第 15 章

卷积神经网络

本章从数学视角描述卷积神经网络 (Convolutional Neural Network, CNN)是如何运作的. 本章内容是自足的, 重点是让 CNN 领域的初学者也能够理解本章内容.

卷积神经网络 (CNN) 在计算机视觉、机器学习和模式识别的许多问题中已取得了卓越的性能. 关于这个话题, 已经发表了许多可靠的文献, 并有相当多高质量、开源的 CNN 软件包可供使用.

现在也有写得很好的 CNN 教程或者 CNN 软件手册. 然而, 我们相信仍然需要像本章这样专门为入门者准备的 CNN 介绍性材料. 研究论文通常很简洁并缺乏细节. 这使得初学者很难阅读这些论文. 目标对象为高级研究者的教程可能并未包含理解 CNN 如何运行的所有必要的细节.

本章的内容试图做到以下几点：

- 是自足的. 预计所有必需的数学背景知识已在本章 (或者本书的其他章节中) 被介绍过;
- 有所有推导步骤的细节. 本章试图解释所有必要的数学细节. 我们试着不跳过推导中任何重要的步骤. 因此, 初学者应该是能跟得上 (尽管专家可能会觉得本章有一点罗嗦.)
- 忽略实现细节. 本章的目的是为了使读者能够从数学层面上理解 CNN 是如何运行的. 我们将忽略那些实现细节. 在 CNN 中, 对不同的实现细节做出正确的选择是得到高准确率的关键因素之一 (即“细节决定成败”). 然而我们有意跳过了这部分内容, 以使读者专注于数学. 在理解了数学原理和细节后, 尝试 CNN 编程以获得亲身体验, 从而学得这些实现和设计的细节会更加有益. 本章的习题提供了 CNN 编程上手体验的机会.

CNN 在许多应用尤其是在与图像有关的任务中都很有用. CNN 的应用包括图像分类 (image classification)、图像语义分割 (image semantic segmentation)、图像中的目标检测 (object detection) 等. 在本章中我们将关注图像分类 (image classification 或 categorization). 在图像分类中, 每张图像有一个占据图像大部分面积的主要物体. 一幅图像根据这个主要物体被分为若干类别之一, 例如狗、飞机、鸟等.

15.1 预备知识

为了理解 CNN 如何运行, 我们从讨论一些必要的背景知识开始. 如果读者熟悉这些基

础知识, 可以跳过本节.

15.1.1　张量和向量化

大家对向量和矩阵都很熟悉. 我们使用粗体符号表示向量, 例如 $\boldsymbol{x} \in \mathbb{R}^D$ 是一个有 D 个元素的列向量. 我们使用大写字母表示矩阵, 例如 $X \in \mathbb{R}^{H \times W}$ 是一个有 H 行和 W 列的矩阵. 向量 $\boldsymbol{x}$ 也可看作是一个 1 列、D 行的矩阵.

这些概念能被推广到高阶的矩阵, 即张量 (tensor). 例如, $\boldsymbol{x} \in \mathbb{R}^{H \times W \times D}$ 是一个 3 阶 (order 3 或 third order) 张量. 它包含 HWD 个元素, 每个元素可以使用三元组 (i, j, d) 来索引, 其中 $0 \leqslant i < H$, $0 \leqslant j < W$, 以及 $0 \leqslant d < D$. 另一种理解 3 阶张量的视角是视其为包含了 D 个矩阵通道 (channel). 每个通道是一个大小为 $H \times W$ 的矩阵. 第一个通道包含了张量中由 $(i, j, 0)$ 索引的所有数字. 请注意, 在本章中我们假设索引从 0 而不是 1 开始. 当 $D = 1$ 时, 3 阶张量退化为矩阵.

我们已经和张量进行过日常交互. 标量 (scalar) 是零阶张量; 向量是 1 阶张量; 矩阵是 2 阶张量. 一张彩色图像事实上是一个 3 阶张量. 一幅 H 行 W 列的图像是大小为 $H \times W \times 3$ 的张量: 如果彩色图像是以 RGB 格式存储的, 它有 3 个通道 (分别对应 R、G 和 B), 每个通道是一个 $H \times W$ 的矩阵 (2 阶张量), 其包含了 R(或 G、或 B) 的所有像素值.

将图像 (或其他原始数据类型) 表示成张量是有益的. 在早期的计算机视觉和模式识别中, 由于相比张量而言我们更擅长处理矩阵, 彩色图像 (即 3 阶张量) 通常被转化为其灰度版本 (即矩阵). 在这个转化过程中颜色信息丢失了. 但是颜色在许多基于图像 (或视频) 的学习和识别问题中非常重要, 故我们确实需要以某种有条理的形式来处理颜色信息, 正如在 CNN 中那样.

张量在 CNN 中至关重要. CNN 中的输入、中间表示和参数都是张量. 高于 3 阶的张量也在 CNN 中被广泛使用. 例如, 我们很快将看到 CNN 中卷积层的卷积核构成了 4 阶张量.

给定一个张量, 我们可以按预先确定的顺序来排列其中所有的数字, 使其构成一个长向量. 例如, 在 Matlab / Octave 中, `(:)` 操作符按照列优先 (column first) 的顺序把一个矩阵转化为一个列向量. 一个例子是:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad A(:) = (1, 3, 2, 4)^T = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix}. \tag{15.1}$$

在数学中, 我们使用“vec”符号来表示该向量化操作. 也就是说, 在公式 (15.1) 的例子里, $\mathrm{vec}(A) = (1, 3, 2, 4)^T$. 为了向量化 3 阶张量, 我们可以先向量化其第一个通道 (这是一个矩阵, 并且我们知道如何将其向量化), 然后是第二个通道, $\cdots$, 直到所有的通道都被向量化. 然后, 3 阶张量的向量化结果就是其所有通道的向量化结果按照此顺序的拼接 (concatenation).

3 阶张量的向量化是一个递归的过程, 其利用了 2 阶张量的向量化. 这种递归的过程可以按相同的方式用于 4 阶 (或甚至更高阶) 的张量的向量化.

15.1.2　向量微积分和链式法则

CNN 的学习过程依赖于向量微积分和链式法则. 假设 z 是一个标量 (即 $z \in \mathbb{R}$), $\boldsymbol{y} \in \mathbb{R}^H$ 是一个向量. 如果 z 是 $\boldsymbol{y}$ 的函数, 那么 z 对 $\boldsymbol{y}$ 的偏导是一个向量, 定义为

$$\left[\frac{\partial z}{\partial \boldsymbol{y}}\right]_i = \frac{\partial z}{\partial y_i}. \tag{15.2}$$

换言之, $\dfrac{\partial z}{\partial \boldsymbol{y}}$ 是一个和 $\boldsymbol{y}$ 有相同大小的向量, 它的第 i 个元素是 $\dfrac{\partial z}{\partial y_i}$. 同时请注意

$$\frac{\partial z}{\partial \boldsymbol{y}^T} = \left(\frac{\partial z}{\partial \boldsymbol{y}}\right)^T.$$

此外, 假设 $\boldsymbol{x} \in \mathbb{R}^W$ 是另一个向量, 并且 $\boldsymbol{y}$ 是 $\boldsymbol{x}$ 的函数. 那么, $\boldsymbol{y}$ 对 $\boldsymbol{x}$ 的偏导被定义为

$$\left[\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T}\right]_{ij} = \frac{\partial y_i}{\partial x_j}. \tag{15.3}$$

该偏导是一个 $H \times W$ 的矩阵, 第 i 行和第 j 列交叉处的元素是 $\dfrac{\partial y_i}{\partial x_j}$.

容易发现, 以链式的形式, z 是 $\boldsymbol{x}$ 的函数: 一个函数将 $\boldsymbol{x}$ 映射到 $\boldsymbol{y}$, 另外一个函数将 $\boldsymbol{y}$ 映射到 z. 链式法则 (chain rule) 可被用于计算 $\dfrac{\partial z}{\partial \boldsymbol{x}^T}$ 如下

$$\frac{\partial z}{\partial \boldsymbol{x}^T} = \frac{\partial z}{\partial \boldsymbol{y}^T}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T}. \tag{15.4}$$

公式 (15.4) 的一个合理性检查的方式是检查矩阵/向量的维度. 注意到 $\dfrac{\partial z}{\partial \boldsymbol{y}^T}$ 是一个有 H 个元素的行向量, 或 $1 \times H$ 的矩阵. (提醒一下, $\dfrac{\partial z}{\partial \boldsymbol{y}}$ 是一个列向量). 由于 $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^T}$ 是一个 $H \times W$ 的矩阵, 它们之间的向量/矩阵乘法是有效的, 且其结果应当是一个有 W 个元素的行向量, 这和 $\dfrac{\partial z}{\partial \boldsymbol{x}^T}$ 的维度匹配.

如果想了解计算向量和矩阵偏导的具体规则, 请参阅第 2 章和《The Matrix Cookbook》[183].

15.2　CNN 概览

在本节中, 我们将从抽象的层面讨论 CNN 是如何训练和预测的, 而细节留在后续的小节给出.

15.2.1　结构

CNN 通常以一个 3 阶张量作为其输入, 例如一幅 H 行、W 列、3 个通道 (R、G、B 颜色通道) 的图像. 然而, CNN 可以用相似的方式处理更高阶的张量输入. 随后, 输入经过一系列的处理. 一个处理步骤通常被称为一层 (layer), 可以是卷积层 (convolution layer)、汇合层 (pooling layer)、规范化层 (normalization layer)、全连接层 (fully connected layer)、损失层 (loss layer) 等.

在本章后续部分我们将介绍这些层的细节. 我们将详细介绍三种层: 卷积、汇合和 ReLU, 它们是几乎所有 CNN 模型的关键组成部分. 适当的规范化, 例如批量规范化 (batch normalization) 在学习好的 CNN 参数的优化过程中很重要. 尽管它们没有在本章中介绍, 但我们将在习题中给出一些相关的资源.

现在让我们首先给出 CNN 结构的一种抽象描述.

$$\boldsymbol{x}^1 \longrightarrow \boxed{\boldsymbol{w}^1} \longrightarrow \boldsymbol{x}^2 \longrightarrow \cdots \longrightarrow \boldsymbol{x}^{L-1} \longrightarrow \boxed{\boldsymbol{w}^{L-1}} \longrightarrow \boldsymbol{x}^L \longrightarrow \boxed{\boldsymbol{w}^L} \longrightarrow z \tag{15.5}$$

上述公式 (15.5) 阐述了 CNN 在前向过程 (forward pass) 中是如何逐层运行的. 输入是 $\boldsymbol{x}^1$, 通常是一幅图像 (3 阶张量). 它经过第一层, 也就是第一个方框的处理. 我们将第一层的处理中牵涉到的所有参数统一记为张量 $\boldsymbol{w}^1$. 第一层的输出是 $\boldsymbol{x}^2$, 它同时也作为第二层处理的输入. 这个处理过程一直向前, 直到 CNN 中所有的层都已处理结束, 那时输出 $\boldsymbol{x}^L$.

然而, 一个额外的层被添加到误差反向传播 (backward error propagation)过程中, 这是一种在 CNN 中学到好的参数值的方法. 让我们假设手头的问题是有 C 个类别的图像分类问题. 一种常用的策略是将 $\boldsymbol{x}^L$ 输出为一个 C 维的向量, 其第 i 个元素编码了预测值 ($\boldsymbol{x}^1$ 来自第 i 个类的后验概率). 为了使得 $\boldsymbol{x}^L$ 成为一个概率质量函数, 我们可以置第 $(L-1)$ 层的处理为对 $\boldsymbol{x}^{L-1}$ 的 softmax 变换 (参见第 9 章). 在其他应用中, 输出 $\boldsymbol{x}^L$ 还可以有其他的形式和解释.

最后一层是损失层 (loss layer). 让我们假设 $\boldsymbol{t}$ 是与输入 $\boldsymbol{x}^1$ 对应的目标值 (真实值), 那么一个代价 (cost) 或损失 (loss) 函数可被用于衡量 CNN 的预测 $\boldsymbol{x}^L$ 和目标 $\boldsymbol{t}$ 之间的不一致程度. 例如, 尽管通常使用更复杂的损失函数, 一个简单的损失函数可以是

$$z = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{x}^L\|^2 . \tag{15.6}$$

该平方 ℓ_2 损失可被用于回归问题中.

在分类问题中, 经常使用交叉熵 (参见第 10 章) 损失. 分类问题的真实值是一个类别变量 t. 我们首先将类别变量 t 转换为一个 C 维的向量 $\boldsymbol{t}$(参见第 9 章). 现在 $\boldsymbol{t}$ 和 $\boldsymbol{x}^L$ 都是概率质量函数, 那么交叉熵损失就度量了它们之间的距离. 因此, 我们可以最小化交叉熵损失 (cross entropy loss). 公式 (15.5) 用损失层显式地对损失函数进行建模, 尽管在很多情形下损失层不包含任何参数, 即 $\boldsymbol{w}^L = \emptyset$, 它的处理仍用一个带参数 $\boldsymbol{w}^L$ 的方框描述.

请注意, 还有其他的层不包含任何参数, 也就是说, 对一些 $i < L$ 来说, $\boldsymbol{w}^i$ 可能是空的. softmax 层就是这样的一个例子. 该层可以将一个向量转化为一个概率质量函数. softmax 层的输入是一个向量, 其中的值可能是正、零或负的. 假设第 l 层是 softmax 层, 其输入是一个向量 $\boldsymbol{x}^l \in \mathbb{R}^d$. 那么, 它的输出是一个向量 $\boldsymbol{x}^{l+1} \in \mathbb{R}^d$, 由下式计算

$$x_i^{l+1} = \frac{\exp(x_i^l)}{\sum_{j=1}^{d} \exp(x_j^l)} , \tag{15.7}$$

即输入在经 softmax 变换之后的版本. 在 softmax 层处理之后, $\boldsymbol{x}^{l+1}$ 的值构成了一个概率质量函数, 可被用作交叉熵损失的输入.

15.2.2　前向运行

假设某 CNN 模型中所有的参数 $\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^{L-1}$ 都已被学得, 那么我们就可以使用这个模型进行预测. 预测只牵涉到前向运行 CNN 模型, 即沿着公式 (15.5) 的箭头方向进行.

让我们用图像分类来作为一个例子. 从输入 $\boldsymbol{x}^1$ 开始, 我们让其通过第一层 (那个有参数 $\boldsymbol{w}^1$ 的方框) 的处理, 得到 $\boldsymbol{x}^2$. 随后, $\boldsymbol{x}^2$ 通过第二层, 并继续向前. 最终, 我们得到 $\boldsymbol{x}^L \in \mathbb{R}^C$, 它估计了 $\boldsymbol{x}^1$ 属于 C 个类别的后验概率. 我们可以输出 CNN 的预测为

$$\arg\max_i x_i^L \,. \tag{15.8}$$

请注意, 损失层在预测时并不需要. 它只在我们利用训练样例的集合来学习 CNN 参数时有用. 那么, 问题来了: 我们该如何学习模型的参数呢?

15.2.3　随机梯度下降

和在许多其他学习系统中一样, CNN 模型的参数通过最小化损失 z 来进行优化, 也就是说, 我们想要 CNN 模型的预测去匹配训练集的真实标记.

让我们假设一个训练样例 $\boldsymbol{x}^1$ 被用来学习这些参数. 训练过程涉及在两个方向上运行 CNN 网络. 我们首先使用当前的 CNN 参数前向运行这个网络得到预测 $\boldsymbol{x}^L$. 我们需要将其与 $\boldsymbol{x}^1$ 对应的目标 $\boldsymbol{t}$ 进行比较, 而不是输出这个预测, 也就是说, 得继续运行前向过程直到最后的损失层. 最终, 我们得到损失 z.

那么损失 z 就是一个监督信号, 指导模型的参数应该如何进行修改 (更新). 随机梯度下降 (Stochastic Gradient Descent, SGD) 按照如下方式修改参数:

$$\boldsymbol{w}^i \longleftarrow \boldsymbol{w}^i - \eta \frac{\partial z}{\partial \boldsymbol{w}^i} \,. \tag{15.9}$$

关于符号的一个备注. 在大多数 CNN 材料中, 上标表示“时间”(例如训练的轮数). 但在本章中, 我们使用上标表示层的索引. 请不要混淆. 我们不使用额外的索引来表示时间. 在公式 (15.9) 中, $\longleftarrow$ 符号隐式地表达了 (第 i 层的) 参数 $\boldsymbol{w}^i$ 是从时刻 t 更新到时刻 $t+1$ 的. 如果显式地使用时间索引 t, 该公式将变成这样

$$\left(\boldsymbol{w}^i\right)^{t+1} = \left(\boldsymbol{w}^i\right)^t - \eta \frac{\partial z}{\partial \left(\boldsymbol{w}^i\right)^t} \,. \tag{15.10}$$

在公式 (15.9) 中, 偏导 $\dfrac{\partial z}{\partial \boldsymbol{w}^i}$ 衡量了当 $\boldsymbol{w}^i$ 的各维变化时, 与之对应的 z 增加的速率. 这个偏导向量在数学优化中被称为梯度(gradient). 因此, 在 $\boldsymbol{w}^i$ 的当前值周围一个小的领域中, 沿着由梯度指定的方向移动 $\boldsymbol{w}^i$ 将增大目标值 z. 为了最小化损失函数, 我们应当沿着梯度的反方向更新 $\boldsymbol{w}^i$. 这个更新规则被称为梯度下降 (gradient descent). 梯度下降的描述见图 15.1, 其中梯度由 $\boldsymbol{g}$ 表示.

然而, 如果我们沿着梯度的反方向移动得太远, 损失函数可能会上升. 因此, 在每次更新时我们只用负梯度的一小部分来改变参数, 这由 η 来控制, η 被称为学习率 (learning rate). 在深度神经网络学习中, $\eta > 0$ 通常被置为一个很小的数 (例如 $\eta = 0.001$).

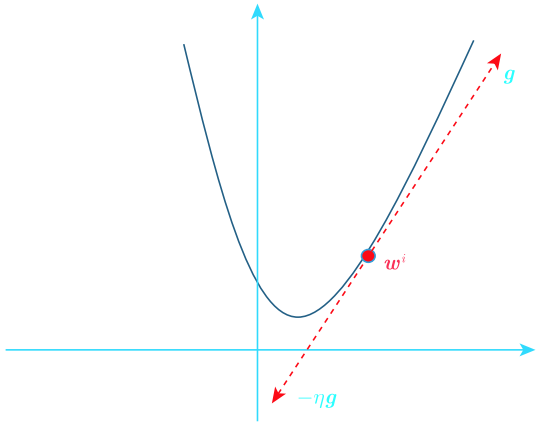


图 15.1　梯度下降方法图示, 其中 η 是学习率 (见彩插)

如果学习率不是过大的话, 基于 $\boldsymbol{x}^1$ 的一次更新将使得针对这个特定训练样例的损失变小. 然而, 它很可能使得其他一些样例的损失增大. 因此, 我们需要使用全部的训练样例来更新参数. 当所有的训练样例都已被用来更新参数时, 我们称处理过一轮 (epoch).

一轮训练通常将减小训练集上的平均损失, 直到学习系统开始过拟合 (overfit) 训练数据. 因此, 我们可以重复梯度下降更新多轮, 并在某点停止, 得到 CNN 的参数 (例如, 当在验证集上的平均损失增大时我们可以停止).

梯度下降可能在其数学形式上 (公式 15.9) 看起来颇为简单, 但是在实际中是一个很有技巧性的操作. 例如, 如果我们使用只基于单个训练样例计算得到的梯度去更新参数, 我们将观测到一个不稳定的损失函数: 所有训练样例的平均损失将以很高频率上下振荡跳跃. 这是由于梯度只用单个训练样例而不是整个训练集估计得到——基于单个样例计算得到的梯度可以极度不稳定.

和基于单个样例的参数更新相反, 我们可以使用全部训练样例来计算梯度并更新参数. 然而, 由于参数在一轮中只更新一次, 这种批量(batch) 处理策略需要大量的计算资源, 因而很不实用, 尤其是当训练样例数目很大时.

一个折中的方案是使用训练样例的小批量(mini-batch), 使用小批量计算梯度, 并据此更新参数. 使用训练样例的一个 (通常是) 小的子集来估计梯度并进行参数更新被称为随机梯度下降(stochastic gradient descent). 例如, 我们可以令一个小批量为 32 或 64 个样例. (使用小批量策略的) 随机梯度下降是学习 CNN 参数的主流方法. 我们也想提醒, 当使用小批量时, CNN 的输入变成了一个 4 阶张量, 例如当小批量大小是 32 时, 输入的大小是 $H \times W \times 3 \times 32$.

至此, 一个新的问题浮出了水面: 该如何计算梯度呢? 这看上去是一个非常复杂的任务.

15.2.4　误差反向传播

最后一层的偏导是容易计算的. 由于 $\boldsymbol{x}^L$ 通过参数 $\boldsymbol{w}^L$ 的控制直接和 z 相连, 计算 $\dfrac{\partial z}{\partial \boldsymbol{w}^L}$ 是容易的. 这一步骤只有当 $\boldsymbol{w}^L$ 不为空时才需要. 使用相同的思路, 计算 $\dfrac{\partial z}{\partial \boldsymbol{x}^L}$ 也是容易的.

例如, 如果使用平方 ℓ_2 损失, 我们得到一个空的 $\dfrac{\partial z}{\partial \boldsymbol{w}^L}$, 并且

$$\frac{\partial z}{\partial \boldsymbol{x}^L} = \boldsymbol{x}^L - \boldsymbol{t}\,.$$

事实上, 对每个层, 我们计算两组梯度: z 对该层参数 $\boldsymbol{w}^i$ 的偏导, 和对该层输入 $\boldsymbol{x}^i$ 的偏导.

- 项 $\dfrac{\partial z}{\partial \boldsymbol{w}^i}$, 如我们在公式 (15.9) 中所见到的, 可被用于更新当前层 (第 i 层) 的参数;
- 项 $\dfrac{\partial z}{\partial \boldsymbol{x}^i}$ 可被用于反向更新参数, 例如更新到第 $(i-1)$ 层. 一个直观的解释是: $\boldsymbol{x}^i$ 是第 $(i-1)$ 层的输出, $\dfrac{\partial z}{\partial \boldsymbol{x}^i}$ 指导 $\boldsymbol{x}^i$ 应当如何变化从而减小损失函数. 因此, 我们可以将 $\dfrac{\partial z}{\partial \boldsymbol{x}^i}$ 看作是由 z 开始、以一层接一层的方式反向传播到当前层的那一部分“误差”监督信息. 故我们可以继续反向传播过程, 并使用 $\dfrac{\partial z}{\partial \boldsymbol{x}^i}$ 反向传播误差到第 $(i-1)$ 层.

这个逐层反向更新的过程使得学习 CNN 变得容易了许多. 事实上, 在 CNN 之外, 这个策略也是其他类型神经网络中的标准做法, 被称为误差反向传播 (error back propagation), 或简称为反向传播 (back propagation).

让我们用第 i 层作为一个例子. 当我们更新第 i 层时, 第 $(i+1)$ 层的反向传播过程应该已经完成了. 也就是说, 我们已经计算出项 $\dfrac{\partial z}{\partial \boldsymbol{w}^{i+1}}$ 和 $\dfrac{\partial z}{\partial \boldsymbol{x}^{i+1}}$. 这两者都已被存储, 随时可供使用.

现在我们的任务是计算 $\dfrac{\partial z}{\partial \boldsymbol{w}^i}$ 和 $\dfrac{\partial z}{\partial \boldsymbol{x}^i}$. 使用链式法则, 我们有

$$\frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)} = \frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^{i+1})^T)} \frac{\partial\, \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}\,, \tag{15.11}$$

$$\frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)} = \frac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^{i+1})^T)} \frac{\partial\, \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}\,. \tag{15.12}$$

由于 $\dfrac{\partial z}{\partial \boldsymbol{x}^{i+1}}$ 已被计算并已在存储器中, 它只需要一个矩阵变形操作 (vec) 以及一个额外的转置操作以得到 $\dfrac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^{i+1})^T)}$, 就是两个公式等号右边的第一项. 只要我们能够计算 $\dfrac{\partial\, \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}$ 和 $\dfrac{\partial\, \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}$, 就可以轻而易举地得到我们想要的结果 (两个公式等号的左边那项).

因 $\boldsymbol{x}^i$ 和 $\boldsymbol{x}^{i+1}$ 通过带参数 $\boldsymbol{w}^i$ 的函数直接关联, 相比直接计算 $\dfrac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}$ 和 $\dfrac{\partial z}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}$ 而言, 计算 $\dfrac{\partial\, \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{w}^i)^T)}$ 和 $\dfrac{\partial\, \mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial (\mathrm{vec}(\boldsymbol{x}^i)^T)}$ 要容易得多. 不同层的这些偏导计算的细节将在下面的各个小节中进行讨论.

15.3　层的输入、输出和符号

既然 CNN 的结构已经清楚了, 下面我们将详细讨论不同类型的层. 将从 ReLU 层开始, 它是本章中我们要讨论的各种层之中最为简单的. 但是在开始之前, 我们需要进一步细化我

们的符号.

我们考虑第 l 层, 其输入构成了一个 3 阶张量 $\boldsymbol{x}^l$, 其中 $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$. 那么, 我们需要一个三元组索引集 (i^l, j^l, d^l) 来定位 $\boldsymbol{x}^l$ 中的任意某个元素. 三元组 (i^l, j^l, d^l) 对应于 $\boldsymbol{x}^l$ 中第 d^l 个通道里空间位置是 (i^l, j^l) (在第 i^l 行、第 j^l 列) 的那个元素. 在实际的 CNN 学习中通常使用小批量策略. 在那种情况下, $\boldsymbol{x}^l$ 变为 4 阶张量 $\mathbb{R}^{H^l \times W^l \times D^l \times N}$, 其中 N 是小批量的大小. 为了简化, 在本章中我们假设 $N = 1$. 然而, 本章的结果容易推广到小批量版本.

为了简化后续出现的符号, 我们采用从 0 开始的索引惯例, 这就规定了 $0 \leqslant i^l < H^l$、$0 \leqslant j^l < W^l$ 及 $0 \leqslant d^l < D^l$.

在第 l 层, 一个函数将输入 $\boldsymbol{x}^l$ 变换成输出 $\boldsymbol{y}$, 其也是下一层的输入. 因此, 我们注意到 $\boldsymbol{y}$ 和 $\boldsymbol{x}^{l+1}$ 事实上指向了同一个对象, 牢记这一点会是很有帮助的. 我们假设输出大小是 $H^{l+1} \times W^{l+1} \times D^{l+1}$, 输出的一个元素由三元组 $(i^{l+1}, j^{l+1}, d^{l+1})$ 索引, $0 \leqslant i^{l+1} < H^{l+1}$、$0 \leqslant j^{l+1} < W^{l+1}$、$0 \leqslant d^{l+1} < D^{l+1}$.

15.4　ReLU 层

ReLU 层不改变输入的大小, 也就是说, $\boldsymbol{x}^l$ 和 $\boldsymbol{y}$ 共享相同的大小. 事实上, 修正线性单元(Rectified Linear Unit, ReLU)可以被认为是对输入张量的每个元素独立地进行截断:

$$y_{i,j,d} = \max\{0, x^l_{i,j,d}\}, \tag{15.13}$$

其中 $0 \leqslant i < H^l = H^{l+1}$、$0 \leqslant j < W^l = W^{l+1}$ 且 $0 \leqslant d < D^l = D^{l+1}$.

ReLU 层内部没有参数, 因此本层不需要进行参数学习.

根据公式 (15.13), 显然

$$\frac{\mathrm{d}y_{i,j,d}}{\mathrm{d}x^l_{i,j,d}} = \left[\!\left[x^l_{i,j,d} > 0\right]\!\right], \tag{15.14}$$

其中 $[\![\cdot]\!]$ 是指示函数, 当参数为真时取值为 1, 否则为 0.

因此, 我们有

$$\left[\frac{\partial z}{\partial \boldsymbol{x}^l}\right]_{i,j,d} = \begin{cases} \left[\dfrac{\partial z}{\partial \boldsymbol{y}}\right]_{i,j,d} & \text{若 } \boldsymbol{x}^l_{i,j,d} > 0 \\ 0 & \text{否则} \end{cases}. \tag{15.15}$$

请注意, $\boldsymbol{y}$ 是 $\boldsymbol{x}^{l+1}$ 的别名.

严格地讲, 函数 $\max(0, x)$ 在 $x = 0$ 处不可微, 因此公式 (15.14) 在理论上会有一点毛病. 而在实际中这并不成为问题, ReLU 用起来很安全.

ReLU 的作用是增加 CNN 中的非线性. 因为图像中的语义信息 (例如一个人和一只哈士奇狗在公园的长椅上相邻而坐) 显然是输入的像素值经过高度非线性的映射后得到的, 我们希望从 CNN 的输入到其输出的映射也是高度非线性的. 尽管简单, ReLU 函数是一个非线性函数, 如图 15.2 所示.

我们可以将 $x^l_{i,j,d}$ 看作是由 CNN 的第 1 层到第 $l-1$ 层提取的 $H^l W^l D^l$ 个特征(feature)之一, 可以是正、零或负的. 例如, 如果输入图像中的某个区域包含了特定的模式 (比如狗的头, 或者猫的头, 或者其他一些相似的模式), $x^l_{i,j,d}$ 可能是正的; 当那个区域没有展现出这些

模式时, $x^l_{i,j,d}$ 是负的或者是零. ReLU 层将所有负值都置为零, 这意味着: 只有对那个特定区域拥有这些模式的图像, $y^l_{i,j,d}$ 才会被激活(activated).

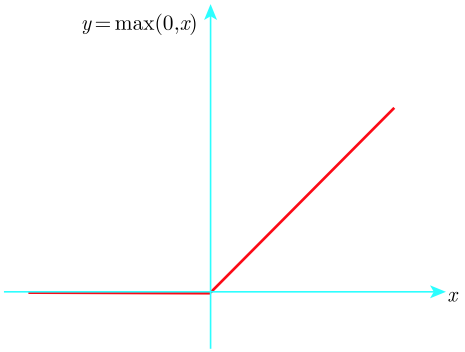


图 15.2　ReLU 函数 (见彩插)

从直觉上讲, 该性质对识别复杂的模式和物体很有用. 例如, 如果一个特征被激活并且那个特征的模式看上去像猫的头, 这仅仅是支撑“输入图像包含一只猫”的一个弱的证据. 然而, 如果我们在 ReLU 层之后找到许多激活的特征, 其目标模式对应于猫的头、躯干、毛发、腿等, (在第 $l+1$ 层时) 我们有更高的信心认为输入图像中很可能有一只猫.

其他的非线性变换曾经被神经网络社区用来增加非线性, 例如对数几率 sigmoid 函数

$$y = \sigma(x) = \frac{1}{1+\exp(-x)}\,.$$

然而, 对数几率 sigmoid 函数在 CNN 学习时显著地劣于 ReLU. 请注意, 如果使用 sigmoid 函数, 则 $0 < y < 1$ 并且

$$\frac{\mathrm{d}y}{\mathrm{d}x} = y(1-y)\,,$$

故我们有

$$0 < \frac{\mathrm{d}y}{\mathrm{d}x} \leqslant \frac{1}{4}\,.$$

因此, 在误差反向传播过程里, 梯度 $\dfrac{\partial z}{\partial x} = \dfrac{\partial z}{\partial y}\dfrac{\mathrm{d}y}{\mathrm{d}x}$ 将有比 $\dfrac{\partial z}{\partial y}$ 小得多的幅度 (至多 $\dfrac{1}{4}$ 倍). 换言之, sigmoid 层将导致梯度的大小显著下降, 在经过若干层 sigmoid 层之后, 梯度将消失(vanish)(即所有的分量将接近 0). 消失的梯度使得基于梯度的学习 (例如 SGD) 异常困难. sigmoid 的另一个主要缺点是它会饱和 (saturated). 当 x 的幅度很大时, 例如当 $x > 6$ 或 $x < -6$ 时, 对应的梯度几乎是 0.

另一方面, ReLU 层将第 l 层中一些特征的梯度置零, 但是这些特征没有被激活 (换句话说, 我们不关心它们). 对那些激活的特征, 梯度不需要任何变化就可回传, 这对 SGD 学习很有益处. 引入 ReLU 来取代 sigmoid 是 CNN 中一个重要的变化, 它显著地降低了学习 CNN 参数的难度并提升了准确率. ReLU 有许多更为复杂的变体, 例如参数化 ReLU(parametric ReLU) 和指数线性单元 (exponential linear unit), 在本章中我们不涉及这些内容.

15.5　卷积层

下面我们转向卷积层 (convolution layer), 这是本章中所介绍的最为复杂的一个层. 这也是 CNN 中最重要的一个层, 因此其被命名为卷积神经网络.

15.5.1　什么是卷积?

让我们从使用单个卷积核对一个矩阵进行卷积 (convolution) 开始. 假设输入图像大小是 3×4, 卷积核大小是 2×2, 如图 15.3 所示.

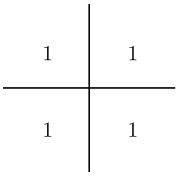
如果我们将卷积核 (convolution kernel)覆盖在输入图像上面, 就可以计算在卷积核和输入图像重合位置的数字的乘积, 将这些乘积累加起来, 可以得到单个数字. 例如, 如果我们把卷积核覆盖在输入的左上角, 在那个空间位置的卷积结果是

$$1 \times 1 + 1 \times 4 + 1 \times 2 + 1 \times 5 = 12\,.$$

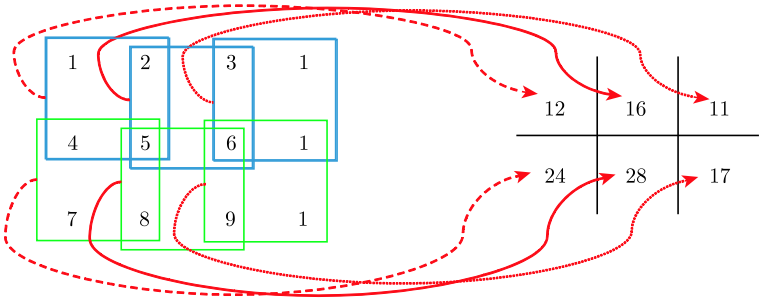
我们随后把卷积核向下移动一个像素并得到下一个卷积结果为

$$1 \times 4 + 1 \times 7 + 1 \times 5 + 1 \times 8 = 24\,.$$

我们继续向下移动卷积核, 直到它到达输入矩阵 (图像) 的底部边界. 接着我们将卷积核返回到顶部, 并将卷积核向右移动一个元素 (像素). 对每个可能的像素位置重复这个卷积操作, 直到我们将卷积核移动到输入图像的右下角, 如图 15.3 所示.



(a) 一个2×2的卷积核



(b) 卷积的输入和输出

图 15.3　卷积操作图示 (见彩插)

针对 3 阶张量, 卷积操作的定义是相似的. 假设第 l 层的输入是大小为 $H^l \times W^l \times D^l$ 的 3 阶张量. 一个卷积核也是一个 3 阶张量, 大小为 $H \times W \times D^l$. 当我们将卷积核覆盖在输入张量的空间位置 $(0,0,0)$ 上时, 我们计算所有 D^l 个通道中对应元素的乘积, 并将这 HWD^l 个乘积相加, 从而得到这个空间位置的卷积结果. 然后, 我们从上到下、从左向右地移动卷积核, 完成整个卷积.

在一个卷积层中通常会使用许多卷积核. 假设使用 D 个卷积核, 每个核的空间大小是 $H \times W$, 我们记所有这些卷积核为 $\boldsymbol{f}$. $\boldsymbol{f}$ 是空间 $\mathbb{R}^{H\times W\times D^l\times D}$ 中的一个 4 阶张量. 相似地, 我们使用索引变量 $0 \leqslant i < H$、$0 \leqslant j < W$、$0 \leqslant d^l < D^l$ 和 $0 \leqslant d < D$ 来定位卷积核中某个特定的元素. 同时也请注意, 卷积核的集合 $\boldsymbol{f}$ 与公式 (15.5) 中的 $\boldsymbol{w}^l$ 记号指向相同的对象. 在这里我们对符号稍作改变, 以使得推导稍微简单一点. 同样显然的是, 即便使用小批量策略, 卷积核保持不变.

如图 15.3 所示, 只要卷积核比 1×1 大, 输出的空间大小将小于输入的空间大小. 有时我们需要输入和输出图像有相同的高和宽, 可以使用一个简单的填补 (padding) 技巧. 如果输入是 $H^l \times W^l \times D^l$, 卷积核是 $H \times W \times D^l \times D$, 卷积结果的大小是

$$(H^l - H + 1) \times (W^l - W + 1) \times D\,.$$

对输入的每个通道, 如果我们在第一行上方填补(即插入)$\left\lfloor \dfrac{H-1}{2} \right\rfloor$ 行, 最后一行下方填补 $\left\lfloor \dfrac{H}{2} \right\rfloor$ 行, 并在第一列左边填补 $\left\lfloor \dfrac{W-1}{2} \right\rfloor$ 列, 最后一列右边填补 $\left\lfloor \dfrac{W}{2} \right\rfloor$ 列, 卷积输出的大小将是 $H^l \times W^l \times D$, 也就是说, 与输入的空间大小相同. $\lfloor\cdot\rfloor$ 是向下取整函数 (也称为地板函数, floor function). 填补的行和列的元素通常设为 0, 但是其他值也是可能的.

步长(stride) 是卷积中的另一个重要概念. 如图 15.3 所示, 对每个可能的空间位置, 我们都使用卷积核对输入进行卷积, 这对应于步长 $s = 1$. 然而, 如果 $s > 1$, 卷积核的每次移动将跳过 $s - 1$ 个像素位置 (即卷积只在每隔 s 个水平和垂直位置进行). 当 $s > 1$ 时, 卷积的输出将比输入小很多——H^{l+1} 和 W^{l+1} 大约分别是 H^l 和 W^l 的 $1/s$ 倍.

在本节中, 我们考虑步长为 1 并且没有填补的简单情况. 因此, 我们知道 $\boldsymbol{y}$(或者 $\boldsymbol{x}^{l+1}$) 是在 $\mathbb{R}^{H^{l+1}\times W^{l+1}\times D^{l+1}}$ 中, 其中 $H^{l+1} = H^l - H + 1$、$W^{l+1} = W^l - W + 1$ 且 $D^{l+1} = D$.

使用精确的数学语言, 卷积操作可被表达为如下的公式:

$$y_{i^{l+1},j^{l+1},d} = \sum_{i=0}^{H-1}\sum_{j=0}^{W-1}\sum_{d^l=0}^{D^l-1} f_{i,j,d^l,d} \times x^l_{i^{l+1}+i,j^{l+1}+j,d^l}\,. \tag{15.16}$$

公式 (15.16) 对所有 $0 \leqslant d < D = D^{l+1}$, 以及满足

$$0 \leqslant i^{l+1} < H^l - H + 1 = H^{l+1}\,, \tag{15.17}$$

$$0 \leqslant j^{l+1} < W^l - W + 1 = W^{l+1} \tag{15.18}$$

的任何空间位置 (i^{l+1}, j^{l+1}) 进行重复. 在这个公式中, $x^l_{i^{l+1}+i,j^{l+1}+j,d^l}$ 表示 $\boldsymbol{x}^l$ 的一个元素, 由三元组 $(i^{l+1}+i, j^{l+1}+j, d^l)$ 索引.

偏置项 b_d 通常被加到 $y_{i^{l+1},j^{l+1},d}$ 上. 在本章中, 为了描述更清晰, 我们省略了这一项.

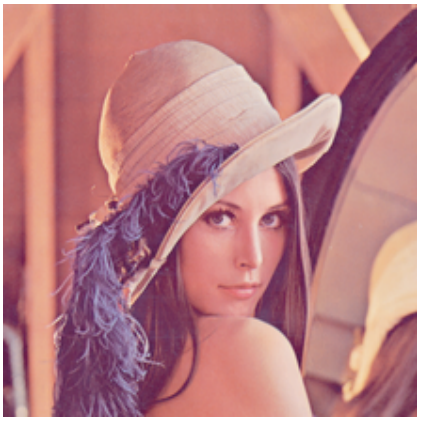
15.5.2　为什么要进行卷积?

图 15.4 展示了一个彩色输入图像 (图 15.4a) 和使用两个不同卷积核 (图 15.4b 和

图 15.4c) 的卷积结果. 使用了一个 3×3 的卷积矩阵

$$K = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

卷积核的大小应该是 $3 \times 3 \times 3$, 其中我们将每个通道都置为 K. 当在位置 (x, y) 有一个水平边时 (即当空间位置 $(x+1, y)$ 和 $(x-1, y)$ 的像素值有很大的差异时), 我们期望卷积的结果有很高的数值. 如图 15.4b 所示, 卷积结果确实高亮了水平边. 当我们置卷积核的每个通道为 K^T (K 的转置) 时, 卷积结果放大垂直边缘, 如图 15.4c 所示. 矩阵 (或滤波器)K 和 K^T 被称为 Sobel 算子 (Sobel operator)[⊖] .



(a) Lenna

(b) 水平边缘

(c) 垂直边缘

图 15.4　Lenna 图像和不同卷积核的效果 (见彩插)

如果我们向卷积运算中加入偏置项, 我们可以使得在特定方向的水平 (垂直) 边缘的卷积结果为正 (例如上方像素比下方像素更亮的水平边缘), 在其他位置为负. 如果下一层是

⊖ Sobel 算子的命名源于欧文　索韦尔 (Irwin Sobel), 一位数字图像处理领域的美国研究人员.

ReLU 层, 下一层的输出事实上定义了许多“边缘检测特征”, 其只对特定方向的水平或垂直边缘激活. 如果我们将 Sobel 核替换为其他卷积核 (例如通过 SGD 学习得到的那些核), 我们可以学习到只对特定角度的边缘激活的特征.

当我们向深度网络的更深层移动时, 后续的层可以学得只对特定 (但是更复杂) 的模式激活, 例如, 一组构成特定形状的边. 这是因为在第 $l+1$ 层的任一特征都考虑了第 l 层中许多特征的综合影响. 这些更复杂的模式将由更深的层进一步组装起来以激活有语义的物体部件或甚至特定类型的物体, 比如狗、猫、树、海滩等.

卷积层的另一个好处是所有空间位置共享相同的卷积核, 这极大地减小了卷积层需要的参数数量. 例如, 如果输入图像中出现了多只狗, 相同的“疑似狗头模式” (dog-head-like-pattern) 特征将在多个位置激活, 对应于不同的狗的头.

在深度神经网络设定中, 卷积也鼓励参数共享. 例如, 假设“疑似狗头模式”和“疑似猫头模式”是深度卷积网络学得的两个特征. CNN 不需要为它们构建两套不同的参数 (诸如多个层中的卷积核). CNN 底部的层可以学得“疑似眼睛模式”和“动物毛皮纹理模式”, 它们能够被这些更加抽象的特征共享. 简而言之, 对视觉识别任务而言, 卷积核再加上深且层次化的结构这一组合在从图像中学习好的表示 (特征) 时非常有效.

我们想要在这里加一个注释. 尽管我们已经使用了诸如“疑似狗头模式”这样的短语, CNN 学得的表示或者特征可能不会和“狗头”这样的语义概念完全对应. 一个 CNN 特征可能经常对狗的头激活, 并对其他类型的模式不激活. 但是, 也有在其他位置误激活的可能性, 在狗的头部也可能不激活.

事实上, CNN (或更一般地, 深度学习 deep learning) 中的一个重要概念是*分布式表示* (distributed representation). 例如, 假设我们的任务是识别 N 种不同类型的物体, 且一个 CNN 从任意输入图像提取 M 个特征. 最有可能的情况是 M 个特征中的任意一个都对识别所有这 N 个物体类别有用; 而识别一个物体类别需要所有 M 个特征的共同努力.

15.5.3　卷积作为矩阵乘法

公式 (15.16) 看上去相当复杂. 存在一种展开 $\boldsymbol{x}^l$ 并将卷积简化为矩阵乘法的方式.

让我们来考虑一种特殊情况, $D^l = D = 1$、$H = W = 2$、$H^l = 3$ 且 $W^l = 4$. 也就是说, 我们考虑一个小的单通道 3×4 矩阵 (或图像) 和一个 2×2 的滤波器的卷积. 使用图 15.3 的例子, 我们有

$$\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{bmatrix}, \tag{15.19}$$

其中第一个矩阵记为 A, $*$ 是卷积操作符.

现在让我们运行 Matlab / Octave 命令 `B=im2col(A,[2 2])`, 我们得到一个 B 矩阵, 它是 A 的展开版本:

$$B = \begin{bmatrix} 1 & 4 & 2 & 5 & 3 & 6 \\ 4 & 7 & 5 & 8 & 6 & 9 \\ 2 & 5 & 3 & 6 & 1 & 1 \\ 5 & 8 & 6 & 9 & 1 & 1 \end{bmatrix}.$$

显然 B 的第一列对应于 A 的服从列优先规则的第一个 2×2 区域, 也就是对应于 $(i^{l+1}, j^{l+1}) = (0,0)$. 相似地, B 的第二到最后一列分别对应于 A 的相应区域, (i^{l+1}, j^{l+1}) 分别是 $(1,0)$、$(0,1)$、$(1,1)$、$(0,2)$ 和 $(1,2)$. 也就是说, Matlab / Octave 的 `im2col` 函数显式地将执行单个卷积运算所需要的元素展开为矩阵 B 中的一列. B 的转置 B^T 被称为 A 的 `im2row` 展开. 请注意, 参数 `[2 2]` 指定了卷积核的大小.

现在, 如果我们将卷积核自身 (按相同的列优先准则) 向量化为一个向量 $(1,1,1,1)^T$, 我们发现㊀

$$B^T \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 24 \\ 16 \\ 28 \\ 11 \\ 17 \end{bmatrix}. \tag{15.20}$$

如果对公式 (15.20) 的结果向量进行适当的形变, 我们正好得到公式 (15.19) 的卷积结果.

也就是说, 卷积操作是一个线性算子 (operator). 我们可以将展开的输入矩阵和向量化的滤波器相乘来得到结果向量, 并把这个向量进行适当的形变来得到正确的卷积结果.

我们可以将这个思路推广到更复杂的情形并加以形式化. 如果 $D^l > 1$ (即输入 $\boldsymbol{x}^l$ 有不止一个通道), 展开操作可以首先展开 $\boldsymbol{x}^l$ 的第一个通道, 随后第二个, ..., 直到所有 D^l 个通道都被展开. 展开的通道将被堆叠在一起, 也就是说, `im2row` 展开后的一行将有 $H \times W \times D^l$ 个、而不是 $H \times W$ 个元素.

更形式化地讲, 假设 $\boldsymbol{x}^l$ 是 $\mathbb{R}^{H^l \times W^l \times D^l}$ 中的一个 3 阶张量, $\boldsymbol{x}^l$ 中的元素由三元组 (i^l, j^l, d^l) 索引. 我们也考虑一组卷积核 $\boldsymbol{f}$, 其空间大小都是 $H \times W$. 那么, 展开操作 (`im2row`) 将 $\boldsymbol{x}^l$ 转换为一个矩阵 $\phi(\boldsymbol{x}^l)$. 我们使用两个索引 (p,q) 来对这个矩阵的元素进行索引. 展开操作把 $\boldsymbol{x}^l$ 的 (i^l, j^l, d^l) 元素复制到 $\phi(\boldsymbol{x}^l)$ 的 (p,q) 位置.

从展开过程的描述可知, 给定一个固定的 (p,q), 由于显然有

$$p = \quad i^{l+1} + (H^l - H + 1) \times j^{l+1}, \tag{15.21}$$

$$q = \quad i + H \times j + H \times W \times d^l, \tag{15.22}$$

$$i^l = \quad i^{l+1} + i, \tag{15.23}$$

$$j^l = \quad j^{l+1} + j, \tag{15.24}$$

我们可以计算其对应的 (i^l, j^l, d^l) 三元组.

㊀ 本章的记号和介绍深受 MatConvNet 软件包的手册影响 (http://arxiv.org/abs/1412.4564, 它是基于 Matlab 的). im2col 展开的转置等价于im2row展开, 其中一次卷积涉及的数字是im2row展开矩阵的一行. 本节的推导使用im2row, 遵从 MatConvNet 的实现. Caffe, 一个广泛使用的 CNN 软件包 (http://caffe.berkeleyvision.org/, 基于 C++ 的) 使用im2col. 这些公式在数学上彼此等价.

在公式 (15.22) 中, 将 q 除以 HW 并取商的整数部分, 我们可以确定它属于哪个通道 (d^l). 相似地, 我们可以得到卷积核内部的偏移量为 (i,j), 其中 $0 \leqslant i < H$ 且 $0 \leqslant j < W$. 通过三元组 (i,j,d^l), q 完全确定了卷积核内部的某个位置.

请注意, 卷积结果是 $\boldsymbol{x}^{l+1}$, 它的空间大小是 $H^{l+1} = H^l - H + 1$ 和 $W^{l+1} = W^l - W + 1$. 因此, 在公式 (15.21) 中, p 除以 $H^{l+1} = H^l - H + 1$ 得到的余数和商将给出卷积结果中的偏移量 (i^{l+1}, j^{l+1}), 或者说是 $\boldsymbol{x}^l$ 中区域的左上角 (该区域将要和卷积核进行卷积).

根据卷积的定义, 显然我们可以使用公式 (15.23) 和公式 (15.24) 找到输入 $\boldsymbol{x}^l$ 中的偏移量 $i^l = i^{l+1} + i$ 和 $j^l = j^{l+1} + j$. 也就是说, 从 (p,q) 到 (i^l, j^l, d^l) 的映射是一对一的. 然而, 我们想要强调, 反向的从 (i^l, j^l, d^l) 到 (p,q) 的映射是一对多的, 该事实在推导卷积层反向传播公式时很有用.

现在我们使用标准 vec 操作符来将卷积核 $\boldsymbol{f}$ (4 阶张量) 转换为一个矩阵. 让我们从单个卷积核开始, 其可以被向量化为 $\mathbb{R}^{HWD^l}$ 中的一个向量. 因此, 全部的卷积核可以被变形为一个 HWD^l 行、D 列的矩阵 (记住 $D^{l+1} = D$.) 我们称这个矩阵为 F.

最后, 根据所有这些符号, 我们得到一个优美的计算卷积结果的公式 (参见公式 (15.20), 其中 $\phi(\boldsymbol{x}^l)$ 对应的是 B^T):

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}(\boldsymbol{x}^{l+1}) = \operatorname{vec}\left(\phi(\boldsymbol{x}^l)F\right) . \tag{15.25}$$

请注意, $\operatorname{vec}(\boldsymbol{y}) \in \mathbb{R}^{H^{l+1}W^{l+1}D}$、$\phi(\boldsymbol{x}^l) \in \mathbb{R}^{(H^{l+1}W^{l+1}) \times (HWD^l)}$ 且 $F \in \mathbb{R}^{(HWD^l) \times D}$. 矩阵乘法 $\phi(\boldsymbol{x}^l)F$ 产生大小为 $(H^{l+1}W^{l+1}) \times D$ 的矩阵. 这个结果矩阵的向量化得到 $\mathbb{R}^{H^{l+1}W^{l+1}D}$ 中的一个向量, 和 $\operatorname{vec}(\boldsymbol{y})$ 的维数匹配.

15.5.4　克罗内克积

为了计算导数, 简短绕道介绍一下克罗内克积 (Kronecker product, 或译为矩阵直积) 是有必要的.

给定两个矩阵 $A \in \mathbb{R}^{m \times n}$ 和 $B \in \mathbb{R}^{p \times q}$, 克罗内克积 $A \otimes B$ 是一个 $mp \times nq$ 的矩阵, 定义为分块矩阵

$$A \otimes B = \left[\begin{array}{c|c|c} a_{11}B & \cdots & a_{1n}B \\ \hline \vdots & \ddots & \vdots \\ \hline a_{m1}B & \cdots & a_{mn}B \end{array}\right] . \tag{15.26}$$

克罗内克积有如下对我们有用的性质. 对有适当大小的矩阵 A、X 和 B (例如当矩阵乘法 AXB 有定义时), 有

$$(A \otimes B)^T = A^T \otimes B^T , \tag{15.27}$$

$$\operatorname{vec}(AXB) = (B^T \otimes A) \operatorname{vec}(X) , \tag{15.28}$$

请注意, 公式 (15.28) 可以从两个方向加以利用.

借助于 $\otimes$, 我们可以写出

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}\left(\phi(\boldsymbol{x}^l)FI\right) = \left(I \otimes \phi(\boldsymbol{x}^l)\right) \operatorname{vec}(F) , \tag{15.29}$$

$$\operatorname{vec}(\boldsymbol{y}) = \operatorname{vec}\left(I\phi(\boldsymbol{x}^l)F\right) = (F^T \otimes I) \operatorname{vec}(\phi(\boldsymbol{x}^l)) , \tag{15.30}$$

其中 I 是一个有适当大小的单位矩阵. 在公式 (15.29) 中, I 的大小由 F 的列数决定, 因此在公式 (15.29) 中 $I \in \mathbb{R}^{D\times D}$. 相似地, 在公式 (15.30) 中, $I \in \mathbb{R}^{(H^{l+1}W^{l+1})\times(H^{l+1}W^{l+1})}$.

卷积层梯度计算规则的推导涉及许多变量和符号. 我们在表 15.1 中汇总了这个推导中使用到的变量. 请注意有些符号将很快在后续小节中介绍.

表 15.1　卷积层梯度计算规则推导中用到的变量及其大小和含义
请注意“别名”表明某变量有一个不同的名字或者可以被变形为另一种形式

	别名	大小 & 含义
X	$\boldsymbol{x}^l$	$H^lW^l \times D^l$, 输入张量
F	$\boldsymbol{f}$, $\boldsymbol{w}^l$	$HWD^l \times D$, D 个卷积核, 每个大小是 $H \times W$ 且有 D^l 个通道
Y	$\boldsymbol{y}$, $\boldsymbol{x}^{l+1}$	$H^{l+1}W^{l+1} \times D^{l+1}$, 输出, $D^{l+1} = D$
$\phi(\boldsymbol{x}^l)$		$H^{l+1}W^{l+1} \times HWD^l$, $\boldsymbol{x}^l$ 的 `im2row` 展开
M		$H^{l+1}W^{l+1}HWD^l \times H^lW^lD^l$, $\phi(\boldsymbol{x}^l)$ 的指示矩阵
$\frac{\partial z}{\partial Y}$	$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})}$	$H^{l+1}W^{l+1} \times D^{l+1}$, $\boldsymbol{y}$ 的梯度
$\frac{\partial z}{\partial F}$	$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{f})}$	$HWD^l \times D$, 更新卷积核的梯度
$\frac{\partial z}{\partial X}$	$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$	$H^lW^l \times D^l$, $\boldsymbol{x}^l$ 的梯度, 对反向传播有用

15.5.5　反向传播: 更新参数

如前所述, 我们需要计算两个导数 $\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$ 和 $\frac{\partial z}{\partial \operatorname{vec}(F)}$, 其中第一项 $\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$ 将被用于向前一层 (第 $(l-1)$ 层) 的反向传播, 第二项将决定当前层 (第 l 层) 的参数如何更新. 一个友情提醒是: 请记住 $\boldsymbol{f}$、F 和 $\boldsymbol{w}^i$ 指向相同的物体 (即在向量或矩阵或张量的变形后等价). 相似地, 我们可以将 $\boldsymbol{y}$ 形变为矩阵 $Y \in \mathbb{R}^{(H^{l+1}W^{l+1})\times D}$, 那么 $\boldsymbol{y}$、Y 和 $\boldsymbol{x}^{l+1}$ 指向同一个物体 (再一次地, 在变形后等价).

根据链式法则 (公式 15.11), 容易计算 $\frac{\partial z}{\partial \operatorname{vec}(F)}$ 为

$$\frac{\partial z}{\partial(\operatorname{vec}(F))^T} = \frac{\partial z}{\partial(\operatorname{vec}(Y)^T)}\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial(\operatorname{vec}(F)^T)}\,. \tag{15.31}$$

等号右边的第一项在第 $(l+1)$ 层已经被 (等价地) 计算为 $\frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{x}^{l+1}))^T}$. 根据公式 (15.29), 第二项的计算也是非常简单:

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial(\operatorname{vec}(F)^T)} = \frac{\partial\left(\left(I\otimes\phi(\boldsymbol{x}^l)\right)\operatorname{vec}(F)\right)}{\partial(\operatorname{vec}(F)^T)} = I\otimes\phi(\boldsymbol{x}^l)\,. \tag{15.32}$$

请注意, 我们使用了如下事实: 只要矩阵乘法是良定义的, 那么 $\frac{\partial X\boldsymbol{a}^T}{\partial \boldsymbol{a}} = X$ 或 $\frac{\partial X\boldsymbol{a}}{\partial \boldsymbol{a}^T} = X$. 这个公式导出

$$\frac{\partial z}{\partial(\operatorname{vec}(F))^T} = \frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{y})^T)}(I\otimes\phi(\boldsymbol{x}^l))\,. \tag{15.33}$$

进行转置, 我们得到

$$\frac{\partial z}{\partial \operatorname{vec}(F)} = \left(I \otimes \phi(\boldsymbol{x}^l)\right)^T \frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})} \tag{15.34}$$

$$= \left(I \otimes \phi(\boldsymbol{x}^l)^T\right) \operatorname{vec}\left(\frac{\partial z}{\partial Y}\right) \tag{15.35}$$

$$= \operatorname{vec}\left(\phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y} I\right) \tag{15.36}$$

$$= \operatorname{vec}\left(\phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y}\right). \tag{15.37}$$

请注意, 在上面的推导中既使用了公式 (15.28) (从等号右边到等号左边), 也用到了公式 (15.27).

因此, 我们得出结论

$$\frac{\partial z}{\partial F} = \phi(\boldsymbol{x}^l)^T \frac{\partial z}{\partial Y}, \tag{15.38}$$

这是更新第 l 层参数的一个简单规则: 卷积层参数的梯度是 $\phi(\boldsymbol{x}^l)^T$(`im2col` 展开) 和 $\dfrac{\partial z}{\partial Y}$ (从第 $(l+1)$ 层传来的监督信号) 的乘积.

15.5.6　更高维的指示矩阵

函数 $\phi(\cdot)$ 在我们的分析中已经很有用了. 它是很高维度的, 如 $\phi(\boldsymbol{x}^l)$ 有 $H^{l+1}W^{l+1}HWD^l$ 个元素. 从上文我们知道, $\phi(\boldsymbol{x}^l)$ 中的一个元素由 p 和 q 进行索引.

快速回忆一下关于 $\phi(\boldsymbol{x}^l)$ 的信息: 1) 从 q 我们可以确定 d^l, 卷积核的哪个通道被使用了; 也可以确定 i 和 j, 卷积核内部的空间偏移; 2) 从 p 我们可以确定 i^{l+1} 和 j^{l+1}, 卷积结果 $\boldsymbol{x}^{l+1}$ 内部的空间偏移; 以及 3) 输入 $\boldsymbol{x}^l$ 中的空间偏移可以由 $i^l = i^{l+1} + i$ 和 $j^l = j^{l+1} + j$ 确定.

也就是说, 映射 $m : (p, q) \mapsto (i^l, j^l, d^l)$ 是一对一的, 因此这是一个有效的函数. 然而, 其逆映射是一对多的 (因此不是一个有效的函数). 如果我们使用 m^{-1} 表示逆映射, 我们知道 $m^{-1}(i^l, j^l, d^l)$ 是一个集合 S, 其中每个 $(p, q) \in S$ 满足 $m(p, q) = (i^l, j^l, d^l)$.

现在我们换一个视角来观察 $\phi(\boldsymbol{x}^l)$. 为了完全确定 $\phi(\boldsymbol{x}^l)$, 需要什么信息呢? 显然需要 (且只需要) 如下三种信息. 对 $\phi(\boldsymbol{x}^l)$ 的*每*个元素, 我们需要知道

(A) 它属于哪个区域, 即 p 的值是什么 ($0 \leqslant p < H^{l+1}W^{l+1}$)?

(B) 它是区域内部 (或等价地在卷积核内部) 的哪个元素, 即 q 的值是什么 ($0 \leqslant q < HWD^l$)?

上述两种类型的信息确定了 $\phi(\boldsymbol{x}^l)$ 内部的位置 (p, q). 唯一仍缺失的信息是

(C) 这个位置的值, 即 $\left[\phi(\boldsymbol{x}^l)\right]_{pq}$ 等于多少?

由于 $\phi(\boldsymbol{x}^l)$ 的任一元素是 $\boldsymbol{x}^l$ 中某个元素的复制, 我们能够将 [C] 变为一个不同但等价的形式:

(C.1) 对 $\left[\phi(\boldsymbol{x}^l)\right]_{pq}$, 这个值是从哪里复制来的? 或者, $\boldsymbol{x}^l$ 内部的原始位置, 即满足 $0 \leqslant u < H^lW^lD^l$ 的索引 u 是什么?

(C.2) 整个 $\boldsymbol{x}^l$.

容易看到, [A, B, C.1](对 p、q 和 u 的整个范围) 的联合信息, 加上 [C.2] ($\boldsymbol{x}^l$) 包含了与 $\phi(\boldsymbol{x}^l)$ 同样的信息量.

由于 $0 \leqslant p < H^{l+1}W^{l+1}$、$0 \leqslant q < HWD^l$ 且 $0 \leqslant u < H^lW^lD^l$, 我们可以用一个矩阵

$$M \in \mathbb{R}^{(H^{l+1}W^{l+1}HWD^l)\times(H^lW^lD^l)}$$

来编码 [A, B, C.1] 中的信息. 这个矩阵的行索引对应 $\phi(\boldsymbol{x}^l)$ 中的一个位置 (即一个 (p,q) 对). M 的一行有 $H^lW^lD^l$ 个元素, 每个元素由 (i^l, j^l, d^l) 索引. 因此, 这个矩阵中的每个元素由五元组 (p, q, i^l, j^l, d^l) 索引.

那么, 我们可以使用“指示”(indicator) 方法将函数 $m(p,q) = (i^l, j^l, d^l)$ 编码进 M. 也就是说, 对任意可能的 M 中的元素, 其行索引 x 决定了一个 (p,q) 对, 而其列索引 y 决定了一个 (i^l, j^l, d^l) 三元组, M 的定义为

$$M(x,y) = \begin{cases} 1 & \text{如果 } m(p,q) = (i^l, j^l, d^l) \\ 0 & \text{否则} \end{cases}. \tag{15.39}$$

矩阵 M 有如下性质:

- 它的维度非常高.
- 但它也是非常稀疏的; 由于 m 是一个函数, 每行的 $H^lW^lD^l$ 个元素中只有 1 个非零项.
- M 使用信息 [A, B, C.1] 只编码了 $\phi(\boldsymbol{x}^l)$ 中任意元素和 $\boldsymbol{x}^l$ 中任意元素之间的一一对应, 它没有编码 $\boldsymbol{x}^l$ 中任意元素的值.
- 最重要的是, 把 M 中的一一对应信息与 $\boldsymbol{x}^l$ 中的值信息结合起来, 显然我们有

$$\operatorname{vec}(\phi(\boldsymbol{x}^l)) = M \operatorname{vec}(\boldsymbol{x}^l). \tag{15.40}$$

15.5.7　反向传播: 为前一层准备监督信号

在第 l 层中, 我们仍然需要计算 $\dfrac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}$. 为了这个目的, 我们想要将 $\boldsymbol{x}^l$ 变形为矩阵 $X \in \mathbb{R}^{(H^lW^l)\times D^l}$, 并交替使用这两个 (变形前后) 等价的形式.

链式法则指出 (参见公式 15.12)

$$\frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{y})^T)} \frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)}.$$

我们将从研究等号右边的第二项开始 (利用公式 15.30 和公式 15.40):

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial(F^T \otimes I)\operatorname{vec}(\phi(\boldsymbol{x}^l))}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)} = (F^T \otimes I)M. \tag{15.41}$$

因此,

$$\frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{y})^T)}(F^T \otimes I)M. \tag{15.42}$$

由于 (从右到左使用公式 15.28)

$$\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{y})^T)}(F^T\otimes I)=\left((F\otimes I)\frac{\partial z}{\partial\,\mathrm{vec}(\boldsymbol{y})}\right)^T \tag{15.43}$$

$$=\left((F\otimes I)\,\mathrm{vec}\left(\frac{\partial z}{\partial Y}\right)\right)^T \tag{15.44}$$

$$=\mathrm{vec}\left(I\frac{\partial z}{\partial Y}F^T\right)^T \tag{15.45}$$

$$=\mathrm{vec}\left(\frac{\partial z}{\partial Y}F^T\right)^T, \tag{15.46}$$

我们有

$$\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l)^T)}=\mathrm{vec}\left(\frac{\partial z}{\partial Y}F^T\right)^T M\,, \tag{15.47}$$

或等价地

$$\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l))}=M^T\,\mathrm{vec}\left(\frac{\partial z}{\partial Y}F^T\right). \tag{15.48}$$

我们仔细看看等号右边. $\dfrac{\partial z}{\partial Y}F^T\in\mathbb{R}^{(H^{l+1}W^{l+1})\times(HWD^l)}$, 且 $\mathrm{vec}\left(\dfrac{\partial z}{\partial Y}F^T\right)$ 是 $\mathbb{R}^{H^{l+1}W^{l+1}HWD^l}$ 中的一个向量. 而 M^T 是 $\mathbb{R}^{(H^lW^lD^l)\times(H^{l+1}W^{l+1}HWD^l)}$ 中的一个指示矩阵.

为了定位 $\mathrm{vec}(\boldsymbol{x}^l)$ 中的一个元素或 M^T 中的一行, 我们需要一个索引三元组 (i^l,j^l,d^l), 其中 $0\leqslant i^l<H^l$、$0\leqslant j^l<W^l$、$0\leqslant d^l<D^l$. 相似地, 为了定位 M^T 中的一列或者 $\dfrac{\partial z}{\partial Y}F^T$ 中的一个元素, 我们需要一个索引对 (p,q), 其中 $0\leqslant p<H^{l+1}W^{l+1}$ 且 $0\leqslant q<HWD^l$.

因此, $\dfrac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l))}$ 的 (i^l,j^l,d^l) 元素等于如下两个向量的乘积: 由 (i^l,j^l,d^l) 索引的 M^T 的行 (或 M 的列), 以及 $\mathrm{vec}\left(\dfrac{\partial z}{\partial Y}F^T\right)$.

此外, 由于 M^T 是一个指示矩阵, 在由 (i^l,j^l,d^l) 索引的行向量中, 只有对应索引 (p,q) 满足 $m(p,q)=(i^l,j^l,d^l)$ 的项其值为 1, 所有其他项为 0. 因此, $\dfrac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^l))}$ 的 (i^l,j^l,d^l) 元素等于 $\mathrm{vec}\left(\dfrac{\partial z}{\partial Y}F^T\right)$ 中对应的各项之和.

将上述描述转化为精确的数学形式, 我们有如下简洁的公式:

$$\left[\frac{\partial z}{\partial X}\right]_{(i^l,j^l,d^l)}=\sum_{(p,q)\in m^{-1}(i^l,j^l,d^l)}\left[\frac{\partial z}{\partial Y}F^T\right]_{(p,q)}. \tag{15.49}$$

换言之, 为了计算 $\dfrac{\partial z}{\partial X}$, 我们不需要显式地使用极高维的矩阵 M. 相反地, 公式 (15.49) 以及公式 (15.21) 到公式 (15.24) 可以被用于高效地找到 $\dfrac{\partial z}{\partial X}$.

我们使用图 15.3 中简单的卷积例子来图示逆映射 m^{-1}, 如图 15.5 所示.

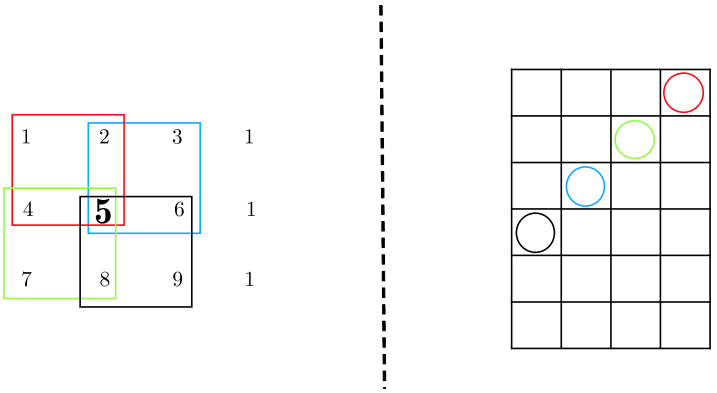


图 15.5　如何计算 $\dfrac{\partial z}{\partial X}$ 的图示

在图 15.5 的右半部分, 那个 6×4 的矩阵是 $\dfrac{\partial z}{\partial Y} F^T$. 为了计算 z 对输入 X 中某个元素的偏导数, 我们需要找出 $\dfrac{\partial z}{\partial Y} F^T$ 中牵涉到的元素并将它们相加. 在图 15.5 的左半部分, 我们展示了输入元素 5(大号字体表示) 牵涉在 4 个卷积运算里, 分别用红色、绿色、蓝色和黑色方框表示. 这 4 个卷积运算分别对应于 $p = 1, 2, 3, 4$. 例如, 当 $p = 2$ 时 (绿色方框), 5 是卷积中的第三个元素, 因此当 $p = 2$ 时有 $q = 3$, 我们在 $\dfrac{\partial z}{\partial Y} F^T$ 矩阵的 $(2, 3)$ 位置画一个绿色的圈. 在我们把 $\dfrac{\partial z}{\partial Y} F^T$ 里所有四个圆圈标记出来后, 偏导数是 $\dfrac{\partial z}{\partial Y} F^T$ 中这四个位置的元素之和.

集合 $m^{-1}(i^l, j^l, d^l)$ 至多包含 HW 个元素. 因此, 公式 (15.49) 至多需要 HW 次求和来计算 $\dfrac{\partial z}{\partial X}$ 中的一个元素㊀ .

15.5.8　用卷积层实现全连接层

正如之前提到的, 卷积层的一个好处是: 卷积是一个局部操作. 卷积核的空间大小通常很小 (例如 3×3). $\boldsymbol{x}^{l+1}$ 中的一个元素通常由其输入 $\boldsymbol{x}^l$ 的一小部分元素计算得到.

全连接层 (fully connected layer)指输出 $\boldsymbol{x}^{l+1}$ (或 $\boldsymbol{y}$) 中的任意元素需要用输入 $\boldsymbol{x}^l$ 中所有元素来计算的层. 全连接层有时在深度 CNN 模型的尾部有用. 例如, 如果在许多卷积、ReLU 和汇合层 (很快将讨论到) 之后, 当前层的输出包含输入图像的分布式表示, 我们想要使用当前层的所有这些特征来构建有更强容量 (capability) 的下一层的特征. 全连接层对此目的而言是很有用的.

假设一层的输入 $\boldsymbol{x}^l$ 有大小 $H^l \times W^l \times D^l$. 如果我们使用大小为 $H^l \times W^l \times D^l$ 的卷积核, 那么 D 个这样的卷积核构成了一个大小为 $H^l \times W^l \times D^l \times D$ 的 4 阶张量. 输出是 $\boldsymbol{y} \in \mathbb{R}^D$. 显然, 为了计算 $\boldsymbol{y}$ 中的任意元素, 我们需要使用输入 $\boldsymbol{x}^l$ 中的所有元素. 因此, 这样的一个层是一个全连接层, 但是可以用卷积层进行实现. 因此, 我们不需要单独地推导全连接层的学习规则.

㊀ 在 Caffe 中, 该计算是由一个名为 `col2im` 的函数实现的. 在 MatConvNet 中, 该操作以 `row2im` 的方式执行, 尽管它没有显式地使用 `row2im` 这个名字.

15.6　汇合层

我们将沿用与卷积层中相同的符号来介绍汇合层 (pooling layer).

令 $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$ 是第 l 层的输入, 这一层现在是一个汇合层. 汇合操作不需要参数 (即 $\boldsymbol{w}^i$ 是空, 因此本层不需要参数学习). 汇合的空间范围 ($H \times W$) 在 CNN 的结构设计中指定. 假设 H 能整除 H^l、W 能整除 W^l, 且步长等于汇合的空间大小㊀, 汇合层的输出 ($\boldsymbol{y}$ 或等价的 $\boldsymbol{x}^{l+1}$) 将是一个大小为 $H^{l+1} \times W^{l+1} \times D^{l+1}$ 的 3 阶张量, 其中

$$H^{l+1} = \frac{H^l}{H}, \quad W^{l+1} = \frac{W^l}{W}, \quad D^{l+1} = D^l. \tag{15.50}$$

汇合层对 $\boldsymbol{x}^l$ 一个通道接着一个通道独立地进行操作. 在每个通道内, 有 $H^l \times W^l$ 个元素的矩阵被分为 $H^{l+1} \times W^{l+1}$ 个不重叠的子区域, 每个子区域的大小是 $H \times W$. 然后汇合操作将一个子区域映射为单个数字.

两种类型的汇合十分常用: 最大汇合和平均汇合. 在最大汇合 (max pooling)中, 汇合操作将一个子区域映射为其最大值, 而平均汇合 (average pooling)将一个子区域映射为其平均值. 用精确的数学语言,

$$\text{最大}: \qquad y_{i^{l+1},j^{l+1},d} = \max_{0 \leqslant i < H, 0 \leqslant j < W} x^l_{i^{l+1} \times H + i, j^{l+1} \times W + j, d}, \tag{15.51}$$

$$\text{平均}: \qquad y_{i^{l+1},j^{l+1},d} = \frac{1}{HW} \sum_{0 \leqslant i < H, 0 \leqslant j < W} x^l_{i^{l+1} \times H + i, j^{l+1} \times W + j, d}, \tag{15.52}$$

其中 $0 \leqslant i^{l+1} < H^{l+1}$、$0 \leqslant j^{l+1} < W^{l+1}$ 且 $0 \leqslant d < D^{l+1} = D^l$.

汇合是一个局部操作, 它的前向计算非常简单. 现在我们关注它的反向传播. 在这里我们只讨论最大汇合, 并且我们可以再次借助于指示矩阵. 平均汇合可以用相似的思路处理.

我们需要在这个指示矩阵中编码的所有信息是: $\boldsymbol{y}$ 中的每个元素来源于 $\boldsymbol{x}^l$ 的哪里?

我们需要一个三元组 (i^l, j^l, d^l) 来定位 $\boldsymbol{x}^l$ 中的一个元素, 以及另一个三元组 $(i^{l+1}, j^{l+1}, d^{l+1})$ 来定位 $\boldsymbol{y}$ 中的一个元素. 当且仅当如下的条件成立时, 汇合的输出 $y_{i^{l+1},j^{l+1},d^{l+1}}$ 来自于 $x^l_{i^l,j^l,d^l}$:

- 它们来自同一个通道;
- 空间位置 (i^l, j^l) 属于第 (i^{l+1}, j^{l+1}) 个子区域;
- 空间位置 (i^l, j^l) 的元素是这个子区域中最大的一个.

将这些条件翻译为公式, 我们有

$$d^{l+1} = d^l, \tag{15.53}$$

$$\left\lfloor \frac{i^l}{H} \right\rfloor = i^{l+1}, \left\lfloor \frac{j^l}{W} \right\rfloor = j^{l+1}, \tag{15.54}$$

$$x^l_{i^l,j^l,d^l} \geqslant y_{i+i^{l+1} \times H, j+j^{l+1} \times W, d^l}, \forall\, 0 \leqslant i < H, 0 \leqslant j < W, \tag{15.55}$$

其中 $\lfloor \cdot \rfloor$ 是向下取整函数 (也称为地板函数, floor function). 如果垂直 (水平) 方向的步长不是 H (W), 公式 (15.54) 需要进行相应地调整.

㊀ 也就是说, 垂直和水平方向的步长分别是 H 和 W, 最常用的汇合设定是 $H = W = 2$, 步长为 2.

给定一个 $(i^{l+1}, j^{l+1}, d^{l+1})$ 三元组, 只有一个三元组 (i^l, j^l, d^l) 满足所有这些条件. 因此, 我们定义一个指示矩阵

$$S(\boldsymbol{x}^l) \in \mathbb{R}^{(H^{l+1}W^{l+1}D^{l+1})\times(H^lW^lD^l)}\,. \tag{15.56}$$

一个三元组索引 $(i^{l+1}, j^{l+1}, d^{l+1})$ 确定了 S 中的一行, 而 (i^l, j^l, d^l) 确定了一列. 这两个三元组一起定位了 $S(\boldsymbol{x}^l)$ 的一个元素. 如果同时满足公式 (15.53) 到公式 (15.55), 我们置该元素为 1, 否则为 0. $S(\boldsymbol{x}^l)$ 中的一行对应于 $\boldsymbol{y}$ 中的一个元素, 一列对应于 $\boldsymbol{x}^l$ 中的一个元素.

借助于指示矩阵, 我们有

$$\operatorname{vec}(\boldsymbol{y}) = S(\boldsymbol{x}^l)\operatorname{vec}(\boldsymbol{x}^l)\,. \tag{15.57}$$

那么, 显然

$$\frac{\partial \operatorname{vec}(\boldsymbol{y})}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)} = S(\boldsymbol{x}^l), \quad \frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{x}^l)^T)} = \frac{\partial z}{\partial(\operatorname{vec}(\boldsymbol{y})^T)}S(\boldsymbol{x}^l)\,, \tag{15.58}$$

随之有

$$\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)} = S(\boldsymbol{x}^l)^T\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})}\,. \tag{15.59}$$

$S(\boldsymbol{x}^l)$ 非常稀疏. 它每行只有一个非零元素. 因此, 在计算时我们不需要使用整个矩阵. 相反地, 我们只需要记录这些非零元素的位置——在 $S(\boldsymbol{x}^l)$ 中只有 $H^{l+1}W^{l+1}D^{l+1}$ 个这样的元素.

一个简单的例子可以解释这些公式的含义. 让我们来考虑 2×2、步长为 2 的最大汇合. 对一个给定的通道 d^l, 第一个子区域包含了输入的 4 个元素, 即 $(i,j) = (0,0)$、$(1,0)$、$(0,1)$ 和 $(1,1)$, 并且让我们假设在空间位置 $(0,1)$ 的元素是它们中最大的. 在前向过程中, 输入中由 $(0,1,d^l)$ 索引的元素的值 (即 $x^l_{0,1,d^l}$) 将会被赋值给输出中由 $(0,0,d^l)$ 索引的元素 (即 $y_{0,0,d^l}$).

如果步长分别是 H 和 W, $S(\boldsymbol{x}^l)$ 的一列至多包含一个非零元素. 在上述的例子中, 由 $(0,0,d^l)$、$(1,0,d^l)$ 和 $(1,1,d^l)$ 索引的 $S(\boldsymbol{x}^l)$ 的列都是零向量. 对应于 $(0,1,d^l)$ 的列只包含一个非零元素, 其行索引由 $(0,0,d^l)$ 确定. 因此, 在反向传播中, 我们有

$$\left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}\right]_{(0,1,d^l)} = \left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})}\right]_{(0,0,d^l)}\,,$$

和

$$\left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}\right]_{(0,0,d^l)} = \left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}\right]_{(1,0,d^l)} = \left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}\right]_{(1,1,d^l)} = 0\,.$$

然而, 如果汇合步长在垂直和水平方向分别比 H 和 W 小, 输入张量的一个元素可以是多个汇合子区域中的最大元素. 因此, $S(\boldsymbol{x}^l)$ 中的一列可以有不止一个非零元素. 让我们考虑图 15.5 的输入样例. 如果使用 2×2、各方向步长都是 1 的最大汇合, 元素 9 是两个汇合区域 $\left(\begin{bmatrix}5&6\\8&9\end{bmatrix}$ 和 $\begin{bmatrix}6&1\\9&1\end{bmatrix}\right)$ 中的最大值. 因此, 在 $S(\boldsymbol{x}^l)$ 中对应于元素 9 (在输入张量中由 $(2,2,d^l)$ 索引) 的列中有两个非零元素, 其行索引分别是 $(i^{l+1}, j^{l+1}, d^{l+1}) = (1,1,d^l)$ 和

$(1,2,d^l)$. 因此, 在这个例子中, 我们有

$$\left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{x}^l)}\right]_{(2,2,d^l)} = \left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})}\right]_{(1,1,d^l)} + \left[\frac{\partial z}{\partial \operatorname{vec}(\boldsymbol{y})}\right]_{(1,2,d^l)}.$$

15.7　案例分析: VGG-16 网络

至今我们已经介绍了卷积、汇合、ReLU 和全连接层, 并简要提及了 softmax 层. 利用这些层, 我们可以构建许多强大的深度 CNN 模型.

15.7.1　VGG-Verydeep-16

VGG-Verydeep-16 CNN 模型是由牛津 VGG 组发布的预训练 CNN 模型[⊖]. 我们将使用它作为学习 CNN 结构细节的一个例子. VGG-16 模型结构见表 15.2.

表 15.2　VGG-Verydeep-16的结构与感受野

	类型	描述	感受野		类型	描述	感受野
1	Conv	64;3x3;p=1,st=1	212	20	Conv	512;3x3;p=1,st=1	20
2	ReLU		210	21	ReLU		18
3	Conv	64;3x3;p=1,st=1	210	22	Conv	512;3x3;p=1,st=1	18
4	ReLU		208	23	ReLU		16
5	Pool	2x2;st=2	208	24	Pool	2x2;st=2	16
6	Conv	128;3x3;p=1,st=1	104	25	Conv	512;3x3;p=1,st=1	8
7	ReLU		102	26	ReLU		6
8	Conv	128;3x3;p=1,st=1	102	27	Conv	512;3x3;p=1,st=1	6
9	ReLU		100	28	ReLU		4
10	Pool	2x2;st=2	100	29	Conv	512;3x3;p=1,st=1	4
11	Conv	256;3x3;p=1,st=1	50	30	ReLU		2
12	ReLU		48	31	Pool		2
13	Conv	256;3x3;p=1,st=1	48	32	FC	(7x7x512)x4096	1
14	ReLU		46	33	ReLU		
15	Conv	256;3x3;p=1,st=1	46	34	Drop	0.5	
16	ReLU		44	35	FC	4096x4096	
17	Pool	2x2;st=2	44	36	ReLU		
18	Conv	512;3x3;p=1,st=1	22	37	Drop	0.5	
19	ReLU		20	38	FC	4096x1000	
				39	σ	(softmax 层)	

这个模型中有六种类型的层.

卷积层　卷积层简写为“Conv”. 它的描述包括了三部分: 通道数目, 卷积核空间大小 (卷积核大小), 填补 ('p') 和步长 ('st') 大小.

ReLU 层　ReLU 层不需要描述.

汇合层　汇合层简写为“Pool”. VGG-16 只用了最大汇合. VGG-16 中汇合核的大小总是 2×2, 步长总是 2.

⊖ http://www.robots.ox.ac.uk/∼vgg/research/very_deep/

全连接层　全连接层简记为“FC”. 在 VGG-16 中, 全连接层用卷积层进行实现. 它的大小描述格式是 $n_1 \times n_2$, 其中 n_1 是输入张量的大小, n_2 是输出张量的大小. 尽管 n_1 可能是一个三元组 (例如 $7 \times 7 \times 512$), n_2 总是一个整数.

随机失活层　随机失活层 (dropout layer) 简记为“Drop”. 随机失活是改善深度学习方法泛化性能的一种技术. 它将网络中连接到一定比例的结点的权重置为零 (VGG-16 在两个随机失活层中置这个比例为 0.5).

Softmax 层　简记为“σ”.

关于这个深度 CNN 结构的样例, 我们想要补充一些注释.

- 在 VGG-16 中, 卷积层后总是跟着 ReLU 层. ReLU 层用于增加 CNN 模型的非线性.
- 两个汇合层之间的各卷积层有相同的通道数、卷积核大小和步长. 事实上, 将两个 3×3 的卷积层叠加在一起等价于一个 5×5 的卷积层; 将三个 3×3 的卷积层叠加在一起将取代一个 7×7 的卷积层. 然而, 叠加若干个 (2 个或 3 个) 小的卷积核将比一个大的卷积核计算起来更快. 此外, 参数量也可以降低, 例如 $2 \times 3 \times 3 = 18 < 25 = 5 \times 5$. 小卷积层之间的 ReLU 层也是有帮助的.
- VGG-16 的输入是 $224 \times 224 \times 3$ 大小的图像. 由于卷积核中的填补是 1(意味着输入的四个边缘外添加了一行或者一列), 卷积将不改变空间大小. 汇合层其将输入的空间大小减半. 因此, 最后一个 (第 5 个) 汇合层后的输出的空间大小是 7×7(并且有 512 个通道). 我们可以将这个张量理解为 $7 \times 7 \times 512 = 25088$ 个“特征”. 第一个全连接层将它们转化为 4096 个特征. 在第二个全连接层后, 特征数保持 4096 不变.
- VGG-16 是为 ImageNet 分类竞赛而训练的, 这是一个有 1000 个类别的物体识别问题. 对每幅输入图像, 最后一个全连接层 (4096×1000) 输出长度为 1000 的向量, softmax 层将这个长度为 1000 的向量转换为对 1000 个类的后验概率的估计.

15.7.2　感受野

CNN 中另外一个重要的概念是感受野(receptive field), 参见表 15.2. 让我们来看看第一个全连接层的输入中的某个元素 (`32|FC`). 由于它是最大汇合的输出, 我们需要 2×2 空间范围内的汇合层输入来计算这个元素 (并且我们只需要这个空间范围内的元素). 这个 2×2 的空间范围被称为该元素的感受野. 在表 15.2 中, 我们列出了最后一个汇合层中的任意元素的空间范围. 请注意, 由于感受野是方形的, 我们只需要一个数字 (例如 48 代表 48×48). 为某一层列出的感受野大小是在这一层的输入中的空间范围.

一个 3×3 的卷积层将感受野增加 2, 而一个汇合层将空间范围翻倍. 如表 15.2 所示, 在第一层的输入中感受野大小是 212×212. 换言之, 为了计算最后一个汇合层的 $7 \times 7 \times 512$ 个输出中的任意一个元素, 需要一个 212×212 的图像块 (包括所有卷积层填补的像素).

当网络变得更深时, 显然感受野将增加, 尤其是当一个汇合层被加入到深度网络中时. 和传统计算机视觉和图像处理特征只依赖于一个小的感受野 (例如 16×16) 不同, 深度 CNN 使用大的感受野计算其表示 (或特征). 大感受野这一特性是 CNN 为何能在图像识别中取得比经典方法更高性能的重要原因之一.

15.8　CNN 的亲身体验

我们希望这个 CNN 介绍章节对我们的读者来说是清晰、自足并容易理解的.

在读者对他或她在 CNN 数学层面的理解有信心之后, 下一步得到一些 CNN 的亲身体验是很有帮助的. 例如, 如果您偏向于 Matlab 环境, 可以使用 MatConvNet 软件包验证本章讨论的内容㈠. 对 C++ 爱好者, Caffe 是一个广泛使用的工具㈡. Theano 包是一个用于深度学习的 Python 包㈢. 还有更多关于深度学习 (不仅是 CNN) 的资源, 例如 Torch㈣, PyTorch㈤, MXNet㈥, Keras㈦, TensorFlow㈧, 以及更多. 本章的习题可以作为合适的 CNN 首次编程练习题.

15.9　阅读材料

在本章的习题中, 我们提供许多 CNN 重要技术的资源链接, 包括随机失活、VGG-Verydeep 和 GoogLeNet 网络结构, 批量规范化 (batch normalization) 和残差连接 (residual connection). 关于更多 CNN 和其他深度学习模型的信息, 请参阅 [97].

更多关于 Sobel 算子的材料请参阅 [96].

[11] 包含了关于分布式表示性质很好的总结.

卷积神经网络已经在许多计算机视觉任务中取得了最高的性能, 例如物体识别 [109]、目标检测 [92]、语义分割 [215]、单图像深度估计 [149] 等.

习题

15.1 随机失活 (dropout) 是一个训练神经网络时很有用的技术, 它由 Srivastava 等人在 JMLR 发表的论文“Dropout: A Simple Way to Prevent Neural Networks from Overfitting”中提出 [222]㈨. 仔细阅读该论文并回答如下问题 (请将每问的答案组织为一个简洁的语句).

(a) 随机失活在训练时如何操作?

(b) 随机失活在测试时如何操作?

(c) 随机失活有什么好处?

(d) 为什么随机失活可以得到这个好处?

15.2 VGG16 CNN 模型 (也称为 VGG-Verydeep-16) 由 Karen Simonyan 和 Andrew Zisserman 在 arXiv 预印服务器上公布的论文“Very Deep Convolutional Networks for Large-Scale

㈠ http://www.vlfeat.org/matconvnet/

㈡ http://caffe.berkeleyvision.org/

㈢ http://deeplearning.net/software/theano/

㈣ http://torch.ch/

㈤ http://pytorch.org/

㈥ https://mxnet.incubator.apache.org/

㈦ https://keras.io/

㈧ https://www.tensorflow.org/

㈨ 可在http://jmlr.org/papers/v15/srivastava14a.html获得

Image Recognition”中提出 [219]㈠. 此外, GoogLeNet 模型由 Szegedy 等人在 arXiv 预印服务器上公布的论文“Going Deeper with Convolutions”中提出 [226]㈡. 这两篇论文几乎同时公开, 并共享一些相似的思路. 仔细阅读两篇论文并回答如下问题 (请将每问的答案组织为一个简洁的语句).

(a) 为什么它们使用小的卷积核 (主要是 3×3) 而不是大的卷积核?

(b) 为什么这两个网络都相当深 (即有许多层, 大概 20 左右)?

(c) 什么困难是由大的深度造成的? 在这两个网络中是如何解决的?

15.3 批量规范化 (Batch Normalization, BN) 是另一个在训练深度神经网络时很有用的技术, 它由 Sergey Ioffe 和 Christian Szegedy 在 ICML 2015 发表的论文“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”中提出 [121]㈢. 仔细阅读该论文并回答如下问题 (请将每问的答案组织为一个简洁的语句).

(a) 什么是内部协变量转移 (internal covariate shift)?

(b) BN 是如何处理这个问题的?

(c) BN 在卷积层内如何操作?

(d) BN 有什么好处?

15.4 ResNet 是一个非常深神经网络的学习技术, 由 He 等人在 CVPR 2016 发表的论文“Deep Residual Learning for Image Recognition”中提出 [109]㈣. 仔细阅读该论文并回答如下问题 (请将每问的答案组织为一个简洁的语句).

(a) 尽管 VGG16 和 GoogLeNet 在训练大约 20∼30 层深的网络时遇到了困难, 是什么使得 ResNet 可以训练深如 1000 层的网络?

(b) VGG16 是一个前馈网络, 其中每一层只有一个输入和一个输出. 而 GoogLeNet 和 ResNet 是有向无环图结构 (Directed Acyclic Graph, DAG), 只要网络结构中的数据流没有构成一个环, 一层能够有多个输入和多个输出. DAG 和前馈结构相比有什么好处?

(c) VGG16 有两个全连接层 (fc6 和 fc7), 而 ResNet 和 GoogLeNet 没有全连接层 (除了用于分类的最后一层). 什么被用来取代 FC? 这有什么好处?

15.5 AlexNet 指在 ILSVRC 竞赛数据上训练的深度卷积神经网络, 这是在计算机视觉任务中深度 CNN 的一个开创性工作. AlexNet 的技术细节在论文“ImageNet Classification with Deep Convolutional Neural Networks”中给出, 这是由 Alex Krizhevsky、Ilya Sutskever 和 Geoffrey E. Hinton 在 NIPS 25 发表的论文 [136]㈤. 它提出 ReLU 激活函数, 并创造性地使用 GPU 来加速计算. 仔细阅读该论文并回答如下问题 (请将每问的答案组织为一个简洁的语句).

㈠ 可在https://arxiv.org/abs/1409.1556获得, 后续作为会议论文发表在 ICLR 2015.

㈡ 可在https://arxiv.org/abs/1409.4842获得, 后续发表在 CVPR 2015.

㈢ 可在http://jmlr.org/proceedings/papers/v37/ioffe15.pdf获得.

㈣ 可在https://arxiv.org/pdf/1512.03385.pdf获得.

㈤ 可从下面的网址http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks获得.

(a) 表述你怎么理解 ReLU 帮助 AlexNet 取得成功? GPU 又是如何助力 AlexNet 的?
(b) 使用多个网络的预测的平均会降低错误率. 为什么?
(c) 随机失活技术是在哪里被应用的? 它是如何提供帮助的? 使用随机失活有什么代价?
(d) AlexNet 中有多少参数? 为什么数据集大小 (120 万) 对 AlexNet 的成功很重要?

15.6 我们将在 MNIST 数据集上尝试不同的 CNN 结构. 在本题中, 我们记 MatConvNet 的 MNIST 例子中的“基准”(baseline) 网络为 BASE ㊀. 在本题中, 卷积层记为 “$x \times y \times$ nIn $\times$ nOut”, 其中卷积核大小是 $x \times y$, 有 nIn 个输入和 nOut 个输出通道, 步长为 1 且填补为 0. 汇合层是 2×2 的最大汇合, 步长为 2. BASE 网络有 4 个模块. 第 1 个模块包括了一个 $5 \times 5 \times 1 \times 20$ 的卷积和一个最大汇合; 第 2 个模块包括了一个 $5 \times 5 \times 20 \times 50$ 的卷积和一个最大汇合; 第 3 个模块是一个 $4 \times 4 \times 50 \times 500$ 的卷积 (FC) 和一个 ReLU 层; 最后一个模块是分类层 ($1 \times 1 \times 500 \times 10$ 的卷积).

(a) MNIST 数据集可以在http://yann.lecun.com/exdb/mnist/获得. 阅读该网页的说明, 编写程序将数据转化为适合你最喜欢的深度学习软件的格式.
(b) 学习深度学习模型通常涉及随机数. 在开始训练之前, 将随机数生成器的种子设为 0. 随后, 使用 BASE 网络结构和前 10000 个训练样例学习其参数. 在 20 轮训练之后,(在 10000 个测试样例上的) 测试集错误率是多少?
(c) 从现在开始, 若未有其他说明, 我们均假设使用前 10000 个训练样例并训练 20 轮. 现在我们定义 BN 网络结构, 其在前三个模块的每个卷积层后面加一个批量规范化层. 其错误率是多少? 你想对 BN 和 BASE 的比较说些什么?
(d) 如果你在第 4 个模块的分类层后面加一个随机失活层. BASE 和 BN 的新的错误率是多少? 你对随机失活想评论些什么?
(e) 现在我们定义 SK 网络结构, 这指的是小的卷积核大小. SK 基于 BN. 第 1 个模块 (5×5 的卷积加上汇合) 现在变为两个 3×3 的卷积, 并在每个卷积后面使用 BN + ReLU. 例如, 第 1 个模块现在是 $3 \times 3 \times 1 \times 20$ 的卷积 + BN + ReLU + $3 \times 3 \times 20 \times 20$ 的卷积 + BN + ReLU + pool. SK 的错误率是多少? 你对此有何评论 (例如, 错误率是如何变化的? 为什么会这样变化?)
(f) 现在我们定义 SK-s 网络结构. 符号‘s’表示改变卷积层通道数的一个乘子. 例如, SK 和 SK-1 是相同的. SK-2 代表所有卷积层的通道数 (除了第 4 个模块) 都乘以 2. 训练 SK-2、SK-1.5、SK-1、SK-0.5 和 SK-0.2 对应的网络. 汇报它们的错误率并进行评论.
(g) 现在我们用 SK-0.2 网络结构对不同大小的训练数据集进行实验. 使用前 500、1000、2000、5000、10000、20000 和 (所有的)60000 个训练样例, 你得到什么样的错误率? 对你的观察进行评论.
(h) 使用 SK-0.2 网络结构, 研究不同训练集对结果的影响. 训练 6 个不同的网络, 在训练第 i 个网络时使用第 $(10000 \times (i-1)+1)$ 个到第 $(i \times 10000)$ 个训练样例. CNN

㊀ MatConvNet 版本是 1.0-beta20. 请查阅 MatConvNet 得到 BASE 的所有细节, 例如参数初始化和学习率.

对不同训练集稳定吗?

(i) 现在我们研究随机性对 CNN 学习的影响. 除了把随机数生成器的种子设为 0 之外, 使用 1、12、123、1234、12345 和 123456 作为种子来训练 6 个不同的 SK-0.2 网络. 它们的错误率是多少? 对你的观察进行评论.

(j) 最后, 在 SK-0.2 中将所有的 ReLU 层替换为 sigmoid 层. 对使用 ReLU 和 sigmoid 激活函数得到的错误率进行比较并评论.