型是没有前途的。直到 7、80 年代，人们发现了训练多层感知机网络的反向传播算法（BP）。BP 的本质是梯度下降算法。多层感知机网络梯度的计算乍看十分繁琐，实则有规律。

人工神经元就是用一个数学模型简单模拟神经细胞。神经细胞有多个树突和一个伸长的轴突。一个神经元的轴突连接到其他神经元的树突，并向其传导神经脉冲。神经元会根据来自它的若干树突的信号决定是否从其轴突向其他神经元发出神经脉冲。

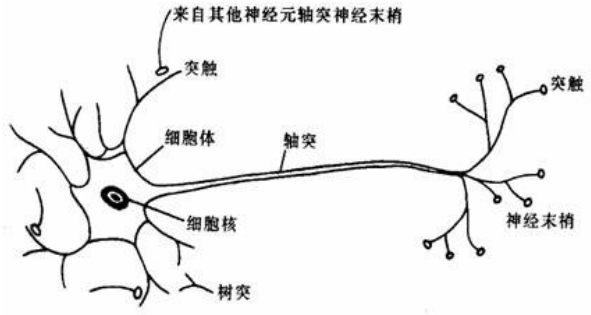


图 2.1

一个**人工神经元**就是对生物神经元的数学建模（*下文中“神经元”就指人工神经元，“神经网络”就指人工神经网络*）。见图 2.2

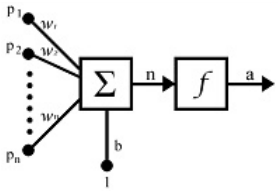


图 2.2

$p_1, p_2, \cdots, p_n$ 是神经元的输入。a 是神经元的输出。神经元将输入 $p_1, p_2, \cdots, p_n$ 加权求和后再加上偏置值 b，最后再施加一个函数 f，即：

$$a = f(n) = f\left(\sum_{i=1}^{n} p_i w_i + b\right) = f\left((w_1, w_2 \cdots w_n)\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} + b\right) = f\left(\mathcal{W}^T \mathcal{P} + b\right)$$

上式最后是这个式子的向量形式。P 是输入向量，W 是权值向量，b 是偏置值标量 。f 称为激活函数（ *Activation Function* ）。激活函数可以采用多种形式。图 2.3 展示了一些常用的激活函数。

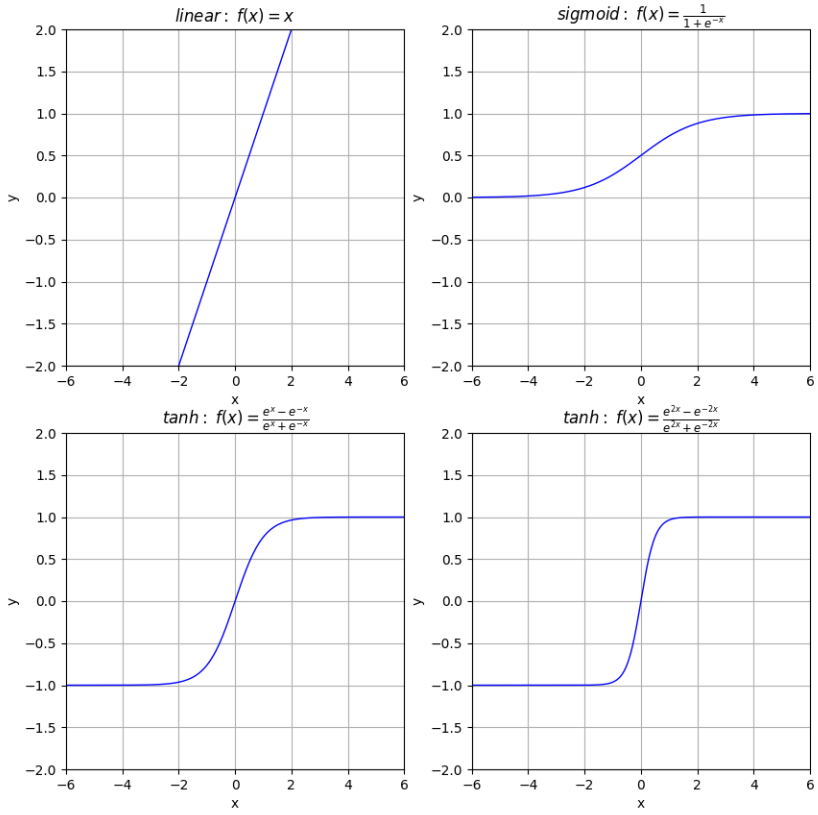


图 2.3

这是单个神经元的定义。神经网络就是把许多这样的神经元连接成一个网络：一个神经元的输出作为另一个神经元的输入。神经网络可以有多种多样的拓扑结构。其中最简单的就是“多层全连接前向神经网络”。它的输入连接到网络第一层的每个神经元。前一层的每个神经元的输出连接到下一层每个神经元的输入。最后一层神经元的输出就是整个神经网络的输出。

图 2.4 是一个三层神经网络。它接受 10 个输入，也就是一个 10 元向量。第一层和第二层各有 12 个神经元。最后一层有 6 个神经元，就是说这个神经网络输出一个 6 元向量。神经网络最后一层称为输出层，

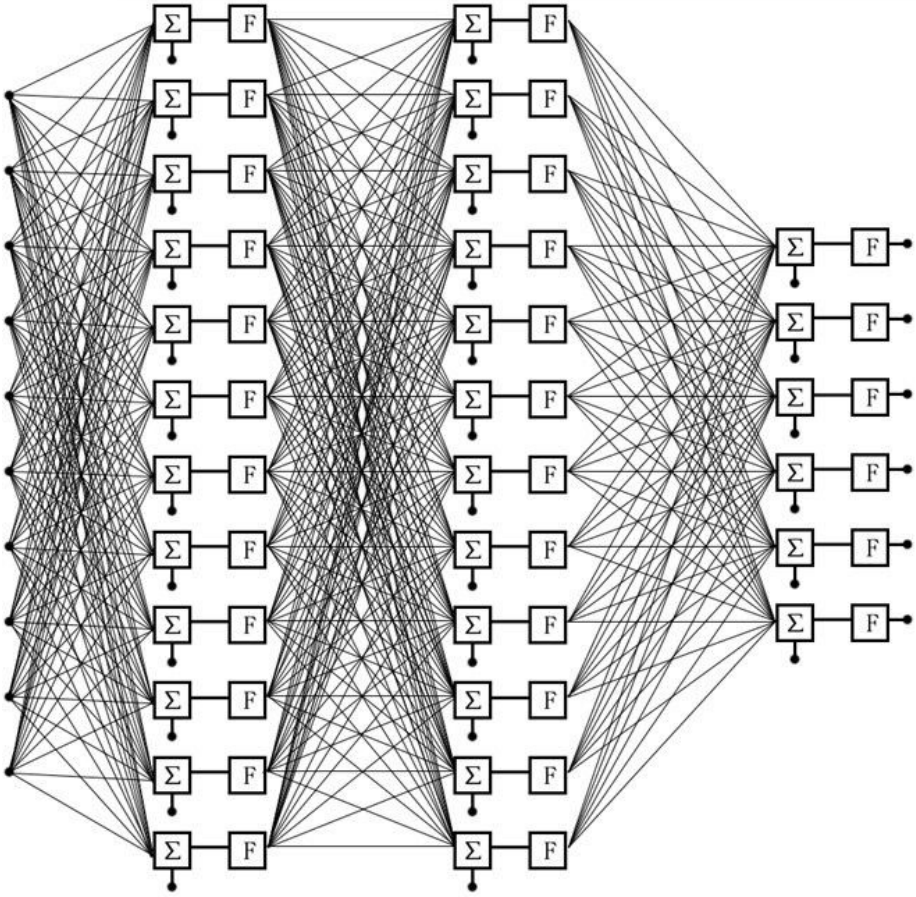


图 2.4

整个神经网络的计算可以用矩阵式给出。我们给出神经网络单层的式子。每层的神经元个数不一样，输入 / 输出维度也就不一样，计算式中的矩阵和向量的行列数也就不一样，但形式是一致的。假设我们考虑的这一层是第 i 层。它接受 m 个输入，拥有 n 个神经元（*n 个输出*），那么这一层的计算如下式所示：

$$\mathcal{O}^i = \begin{pmatrix} o_1^i \\ \vdots \\ o_n^i \end{pmatrix} = f\left(\begin{pmatrix} w_{11}^i & \cdots & w_{1m}^i \\ \vdots & \ddots & \vdots \\ w_{n1}^i & \cdots & w_{nm}^i \end{pmatrix}\begin{pmatrix} o_1^{i-1} \\ \vdots \\ o_m^{i-1} \end{pmatrix} + \begin{pmatrix} b_1^i \\ \vdots \\ b_n^i \end{pmatrix}\right)$$

上标 i 表示第 i 层。 $\mathcal{O}^i$ 是输出向量，n 元，因为第 i 层有 n 个神经元。第 i 层的输入，即第 i-1 层的输出，是 m 元向量。权值矩阵 W 是 n x m 矩阵：n 个神经元，每个神经元有 m 个权值。W 乘以第 i - 1 层输出的 m 向量，得到一个 n 向量，加上 n 元偏置向量 b，再对结果的每一个元素施以激活函数 f，最终得到第 i 层的 n 元输出向量。

若不嫌繁琐，可以将第 i - 1 层的输出也展开，最终能写出一个巨大的式子。它就是整个全连接前向神经网络的计算式。可以看出整个神经网络其实就是一个向量到向量的函数。至于它是什么函数，就取决于网络拓扑结构和每一个神经元的权值和偏置值。如果随机给出权值和偏置值，那么这个神经网络是无用的。我们想要的是有用的神经网络。它应该表现出我们想要的行为。

要达到这个目的，首先准备一个从目标函数采样的包含若干“输入 - 输出对”的集合——训练集。把训练集的输入送给神经网络，得到的输出肯定不是正确的输出。因为一开始这个神经网络的行为是随机的。

把一个训练样本输入给神经网络，计算输出与目标输出的（*向量*）差的模平方（*自己与自己的内积*）。再把全部 n 个样本的差的模平方求平均，得到 e：

$$e = \frac{1}{2n}\sum_{i=1}^{n}||o_i^{real} - o_i^{output}||^2$$

e 称为均方误差 mse 。全部输出向量和目标输出向量之间的距离（*差的模*）越小，则 e 越小。e 越小则神经网

目标是使 e 变小。在这里 e 可以看做是全体权值和偏置值的一个函数。这就成为了一个无约束优化问题。如果能找到一个全局最小点，e 值在可接受的范围内，就可以认为这个神经网络训练好了。它能够很好地拟合目标函数。这里待优化的函数也可以是 mse 外的其他函数，统称 Cost Function，都可以用 e 表示。

经典的神经网络的训练算法是反向传播算法（Back Propagation, BP）。BP 算法属于优化理论中的梯度下降法（Gradient Descend）。将误差 e 作为全部权值和全部偏置值的函数。算法的目的是在自变量空间内找到 e 的全局极小点。

首先随机初始化全体权值和全体偏置值，之后在自变量空间中沿误差函数 e 在该点的梯度方向的反方向前进一个步长。梯度的反方向上函数方向导数最小，函数值下降最快。步长称为学习速率（Learning Rate, LR）。如此反复迭代，最终（*至少是期望*）解运动到误差曲面的全局最小点（*请参考专栏文章：神经网络之梯度下降与反向传播（上）*）。

图 2.5 是用 matlab 训练一个极简单的神经网络。它只有单输入单输出。输入层有两个神经元，输出层有一个神经元。整个网络有 4 个权值加 3 个偏置。图中展示了固定其他权值，只把第一层第一个神经元的权值 $w^1_{(1,1)}$ 和偏置 b^1_1 做自变量时候的 e 曲面，以及随着算法迭代，解的运动轨迹。

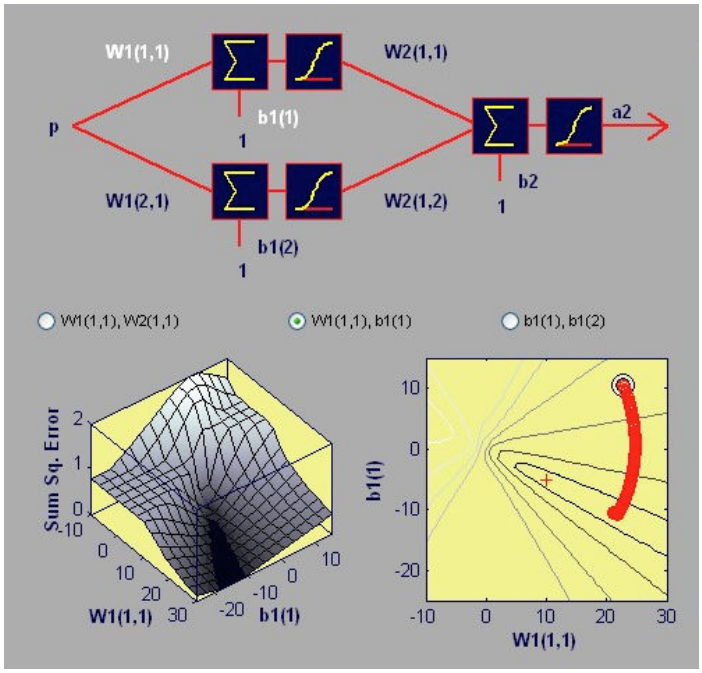


图 2.5

最终算法没有收敛到全局最优解（*红 +*）。但是解已经运动到了一个峡谷的底部。由于底部过于平缓，解“走不动”了。所得解比最优也差不到哪去。

对于一个稍复杂的神经网络，e 对权值和偏置值的函数是一个复杂的函数。求梯度需要计算 e 对每一个权值和偏置值的偏导数。所幸的是求偏导的公式不会因为这个权值或偏置值距离输出层越远而越复杂。对于每个神经元可以计算一个值 delta ，称“局部误差”或“灵敏度”。得到了每个神经元的 delta 就很容易计算 e 对任何一个权值或偏执值的偏导数。

计算某神经元的 delta 用到该神经元激活函数的导函数，对于输出层用到输出与目标输出的差；对于隐藏层用到下一层各神经元的 delta 。每个神经元将其 delta 传递给前一层的各神经元。前一层的各神经元收集后一层的神经元的 delta 计算自己的 delta 。这就是“反向传播”名称的由来——“局部误差”或“灵敏度” delta 沿着反向向前传，逐层计算所有权值和偏置值的偏导数，最终得到梯度。详细推导可参考书籍［1］第八章、［2］第十一章、［3］第四章、［4］第三章或［5］第十一章（*或参考专栏文章：神经网络之梯度下降与反向传播（下）*）。

梯度下降法有
部最优点以及

的 e 更新权值。梯度下降是基于误差函数的一阶性质。还有其他方法基于二阶性质进行优化，比如牛顿法等等。优化作为一门应用数学学科是机器学习的一个重要理论基础，在理论和实现上均有众多结论和方法。参考［1］。

三、卷积神经网络

现在把卷积滤波器和神经网络两个思想结合起来。卷积滤波器无非就是一套权值。而神经网络也可以有（*除全连接外的*）其它拓扑结构。可以构造如图 3.1 所示意的神经网络。

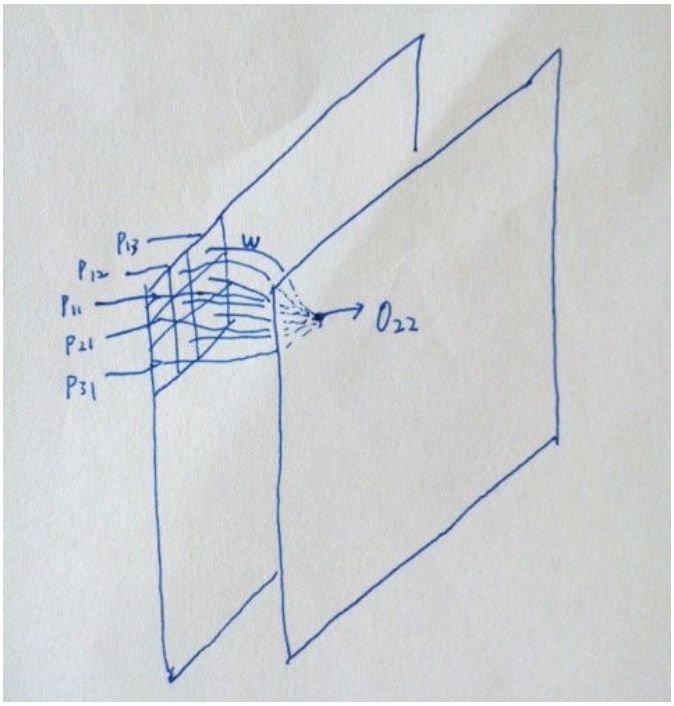


图 3.1

该神经网络接受 n x n 个输入，产生 n x n 个输出。图中左边的平面包含 n x n 个格子，每个格子中是一个 [0, 255] 的整数值。它就是输入图像，也是这个神经网络的输入。右边的平面也是 n x n 个格子，每个格子是一个神经元。每个神经元连接到输入上它对应位置周围 3 x 3 范围内的值。每个连接有一个权值。所有神经元都如此连接（*图中只画了一个，出了输入图像边缘的连接就认为连接到常数 0*）。右边层的每个神经元将与它连接的 3 x 3 个输入的值乘上连接权重并加和，得到该神经元的输出。n x n 个神经元的输出就是该神经网络的输出。

这个神经网络有两点与全连接神经网络不同。首先它不是全连接的。右层的神经元并非连接上全部输入，而是只连接了一部分。这里的一部分就是输入图像的一个局部区域。我们常听说 CNN 能够把握图像局部特征就是这个意思。这样一来权值少了很多，因为连接少了。权值其实还更少，因为每一个神经元的 9 个权值都是和其他神经元共享的。全部 n x n 个神经元都用这共同的一组 9 个权值，并且不要偏置值。那么这个神经网络其实一共只有 9 个参数需要调整。

看了第一节的同学们都看出来了，这个神经网络所进行的计算不就是一个卷积滤波器么？只不过卷积核的参数未定，需要我们去训练——它是一个“可训练滤波器”。这个神经网络其实就是一个只有一个卷积层、且该卷积层只有一个滤波器（*通道*）的 CNN 。

试着用 Sobel 算子滤出来的图片作为目标值去训练这个神经网络。给神经网络的输入是灰度 lena 图，目标输出是经过 Sobel 算子滤波的 lena 图，见图 1.4 。这唯一的一对输入输出图片就构成了训练集。神经网络权值随机初始化，训练 2000 轮。如图 3.7 。

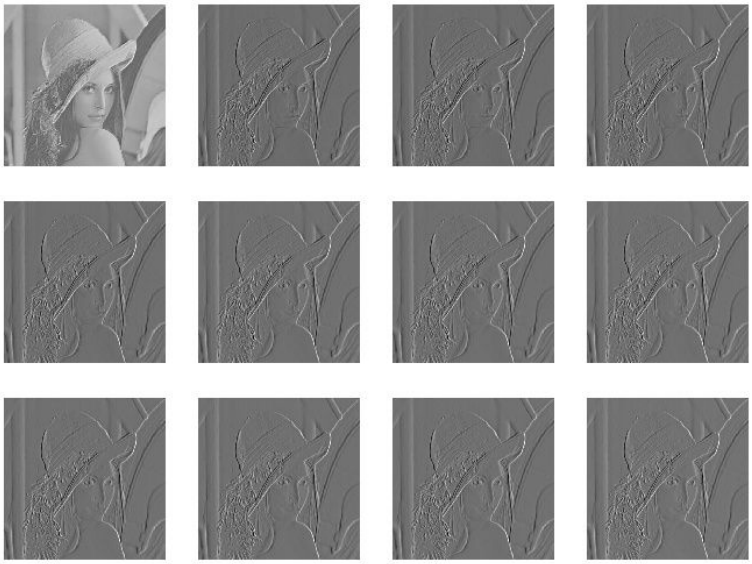


图 3.2

从左上到右下依次为：初始随机滤波器输出、每个 200 轮训练后的滤波器输出（ *10 幅*）、最后一幅是 Sobel 算子的输出，也就是用作训练的目标图像。可以看到经过最初 200 轮后，神经网络的输出就已经和 Sobel 算子的输出看不出什么差别了。后面那些轮的输出基本一样。输入与输出的均方误差 mse 随着训练轮次的变化。如图 3.3 。

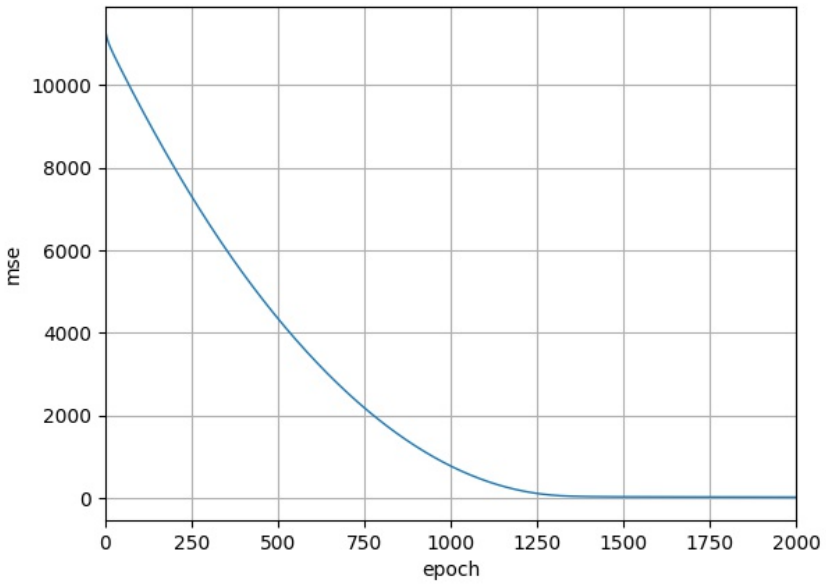


图 3.3

1500 轮过后，mse 基本就是 0 了。训练完成后网络的权值是：

```
+-------+--------+--------+
| 1.29  |  0.04  |  -1.31 |
+-------+--------+--------+
| 1.43  |  0.01  |  -1.45 |
+-------+--------+--------+
| 1.34  |  -0.07 |  -1.28 |
+-------+--------+--------+
```

与 Sobel 算子比较一下：

```
+----+----+----+
| -1 |  0 |  1 |
+----+----+----+
| -2 |  0 |  2 |
+----+----+----+
| -1 |  0 |  1 |
+----+----+----+
```

注意训练出来的滤波器负数列在右侧而不是左侧。因为计算卷积是把滤波器上下左右翻转反着扣上去的。这并不重要，本质是相同的。关键是一正列、一负列，中间零值列。非零值列三个值之比近似 1:2:1 。我们得到的就是一个近似的 Sobel 算子。我们以训练神经网络的方式把一个随机滤波器训练成了 Sobel 算子。这就是优化的魔力（*代码见本文最后*）。

在 CNN 中，这样的滤波器层叫做卷积层。一个卷积层可以有多个滤波器，每一个叫做一个 channel 。图像是二维信号。信号也可以是其他维度的，比如一维、三维乃至更高维度。那么滤波器相应的也有各种维度。回到二维图像的例子，实际上一个卷积层面对的是多个 channel 的 “一摞” 二维图像。比如一幅 100 x 100 大小的彩色图就会有 RGB 三个 channel ，其数据维度是 3 x 100 x 100 。那么直接连接彩色图像输入的卷积层面对的是 3 x 100 x 100 的数据，这时它的滤波器是 3 维度，第一维等于输入 channel 数（*这里是 3*）。第 2、3 维度是指定的滤波器大小，例如 5 x 5 。卷积层把输入的多 channel 的一摞二维图像用三维滤波器滤出一幅二维图像。假如这层有 32 个滤波器，那么这层输出 32 个 channel ，每个 channel 是一个二维图像。

激活函数构成 CNN 的一种层——激活层，这样的层没有可训练的参数。它为输入施加激活函数，例如 Sigmoid 、Tanh 等。

还有一种层叫做 Pooling 层（*池化层*）。它也没有参数，起到降维的作用。将输入切分成不重叠的一些 n x n 区域。每一个区域就包含 n x n 个值。从这 n x n 个值计算出一个值。计算方法可以是求平均、取最大等等。假设 n = 2，那么 4 个输入变成一个输出。输出图像就是输入图像的 1/4 大小。若把 2 维的层展平成一维向量，后面可再连接一个全连接前向神经网络。

通过把这些组件进行组合就得到了一个 CNN 。它直接以原始图像为输入，以最终的回归或分类问题的结论为输出，内部兼有滤波图像处理和函数拟合，所有参数放在一起训练。这就是卷积神经网络。

> 四、举个栗子

手写数字识别。数据集中一共有 42000 个 28 x 28 的手写数字灰度图片。十个数字（ *0~9* ）的样本数量大致相等。为减少训练时间，随机抽取其中 10000 个。图 4.1 展示其中一部分。

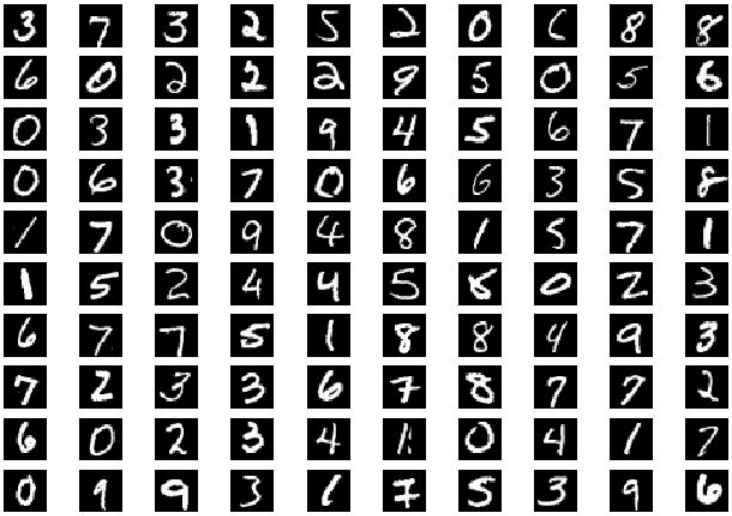


图 4.1

将样本集合的 75% 用作训练，剩下的 25% 用作测试。构造一个结构如图 4.2 的 CNN 。

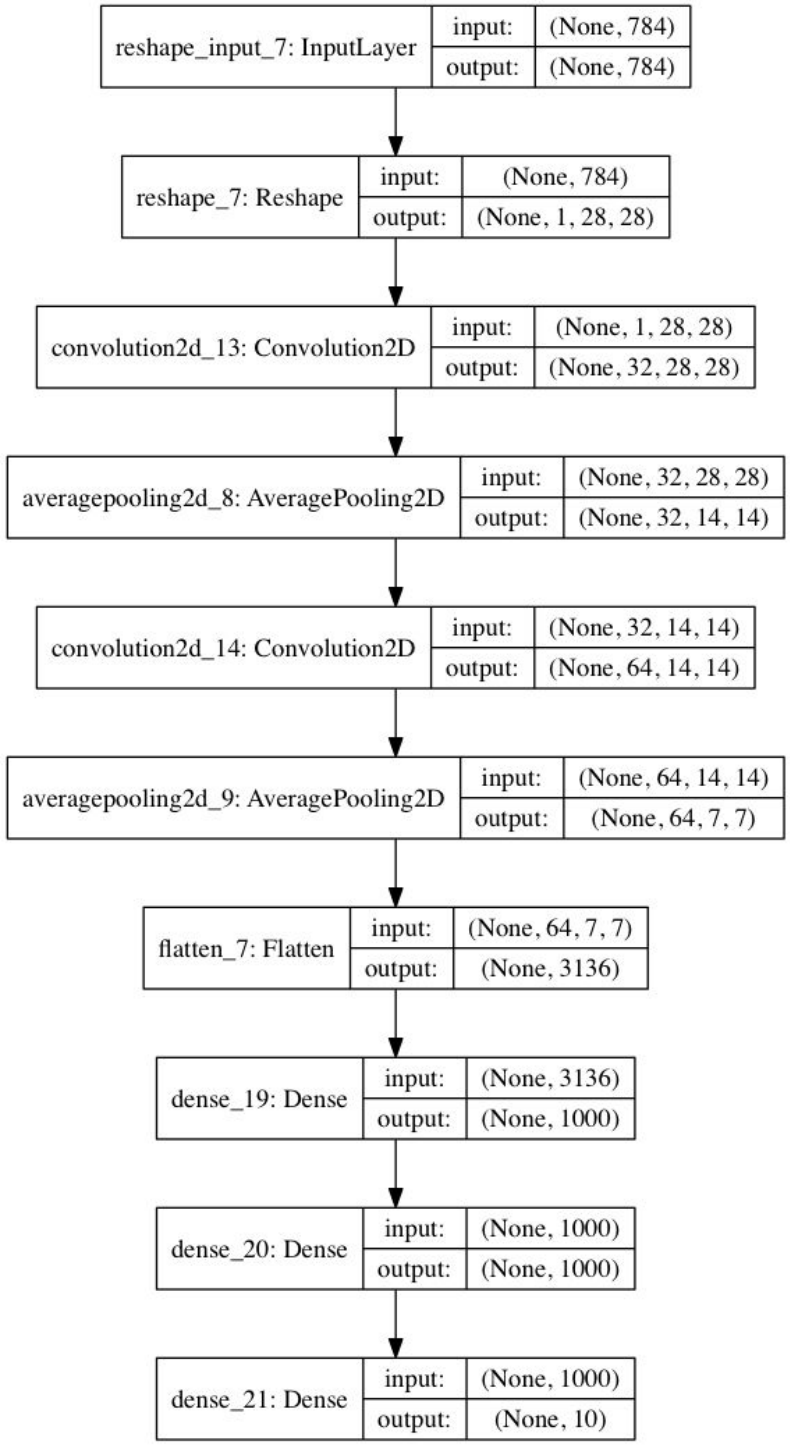


图 4.2

该 CNN 共有 9 层（*不包括输入层*）。它接受 784 元向量作为输入，就是一幅 28 x 28 的灰度图片。并没有将图片先变形成 28 x 28 再输入，因为在 CNN 的第一层放了一个 reshape 层。该层负责将 784 元的输入向量变形成 1 x 28 x 28 的阵列。最开始那个 1 x 表示只有一个通道，因为这是灰度图像。如果是彩色图像，就有 RGB 三个通道 。

接下来放一个卷积层。它包含 32 个 3 x 3 的滤波器，所以它的输出维度是 32 x 28 x 28 。32 个滤波器搞出来 32 幅图像（*通道*），每个都是 28 x 28 大小。后续一个 2 x 2 的取平均值 Pooling 层把维度减小一半：32 x 14 x 14 。

接着是第二个卷积层。它包含 64 个 32 x 3 x 3 的滤波器。它的输出维度是 64 x 14 x 14 。注意该卷积层的输入是 32 个 channel ，每个 14 x 14 大小。可以看作 32 x 14 x 14 的一个 3 维输入。该层的滤波器是 32 x 3 x 3 的一个 3 维滤波器。该层的输出维度是 64 x 14 x 14 。后面再续一个 2 x 2 的取平均值 Pooling 层，输出维度：64 x 7 x 7 。

接着是一个展平层，没有运算也没有参数，只变化一下数据形状：把 64 x 7 x 7 展平成了 3136 元向量。该 3136 元向量送给后面一个三层的全连接神经网络。该网络的结构是 1000 x 1000 x 10 。两个隐藏层各有 1000 个神经元，最后的输出层有 10 个神经元，代表 10 个数字。假如第六个输出为 1 ，其余输出为 0 ，就表示网络判定这个手写数字为 “5” （*数字 “0” 占第一个输出，所以 “5” 占第六个输出*）。数字 “5” 就编码成了：

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

训练集和测试集的数字标签都这么编码（ *one-hot 编码*）。

全连接神经网络的隐藏层的激活函数采用 Sigmoid ，输出层的激活函数采用 Linear 。误差函数采用均方误差 mse 。优化算法采用随机梯度下降 SGD 。SGD 是梯度下降的一个变体。它并不是用全体样本计算 e 的梯度，而是每次迭代使用随机选择的一部分样本来计算。学习速率 LR 初始为 0.01 ，每次迭代以 1e-6 的比例衰减。以 0.9 为参数设置冲量。训练过程持续 10 轮（ *epoch* ）。注意这里 10 轮不是指当前解在解空间只运动 10 步。一轮是指全部 7500 个训练样本都送进网络迭代一次。每次权值更新以 32 个样本为一个 batch 提交给算法。

图 4.3 展示了随着训练进行，mse 以及分类正确率（ *accuracy* ）的变化情况（ *横坐标取了 log* ）。

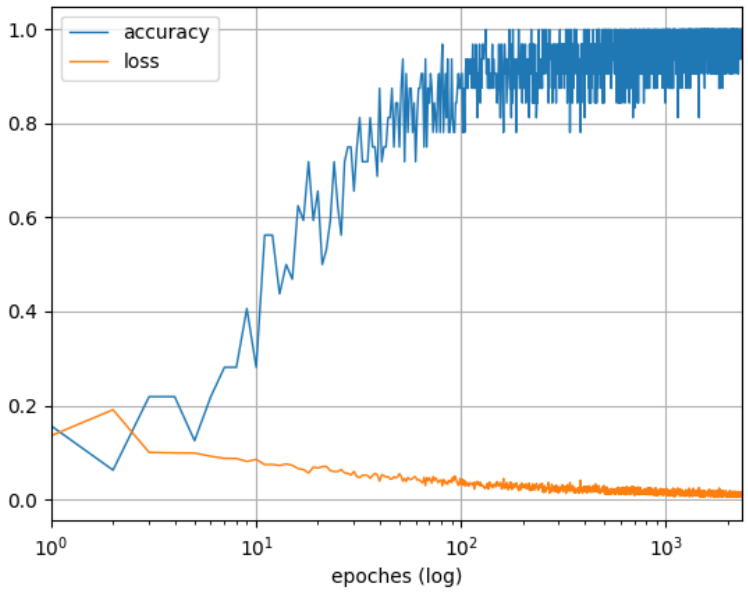


图 4.3

该 CNN 在测试集上的正确率（*accuracy*）是 96.12%，各数字的准确率（*precision*）/ 召回率（*recall*）/ f1-score 如下：

	precision	recall	f1-score	support
0	0.96	0.98	0.97	252
1	0.99	0.99	0.99	281
2	0.98	0.94	0.96	240
3	0.97	0.95	0.96	258
4	0.96	0.92	0.94	239
5	0.98	0.95	0.97	219
6	0.96	0.99	0.97	273
7	0.97	0.97	0.97	259
8	0.92	0.96	0.94	231
9	0.91	0.97	0.94	248
avg / total	0.96	0.96	0.96	2500

训练完成神经网络后，最有趣的是将其内部权值以某种方式展现出来。看着那些神秘的、不明所以的连接强度最后竟产生表观上有意义的行为，不由让我们联想起大脑中的神经元连接竟构成了我们的记忆、人格、情感 ... 引人遐思。

在 CNN 上就更适合做这种事情。因为卷积层训练出来的是滤波器。用这些滤波器把输入图像滤一滤，看看 CNN 到底“看到”了什么。图 4.4 是该 CNN 第一卷积层对一个手写数字 “5” 的 32 个输出。

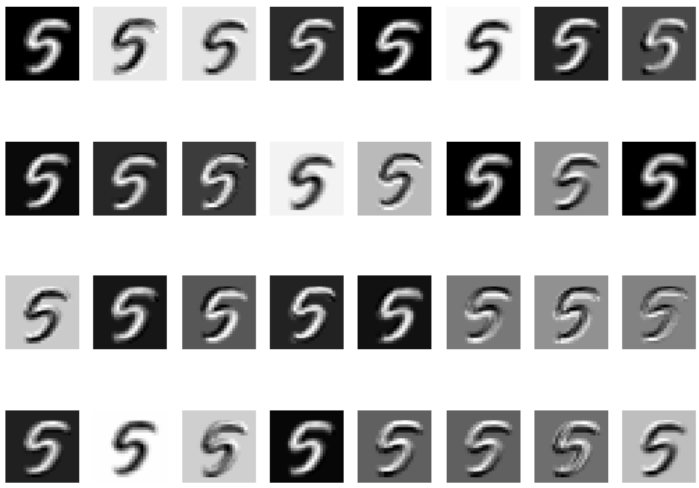


图 4.4

接下来看一看第二卷积层输出的 64 幅图像。

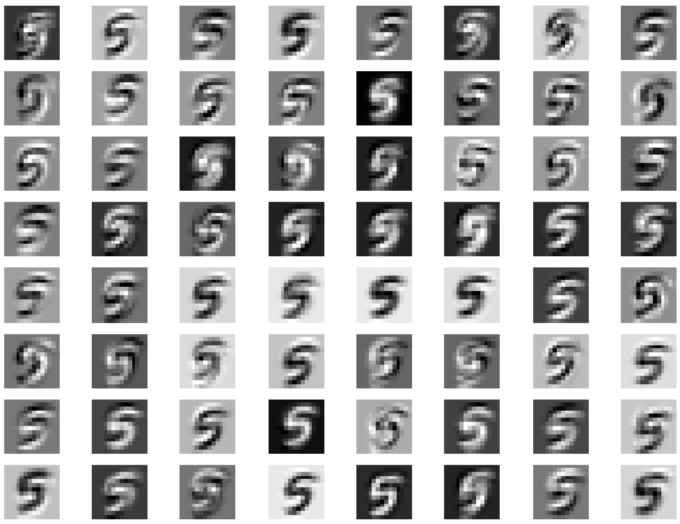


图 4.5

这些就是 CNN 经两步滤波后“看到”的信息。现在将展平层的 3136 元输出呈现出来。呈现方式是：“0”～“9” 十个数字各取 100 个（*共 1000 个*），将对每一个样本的输出作为一行，得到一副 1000 x 3136 大小的图像，根据数值用伪彩色呈现出来。如图 4.6 。

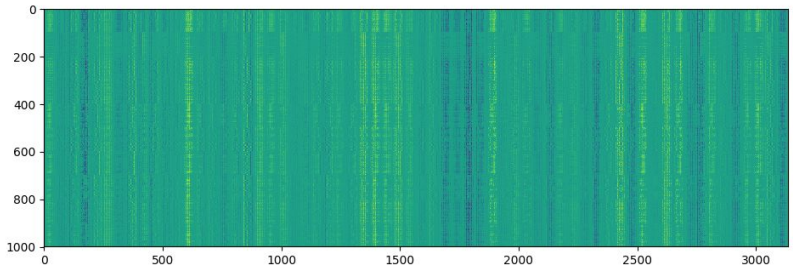


图 4.6

是否能从中看到 10 个条带，每个条带对应同一个数字的 100 个样本？再把两个全连接层的输出以同样的方式显示出来，是两个 1000 x 1000 的伪彩色图。如图 4.7 。

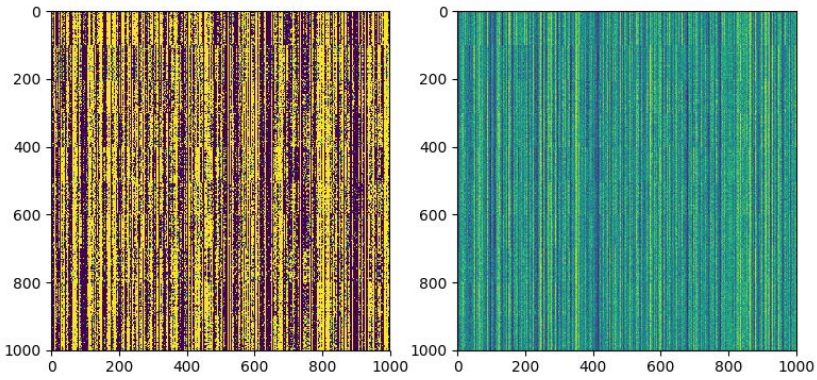


图 4.7

经过各卷积层、采样层和全连接层，信息表示的抽象程度逐层提高。CNN 就这样“认出”了手写数字。多层的 CNN 逐层提高了“逻辑深度”，这就是 “Deep Learning” 的含义。

最后把代码附上。CNN 实现使用的是 keras 库。数据集来自 kaggle： 这里。

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Flatten, Reshape, AveragePooling2D, Convolution2D, Act
from keras.utils.np_utils import to_categorical
from keras.utils.visualize_util import plot
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from keras.callbacks import Callback
from keras.optimizers import SGD


class LossHistory(Callback):
    def __init__(self):
        Callback.__init__(self)
        self.losses = []
        self.accuracies = []

    def on_train_begin(self, logs=None):
        pass

    def on_batch_end(self, batch, logs=None):
        self.losses.append(logs.get('loss'))
        self.accuracies.append(logs.get('acc'))


history = LossHistory()

data = pd.read_csv("train.csv")
data = data.sample(n=10000, replace=False)
digits = data[data.columns.values[1:]].values
labels = data.label.values

train_digit
```

```
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

model = Sequential()
model.add(Reshape(target_shape=(1, 28, 28), input_shape=(784,)))
model.add(
    Convolution2D(nb_filter=32, nb_row=3, nb_col=3, dim_ordering="th", border_mode="sa
model.add(AveragePooling2D(pool_size=(2, 2), dim_ordering="th"))
model.add(
    Convolution2D(nb_filter=64, nb_row=3, nb_col=3, dim_ordering="th", border_mode="sa
model.add(AveragePooling2D(pool_size=(2, 2), dim_ordering="th"))
model.add(Flatten())
model.add(Dense(output_dim=1000, activation="sigmoid"))
model.add(Dense(output_dim=1000, activation="sigmoid"))
model.add(Dense(output_dim=10, activation="linear"))

with open("digits_model.json", "w") as f:
    f.write(model.to_json())

plot(model, to_file="digits_model.png", show_shapes=True)

opt = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss="mse", optimizer=opt, metrics=["accuracy"])

model.fit(train_digits, train_labels_one_hot, batch_size=32, nb_epoch=10, callbacks=[h

model.save_weights("digits_model_weights.hdf5")

predict_labels = model.predict_classes(test_digits)

print(classification_report(test_labels, predict_labels))
print(accuracy_score(test_labels, predict_labels))
print(confusion_matrix(test_labels, predict_labels))
```

用 lena 图训练 sobel 算子的代码：

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.callbacks import Callback
from PIL import Image
import numpy as np
from scipy.ndimage.filters import convolve


class LossHistory(Callback):
    def __init__(self):
        Callback.__init__(self)
        self.losses = []

    def on_train_begin(self, logs=None):
        pass

    def on_batch_end(self, batch, logs=None):
        self.losses.append(logs.get('loss'))


lena = np.array(Image.open("lena.png").convert("L"))
lena_sobel
```

```
# sobel 算子。
sobel = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
])

# 计算卷积：用 sobel 算子滤波。结果保存在 lena_sobel 中。
convolve(input=lena, output=lena_sobel, weights=sobel, mode="constant", cval=1.0)

# 将像素值调整到 [0，255] 区间并保存 sobel 算子滤波后的 lena 图。
lena_tmp = np.uint8((lena_sobel - lena_sobel.min()) * 255 / (lena_sobel.max() - lena_s
Image.fromarray(lena_tmp).save("lena_sobel.png")

# 将原始 lena 图和 sobel 滤波 lena 图转换成 (1, 1, width, height) 尺寸。第一个 1 表示训练集
X = lena.reshape((1, 1) + lena.shape)
Y = lena_sobel.reshape((1, 1) + lena_sobel.shape)

# 建一个神经网络模型。
model = Sequential()

# 只添加一个卷积层。卷积层只有一个滤波器。滤波器尺寸 3x3 。输入维度顺序是 "th" 表示 (channel,
model.add(
    Convolution2D(nb_filter=1, nb_row=3, nb_col=3, dim_ordering="th", input_shape=X.sh
                  bias=False, init="uniform"))

# 代价函数取 mse 。优化算法取 rmsprop 。
model.compile(loss="mse", optimizer="rmsprop", metrics=["accuracy"])

history = LossHistory()

# 训练 10 轮，每轮保存一下当前网络输出图像。
for i in np.arange(0, 10):
    lena_tmp = model.predict(X).reshape(lena.shape)
    lena_tmp = np.uint8((lena_tmp - lena_tmp.min()) * 255 / (lena_tmp.max() - lena_tmp
    Image.fromarray(lena_tmp).save("lena_sobel_stage_{:d}.png".format(i))
    print("lena_sobel_stage_{:d}.png saved".format(i))

    model.fit(X, Y, batch_size=1, nb_epoch=200, verbose=1, callbacks=[history])
    print("Epoch {:d}".format(i + 1))

lena_tmp = model.predict(X).reshape(lena.shape)
lena_tmp = np.uint8((lena_tmp - lena_tmp.min()) * 255 / (lena_tmp.max() - lena_tmp.min
Image.fromarray(lena_tmp).save("lena_sobel_stage_final.png")
```

> 五、参考书目

最优化导论 (豆瓣)
book.douban.com

神经网络设计 (豆瓣)
book.douban.com

机器学习 (豆瓣)
book.douban.com

神经计算原理 (豆瓣)
book.douban.com

人工智能 复杂问题求解的结构和策略 英文版 第6版
book.douban.com

编辑于 2019-10-29

「真诚赞赏，手留余香」

赞赏

21 人已赞赏

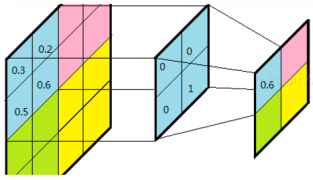
卷积神经网络（CNN） 神经网络 深度学习（Deep Learning）

文章被以下专栏收录

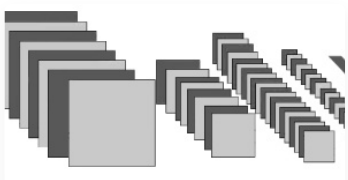
计算主义
“我不能建造者，我则没有真正理解”（费恩曼）

关注专栏

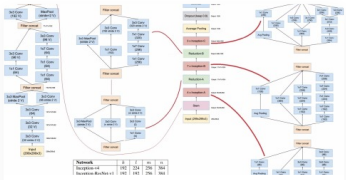
推荐阅读



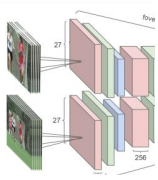
卷积神经网络CNN完全指南终极版（一）
沉迷学习的糕糕



CS231n课程笔记翻译：卷积神经网络笔记
猴子



YJango的卷积神经网络——介绍
YJango



卷积神经网络CI极版（二）
沉迷学习的...

89 条评论 ▲ 赞同 1089 ▼ 89 条评论 分享 收藏 ···

写下你的评论...

黄GW 2017-02-17
best ever!
赞

Phylosopher 2017-02-17
👍，非常清晰的简介
赞

qaqawwv 2017-02-17
条理分明
赞

lvboodvl 2017-02-17
讲得太好了😄
赞

cortex 2017-02-18
大神
赞

zahet 2017-02-18
写得好，让我一下就懂了卷积神经网络是什么了
6

原来ID可以改 2017-02-18
mark
赞

TyLIU 2017-02-18
必须大赞！先赞后看
赞

张某 回复 TyLIU 2017-02-18
卷积学过,不知道怎么用.看来只能在计算机实现.手算心算都完成不了.计算量太大啊
赞

岚少 2017-02-19
太棒了
赞

无名 2017-02-19
问一下哈，为什么第二层的卷积层是输入的32x28x28输出的还是32x28x28呢？如果第二层也是有32个卷积核的话那输出的不应该是32^2x28x28的吗？不知道我有没有表达清楚我的意思😂😂
赞

张觉非 (作者) 回复 无名 2017-02-19
我仔细看了一下。请看以下代码：

In [41]: cl2 = model.layers[2]

In [42]: cl2.input_shape

O

In [43]: cl2.output_shape

Out[43]: (None, 32, 28, 28)

到这里疑惑：卷积层 2 接受 32 个 28*28 大小的 channel 。给它指定的是 32 个滤波器，它应该每个输入 channel 输出 32 个channel 。输出维度应该是(1024=23^2,28,28)啊。

看一下卷积层 2 的权值：
In [49]: w = cl2.get_weights()

In [50]: len(w)

Out[50]: 1

In [51]: w[0].shape

Out[51]: (32, 32, 3, 3)

（构造的时候指定了不要 bias）权值列表的长度是 1 。唯一的元素的维度是(32, 32, 3, 3) 。第一个 32 对应着 32 个滤波器。后面的 32*3*3 看来是一个三维离散滤波器，把输入的 “一摞” 32 个 channel 的一个 3*3 区域一起滤，滤出一个 28*28 的 channel ，所以卷积层 2 还是输出 32 个 channel ，每一个 channel 是用一个 32*3*3 的滤波器把输入的 32 个 channel 摞一起滤出来的。

看来 keras 作者是这么想的：构造时候制定 32 个滤波器，输出就一定是 32 个 channel 。至于想达到咱们想象的 32*32 个channel ，恐怕还得自己想办法（先用，reshape 层，flatten层 之类处理处理）。

5

无名 回复 张觉非 (作者) 2017-02-19

谢谢啦～

赞

展开其他 1 条回复

张包峰 2017-02-20

参考书目2绝版了，请问哪里有的卖?

赞

张觉非 (作者) 回复 张包峰 2017-02-20

不好办了。我也是多年前买的了。那书讲解得特别好。要不孔夫子旧书网看看?

赞

生而无畏 回复 张包峰 2017-02-26

复制这条信息，打开手机淘宝即可看到【*神经网络设计 哈根 ,戴葵 机械工业出版社2005】￥xiLG9WqJqY￥c.b4za.com/h.WaYfEK?...

赞

NaNg 2017-02-21

谢谢作者，学习了!

赞

老蛤 2017-02-21

讲的很好

赞

秋水无涯 2017-02-22

顿时对

赞

Ge皮炖蛋汤 2017-02-22

非常非常好啊，能不能可视化下那64个滤波器

赞

张觉非 (作者) 回复 Ge皮炖蛋汤 2017-02-22

回头我搞搞看

赞

HTLiu 2017-02-23

"接着是第二个卷积层。它包含 64 个 32 x 3 x 3 的滤波" 这句话是不是有点不妥?个人感觉滤波还是[3,3]的?

赞

张觉非 (作者) 回复 HTLiu 2017-02-23

这应该是 keras 自己的实现。请看代码：

layer = model.layers[3]
<keras.layers.convolutional.Convolution2D at 0x10fa86eb8>

这个 layer 就是第二个卷积层对象。

weights = layer.get_weights()

得到这层的权重。

len(weights)
1

len 是 1 ，因为 weights 是一个 list ，第一个元素是权重列表，第二个元素是偏置值列表。例子中设定不要偏置，所以 weights 长度是 1 。

weights = weights[0]
weights.shape
(64, 32, 3, 3)

weights 维度是 (64, 32, 3, 3) 。64 肯定是指定的 64 个滤波器。3 x 3 肯定是平面上的 3 x 3 区域，这都没问题。关键是中间那个 32 。

如果 32 代表 32 个 3 x 3 滤波器，分别去滤上一层的 32 个 channel， 每一个滤出一幅 14 x 14 的图像，那么一个滤波器应该产生 32 x 24 x 14 的输出，64 个滤波器就该产生 64 x 32 x 14 x 14 维度的输出。但是这个卷积层的输出维度是 64 x 14 x 14 。

综上判断（没有读 keras 源码），我认为这一层应该是用一个 32 x 3 x 3 的滤波器去滤上一层给过来的 32 x 14 x 14 的（一摞）图像。相当于拿 32 x 3 x3 滤波器去滤 32 x 14 x 14 的 “图像体” 。

主要是要相吻合构造出来的 CNN 的各输入输出维度，第一卷积层 + 采样层给出来的 32 个 channel 肯定是要以某种方式 “合” 在一起，这样第二卷机层才会输出 64 x 14 x 14 而不是 64 x 32 x 14 x 14 。

没看源码，从表象上推理。您可以读读源码，如果这里说得有误，欢迎指正：)

赞

HTLiu 回复 张觉非 (作者) 2017-02-23

从代码实现上说，那个32其实就是多了一个维度。不过理论上那个32貌似应该指的是filter形成的feature map的每一个神经元与上一层32个3*3大小的区域的连接，而且 这个32是可以是任意变化。
以上是我的理解，不知道对不对。我也是看了LeNet的参数解释之后，才这样想的。参考

lyon 2017-02-26
卷积的从积分到求和不对吧
赞

张觉非 (作者) 回复 lyon 2017-02-26
请指教：）
赞

张觉非 (作者) 回复 lyon 2017-02-26
明白您意思了。写成离散式子应该是 SIGMA F(s, t) x G(x-s, y-t) delta s delta t （评论里没法输公式，就凑合了）。文中式子中少了 delta s 和 delta t 。是因为作为图片来讲自变量就是像素位置，离散化相当于一像素一采样，delta s 和 delts t 都等于 1 。所以式中省略了。

也就是离散化后相当于把一堆 “小柱子” 的体积加到一起而得到曲面下的体积。由于这些小柱子的底面积都是 delta s x delta t = 1 x 1 = 1 ，所以式子中把 “高” 加和就是体积。

您想指出的是这个问题吧?
赞

阿典 2017-02-26
学习了
赞

cha点点 2017-03-07
厉害了
赞

黄鼎 2017-03-07
看到过的关于CNN最清晰的解释了！
赞

1 2 3 4 下一页