深度卷积神经网络的实现、常用技术以及其在图像识别上的应用

伍炜臻 20142210022
数学科学学院
信息与计算科学系

2016 年 12 月 25 日

目录

摘要

在图像处理领域当中，普通的神经网络会忽视图像中局部相关性较高的关键特性，而且同一种特征在不同的图像中的位置、方向、形状往往是不相同的。卷积神经网络的特殊结构就是为了从图像中提取局部特征，使得网络具有更好的泛化能力。本文首先介绍了卷积神经网络的结构细节，然后简单介绍了卷积神经网络实践中一些常用的技术，最后展示了卷积神经网络在 MNIST 手写数字数据集和 CIFAR-10 图像数据集上的识别实例。

1 卷积神经网络结构

卷积神经网络 [6] 是一种带有特殊结构的神经网络，它被广泛应用于图像处理领域。普通神经网络会忽视图像的一个关键性质，图像中距离较近的像素之间的相关性远远大于距离较远的像素之间的相关性。图像中目标的位置、大小、方向的变化和一定程度的变形不影响目标的所属的类别。即使输入的发生一定的变换 (平移、缩放、旋转和变形) 也应该有相同的预测输出，这被称为不变性。卷积神经网络的卷积结构就是为

了达到这种不变性，其通过卷积来提取局特征，通过降采样的来在保留特征信息的前提下降低特征矩阵的维数。

本文介绍的卷积神经网络中，不在卷积层、采样层、全连接层中设置激活函数，而是显式地在网络中特定位置设置激活函数层，这是为了实现更灵活的结构。

1.1 矩阵的卷积

卷积神经网络的核心思想就是利用称为卷积核的矩阵，通过卷积运算从矩阵中提取特征。可以把卷积核视作提取某种特定特征的滤波器，那么用多个不同的滤波器就可以从矩阵中提取一些我们感兴趣的特征。矩阵的卷积运算是将矩阵的每个元素都按某个权值添加到其邻域中。权值和邻域大小由一个称为卷积核的矩阵确定。

1.1.1 维数相同的卷积

两个维数相同的矩阵的卷积计算定义为：将其中一个矩阵旋转 180 度 (上下方向和左右方向都翻转)，然后将经过旋转的矩阵和另一个矩阵相同位置的元素都相乘，将所有乘积加起来得到的一个数就是卷积的结果。例如

$$\begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} * \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = iA + hB + gC + fD + eE + dF + cG + bH + aI \tag{1}$$

1.1.2 维数不同的卷积

在通常应用当中进行卷积计算中两个矩阵的维数不相同，存在不同的卷积计算方法，它们的区别是处理边缘的方式不同。

裁切边缘的卷积 裁切边缘的卷积运算 $*$ 定义为：设矩阵 X 的维数是 $r_1 \times c_1$，矩阵 Ker 的维数 $r_2 \times c_2$，且 $r_1 \geq r_2, c_2 \geq c_1$。从 X 中可以取出 $(r_1 - r_2 + 1) \times (c_1 - c_2 + 1)$ 个与 Ker 有相同维数的子矩阵 (行列顺序不变，并且没有跳行或跳列)，将每个子矩阵都与 Ker 进行卷积，将所有结果都按照原子矩阵的相对排列，得到 $(r_1 - r_2 + 1) \times (c_1 - c_2 + 1)$ 的矩阵中。例如

$$\begin{pmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{pmatrix} * \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = \begin{pmatrix} X_{11} * Ker & X_{12} * Ker \\ X_{21} * Ker & X_{22} * ker \end{pmatrix} \tag{2}$$

其中

$$X_{11} = \begin{pmatrix} A & B & C \\ E & F & G \\ I & J & K \end{pmatrix}, X_{12} = \begin{pmatrix} B & C & D \\ F & G & H \\ J & K & L \end{pmatrix}, X_{21} = \begin{pmatrix} E & F & G \\ I & J & K \\ M & N & O \end{pmatrix}, \tag{3}$$

$$X_{22} = \begin{pmatrix} F & G & H \\ J & K & L \\ N & O & P \end{pmatrix}, Ker = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \tag{4}$$

可以把卷积的过程看作较小的卷积核在较大的矩阵内部以 1 的步长滑动。通过卷积核在每个位置都通过卷积计算得到一个值，那么在 $(r_1 - r_2 + 1) \times (c_1 - c_2 + 1)$ 个位置的值就构成卷积的结果。

扩展边缘的卷积 扩展边缘的卷积运算 $\star$ 定义为：设矩阵 X 的维数是 $r_1 \times c_1$，矩阵 Ker 的维数 $r_2 \times c_2$，且 $r_1 \geq r_2, c_2 \geq c_1$。将 X 的上下边缘都分别添加 $r_2 - 1$ 行 0 元素，再将其左右边缘都分别添加 $c_2 - 1$ 列 0 元素，得到扩展的矩阵 X_0。那么卷积计算为

$$X \star Ker = X_0 * Ker \tag{5}$$

得到的是一个 $(r_1 + r_2 - 1) \times (c_1 + c_2 - 1)$ 维的矩阵。

1.2 综述

卷积神经网络是一种函数，其输入为一个或多个相同维数的矩阵，输出为一个向量。卷积神经网络具有复杂的结构，其含有大量参数。参数主要分为两类，一类是包括权值系数和偏置 (截距) 系数的普通参数，另一类是控制网络层数、各层单元大小数量、各卷积层或采样层的处理细节等网络结构的超参数。在卷积神经网络的结构确定的情况下，可以把网络的输出 $\boldsymbol{y}$ 看做是关于输入矩阵集合 $\boldsymbol{X}$ 和系数集合 $\boldsymbol{\theta}$ 的函数

$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{X}, \boldsymbol{\theta}) \tag{6}$$

利用训练数据集合 $\{(\boldsymbol{X_1}, \boldsymbol{t_1}), \ldots, (\boldsymbol{X_n}, \boldsymbol{t_n})\}$ 来训练网络参数 $\boldsymbol{\theta}$，使得

$$\boldsymbol{y}(\boldsymbol{X_i}, \boldsymbol{\theta}) \approx \boldsymbol{t_i} \tag{7}$$

定义一个损失函数来衡量网络的欠拟合的程度。对于训练数据的一个样例 $(\boldsymbol{X_i}, \boldsymbol{t_i})$，损失函数形式为

$$E(\boldsymbol{\theta}; \boldsymbol{X_i}, \boldsymbol{t_i}) \tag{8}$$

对于整个训练数据集合，损失函数为

$$E(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{t}) = \frac{1}{n} \sum_{i=1}^{n} E(\boldsymbol{\theta}; \boldsymbol{X_i}, \boldsymbol{t_i}) \tag{9}$$

训练参数使网络拟合训练数据集合的过程就可以变成优化损失函数的问题

$$\boldsymbol{\theta} = arg\min E(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{t}) \tag{10}$$

可以使用梯度下降的方法来迭代优化，η 为学习速率。

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \frac{\partial E}{\partial \boldsymbol{\theta}} \tag{11}$$

1.2.1 前向传播

从输入 $\boldsymbol{X}$ 到输出 $\boldsymbol{y}$ 的计算过程称为前向传播。如果网络有 L 层，那么前向传播的过程为，

$$\boldsymbol{X}^{(l)} := \begin{cases} \boldsymbol{X} & , l = 1, \\ g^{(l)}(\boldsymbol{X}^{(l-1)}) & , l = 2, \ldots, L \end{cases} \tag{12}$$

$$\boldsymbol{y} := \boldsymbol{X}^{(L)} \tag{13}$$

其中 L 是网络的深度、层数。$\boldsymbol{X}^{(l)}$ 是各层的输出值，可以是向量，也可以是一个或多个相同维数的矩阵。$\boldsymbol{X}^{(l)} := g^{(l)}(\boldsymbol{X}^{(l-1)})$ 表示从 $l-1$ 层的输出值传播到 l 层。

1.2.2 反向传播

求损失函数 E 对各层系数梯度需要借助一个被称为残差 δ 的量。在完成前向传播和计算损失函数之后，网络中除了输入层的每层每个输出值 $x_i^{(l)}$ 都有一个相应地残差，定义为

$$\delta_i^{(l)} = \frac{\partial E}{\partial x_i^{(l)}} \tag{14}$$

对于一层的输出矩阵 (向量)$\boldsymbol{X}^{(l)}$，矩阵化 (向量化) 的残差可以表示为

$$\boldsymbol{\delta}^{(l)} = \frac{\partial E}{\partial \boldsymbol{X}^{(l)}} \tag{15}$$

残差的计算过程是先计算输出层的残差，然后从后往前逐层地计算残差，因此被称为反向传播。

以下是训练过程中各层的细节。其中分析的参数梯度和残差是针对利用单个样例进行训练的情况。在实际训练当中，可以用批量或者小批量来进行训练。

1.3 卷积层

卷积层是卷积神经网络的核心结构，作用是通过卷积的方式从前一输入层的多个矩阵中提取出多个带有不同特征信息的矩阵。设网络的第 l 层的卷积层有 D_l 个输出矩阵，其输入层 $l-1$ 层有 D_{l-1} 个输出矩阵，那么 $l-1$ 层的输出矩阵都是 l 层的输入矩阵，并且 $l-1$ 与 l 层之间最多由 $D_{l-1} \times D_l$ 个卷积核连接。通常同一层的卷积核是多个维数相同的方阵。

1.3.1 前向传播

在前向传播过程中，卷积层的每个输出矩阵具体计算方式为

$$X_j^{(l)} = \sum_{i=1}^{D_{l-1}} X_i^{(l-1)} * Ker_{ij}^{(l)} + b_j^{(l)}, \quad j = 1, \ldots, D_l \tag{16}$$

其中 $Ker_{ij}^{(l)}$ 是卷积核，也称为权值，$b_j^{(l)}$ 是偏置系数。

1.3.2 卷积核的梯度

网络输出的误差函数关于卷积核 $Ker_{ij}^{(l)}$ 的梯度的计算方式为

$$\frac{\partial E}{\partial Ker_{ij}^{(l)}} = rot180(X_i^{(l-1)}) * \delta_j^{(l)} \tag{17}$$

其中 $rot180$ 表示将矩阵旋转 180 度，$\delta_j^{(l)}$ 是残差矩阵。每一层都有残差矩阵，其定义为

$$\delta_j^{(l)} = \frac{\partial E}{\partial X_j^{(l)}} \tag{18}$$

1.3.3 偏置的梯度

偏置系数的梯度的计算方式为

$$\frac{\partial E}{\partial b_j^{(l)}} = \sum_{u,v} (\delta_j^{(l)})_{uv} \tag{19}$$

1.3.4 残差反向传播

如果当前卷积层 l 不是第二层，也就是说卷积层的输入层不是整个网络的输入层，那么就需要将残差从 l 层传播到 $l-1$ 层。具体计算方式为

$$\delta_i^{(l-1)} = \sum_{j=1}^{D_l} \delta_j^{(l)} \star rot180(Ker_{ij}^{(l)}) \tag{20}$$

1.3.5 推广形式

这里的前向传播的卷积计算还可以有更一般的形式，比如可以将卷积核的滑动步长 stride 设置为比 1 更大的整数 (在上述定义中，滑动步长为 1)。在前向传递时，可以先将输入矩阵的四周分别补上 p 行 p 列 0 元素来扩展，从而对卷积运算后矩阵的变小进行了补偿。D_{l-1} 和 D_l 层之间的连接也可以修改为部分连接，比如卷积层的一些矩阵只来自输入层的部分输出矩阵。这种情况下卷积核的数量将少于 $D_{l-1} \times D_l$ 个。

1.4 采样层

采样层是卷积神经网络的重要结构。采样也称作池化，其的目的是在尽可能保留特征信息的前提下减小输出矩阵的维度，从而减小网络的计算量并且进一步的将特征信息“抽象化”。将 $l-1$ 层的每个输出矩阵都分别进行采样得到 l 层的各个输出矩阵，因此采样层的输出矩阵数 D_l 与输入的矩阵数 D_{l-1} 相等。设输入矩阵 $X_i^{(l-1)}$ 维数是 $r \times c$，采样窗大小为 $k \times k$，那么 $X_i^{(l)}$ 可以划分为 $\frac{r}{k} \times \frac{c}{k}$ 块的分块矩阵，每个块大小为 $k \times k$。用某种采样方式对每个块进行计算，分别得到一个采样值，因此输出的是一个大小为 $\frac{r}{k} \times \frac{c}{k}$ 的矩阵。

采样方式一般有平均采样、最大采样、随机采样等。

1.4.1 平均采样

在前向传播过程中，平均采样是将分块矩阵中每个矩阵块的平均值所构成的新的矩阵作为采样的输出矩阵。在残差反向传播过程中，前一层的残差矩阵计算方式为

$$\delta_j^{(l-1)} = \frac{1}{k^2} \delta_j^{(l)} \otimes 1_{k \times k} \tag{21}$$

其中 $1_{k \times k}$ 是元素全为 1 的 k 阶矩阵，$\otimes$ 是矩阵的克罗内克积。

1.4.2 最大采样

在前向传播过程中，最大采样是将分块矩阵中每个矩阵块的最大值所构成的新的矩阵作为采样的输出矩阵。用大小和输入矩阵相同的布尔矩阵 P 来记录采样所选取的各个最大值的位置。在残差反向传播过程中，前一层的残差矩阵计算方式为

$$\delta_j^{(l-1)} = P \bullet \delta_j^{(l)} \otimes 1_{k \times k} \tag{22}$$

其中 $\bullet$ 为阿达马乘积。

1.4.3 随机采样

在前向传播过程中，随机采样是先将按照块中每个元素的相对大小来赋予每个位置一个概率，然后依据概率来从块中选择一个值来作为这个块采样值，对于整个分块矩阵而言，就相应有一个采样的输出矩阵。同样地，用大小和输入矩阵相同的布尔矩阵 P 来记录采样所随机选取的各个采样值的位置。在残差反向传播过程中，前一层的残差矩阵计算方式为

$$\delta_j^{(l-1)} = P \bullet \delta_j^{(l)} \otimes 1_{k \times k} \tag{23}$$

1.4.4 推广形式

在传统的卷积神经网络中，使用互不重叠的矩阵块来进行采样，也就是说采样窗的边长等于采样窗的滑动步长。可以考虑设置使采样窗边长大于采样窗的滑动步长，从而进行带有重叠的采样。

1.5 全连接层

在不考虑激活函数的情况下，卷积神经网络中全连接层的处理方法和普通神经网络的基本一致。

1.5.1 前向传播

如果全连接层的输入层的输出是多个矩阵而非一个向量，则先要将各个矩阵变形为列向量，将每个矩阵的列项首尾拼接成一个更长的向量，作为全连接层的输入向量。

$$X^{(l)} = \boldsymbol{W}^{(l)} X^{(l-1)} + \boldsymbol{b}^{(l)} \tag{24}$$

1.5.2 权重和偏置的梯度

$$\frac{\partial E}{\partial \boldsymbol{W}^{(l)}} = \delta^{(l)} X^{(l-1)T} \tag{25}$$

$$\frac{\partial E}{\partial \boldsymbol{b}^{l}} = \delta^{(l)} \tag{26}$$

1.5.3 残差反向传播

需要将全连接层 l 层的残差传播到它的输入层 $l-1$ 层。以下公式给出的是向量形式的残差

$$\delta^{(l-1)} = \boldsymbol{W}^{(l)T} \delta^{(l)} \tag{27}$$

如果 $l-1$ 层的输出不是向量而是多个矩阵，那么需要依照前向传播中矩阵转换为向量的变形方式，将残差向量逆变形为多个矩阵。

1.6 激活层

激活函数 f 是一个实值到实值的非线性映射。为了简洁的表示，将激活函数 f 扩展为向量形式

$$f\left(\begin{pmatrix} x_{11} & \dots & x_{1c} \\ \vdots & & \vdots \\ x_{r1} & \dots & x_{rc} \end{pmatrix}\right) = \begin{pmatrix} f(x_{11}) & \dots & f(x_{1c}) \\ \vdots & & \vdots \\ f(x_{r1}) & \dots & f(x_{rc}) \end{pmatrix} \tag{28}$$

激活函数层输出的形式和输入层的输出一致，即不改变输入的维数和结构。如果激活层 l 层的输入向量形式，那么激活层的输出是相同维数的向量。如果输入是多个矩阵，那么激活层的输出是相同数量的矩阵，并且矩阵的维数不变。前向传播的计算公式为

$$X^{(l)} = f(X^{(l-1)}) \tag{29}$$

相应地，残差反向传播公式为

$$\delta^{(l-1)} = f'(X^{(l-1)}) \bullet \delta^{(l)} \tag{30}$$

1.6.1 Rectified Linear Unit

以往最常用的激活函数是 logistic sigmoid 和 hyperbolic tangent。网络在反向传播求各层参数梯度时利用了链式法则，梯度的计算需要进行一系列的连乘，导致梯度从输出层到输入层逐层衰减 [1]。各层参数训练速度不一致，网络浅层部分几乎无法训练，而深层部分却已经对由随机初始化产生的噪声过拟合，最终网络训练失败。近几年发现将 Rectifier 函数作为卷积神经网络的激活函数能改善深层网络中的梯度消失问题，明显提高训练效果 [2]，而且高效的梯度传播加快了网络的训练速度。使用了 Rectifier 激活函数的神经单元被称为修正线性单元 (ReLU)，Rectifier 函数定义为

$$\begin{aligned} f(x) &= \max(0, x) \\ &= \begin{cases} x & , x > 0, \\ 0 & , x <= 0. \end{cases} \end{aligned} \tag{31}$$

它将非负的输入按原来的值输出，将负输入值输出为 0。它的导函数为

$$\frac{df(x)}{dx} = \begin{cases} 1 & ,x > 0 \\ 0 & ,x < 0 \end{cases} \tag{32}$$

由于它在 $x = 0$ 处不可微，在实际求梯度的时候取 $f'(0) = 0$。从 Rectifier 的导函数可以看到，在单元的输出值为正的时其导数为 1，这样就能保证在训练的残差反向传播过程中，不容易发生梯度逐层衰减。

1.7 输出层

输出层可以看做是特殊的激活函数层。输出函数 f 可以是实值到实值的一元映射，也可以是向量到向量的多元映射，形式为

$$X^{(l)} = f(X^{(l-1)}) \tag{33}$$

输出层的输出向量就是整个网络的输出向量

$$\boldsymbol{y} = X^{(l)} \tag{34}$$

例如，在分类的应用当中可以使用 softmax 函数作为输出函数，其输入是一个向量，输出是经过归一化的向量 (和为 1)。softmax 函数定义为

$$\begin{aligned} X^{(l)} &= f(X^{(l-1)}) \\ &= f(\begin{pmatrix} x_1^{(l-1)} \\ \vdots \\ x_{D_{l-1}}^{(l-1)} \end{pmatrix}) \\ &= \frac{1}{\sum_{i=1}^{D_{l-1}} e^{x_i{}^{(l-1)}}} \begin{pmatrix} e^{x_1^{(l-1)}} \\ \vdots \\ e^{x_{D_{l-1}}^{(l-1)}} \end{pmatrix} \end{aligned} \tag{35}$$

1.7.1 残差反向传播

如果输出层 l 层使用 softmax 函数，那么网络使用交叉熵损失函数

$$E(\theta; X, t) = \sum_{i=1}^{D_l} t_i \ln y_i \tag{36}$$

$l-1$ 层的残差计算公式为

$$\delta^{(l-1)} = X^{(l)} - t \tag{37}$$

如果输出层 l 层使用实值到实值的一元映射，那么网络使用 $\frac{1}{2}$ 的误差平方和作为损失函数残差反向传播计算公式是

$$E(\theta; X, t) = \frac{1}{2}||X^{(l)} - t||^2 \tag{38}$$

$l-1$ 层的残差计算公式为

$$\delta^{(l-1)} = f'(X^{(l-1)})) \bullet (X^{(l)} - t) \tag{39}$$

2 常用技术

以下是一些卷积神经网络中常用的技术。由于篇幅有限，只能做简单的介绍。

2.1 数据预处理

如果使用饱和的激活函数，需要对训练数据进行归一化。由于 Logistic sigmoid 和 hyperbolic tangent 这类激活函数具有饱和的性质，在输入值 x 较大的时候激活函数的导数几乎为 0。输入值达到饱和范围会导致网络无法训练，因此需要将输入数据平移、伸缩到一个 0 附近的较小的范围。一种方法是基于均值和方差的归一化，具体方法是将训练数据的每个维都都减去它的均值并除以它的标准差，从而均值归零方差归一，但不需要对方差为 0 的维度进行方差归一。另一种方法是当在处理色阶为 0-255 的图像时候，可以直接将训练数据每个维都除以 255。

如果使用如 Rectifier 这类非饱和激活函数，则不需要考虑输入值过大带来的激活函数饱和。对于图片处理问题，一般是对训练数据的每个维度进行均值归 0 的平移。

如果在模型训练完毕之后需要对测试数据或者其他新的数据进行预测，则需要将这些输入数据进行相同的预处理，用相同的平移量和伸缩量。

2.2 增大数据量

将大型网络应用于图像识别时，可以通过扩大数据集来减小过拟合程度。是在不改变标签的前提下，对原始的训练数据进行变换，比如将图片平移、水平翻转和轻微旋转。

2.3 为各层设置学习速率

网络不同深度中参数梯度的大小 (量级) 可能不相同，可以给各层的权值系数、偏置系数设置不同的学习速率。一种方法是先设置一个全局的学习速率，再对各层设置相对于全局的学习速率。

2.4 随机小批量梯度下降

使用小批量的样本而非使用完整的样本进行训练，可以显著提高训练的效率。因为每次迭代的计算量和样本大小成正比，使用很小的样本量来进行训练就能在相同时间内完成更多次训练迭代。但是用小样本所计算得到的梯度相对大样本的不稳定，需要相应减小学习速率。另一方面，训练的批量太小也会降低效率，因为会每次迭代时读取训练数据需要额外的时间代价。

2.5 初始化参数

相比普通的神经网路，深度神经网络更需要注重权重的初始化。适当的初值更利于网络的快速收敛，而不适当的初值容易使网络输出数值溢出，从而求解失败。初始化的思想是使每一层的输入和输出的方差尽可能一致，否则可能导致输出值的方差逐层扩大，或者逐层缩小，不利于网络的训练。一般将偏置系数初始化为 0，权值系数则可以按照 XAVIER、MSRA、高斯分布等方法来初始化。

当在对第 l 层的权值初始化时，先对 $l-1$ 层和 l 层进行考察。l 层的每个值的输入是 $l-1$ 层的多个值的通过权值的线性组合得到，记这个输入数为 n_{in}。在反向传播当中，$l-1$ 层的每个值对应的残差都由 l 层的多个残差通过权值的线性组合得到，记这个残差数为输出数 n_{out}。要注意的是各层的 n_{in}, n_{out} 不相同。

2.5.1 XAVIER 初始化方法

Xavier Glorot 和 Yoshua Bengio 提出初始化方法 [1] 是按照均值为 0 方差为 $\frac{2}{n_{in}+n_{out}}$ 的均匀分布来初始化各层权值

$$W \sim U(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}) \tag{40}$$

显然有

$$E[W] = 0 \tag{41}$$

$$Var[W] = \frac{2}{n_{in} + n_{out}}. \tag{42}$$

2.5.2 MSRA 初始化方法

微软亚洲研究院 (MSRA) 的何恺明 (Kaiming He) 等研究员提出了一种针对 ReLU 网络的初始化方法 [3]。用均值为 0 方差为 $\frac{2}{n_{in}}$ 的高斯分布来初始化各层权值。

$$W \sim N(0, \frac{2}{n_{in}}) \tag{43}$$

2.5.3 高斯初始化方法

可以通过预先设置各层权值的方差，然后用零均值和相应方差的高斯分布来初始化各层的权值。

2.6 权重衰减项

梯度下降中使用权重衰减项来对权值进行抑制，可以缓解过拟合的问题

$$\boldsymbol{W} := \boldsymbol{W} - \eta(\frac{\partial E}{\partial \boldsymbol{W}} + \lambda \boldsymbol{W}) \tag{44}$$

其中 λ 是一个预先设置的衰减系数。权重衰减也被称为 $L2$ 正则化项，等价于在原来损失函数 E 中加上一个 $L2$ 正则化项 $\lambda||\boldsymbol{W}||^2$

$$E(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{t}) = \frac{1}{n}\sum_{i=1}^{n} E(\boldsymbol{\theta}, \boldsymbol{X_i}, \boldsymbol{t_i}) + \frac{\lambda}{2}||\boldsymbol{W}||^2 \tag{45}$$

这里 $\boldsymbol{\theta}$ 表示所有参数，$\boldsymbol{W}$ 表示所有的权值参数。

2.7 带动量的梯度优化

梯度下降的参数更新形式为

$$\theta := \theta - \eta\frac{\partial E}{\partial \boldsymbol{\theta}} \tag{46}$$

使用随机梯度下降时利用各个小训练样本计算得到参数梯度的方向可能有较大的差异，导致参数来回震荡。

图 1: 无动量的梯度下降中，参数变化示意图

可以使用带动量 (monentum) 的梯度下降 [7]，预先设置一个动量系数 $m(0 < m < 1)$，并且初始化 $\Delta\theta = 0$，在每次迭代中按照以下的公式来更新参数

$$\Delta\theta := m\Delta\theta + \eta\frac{\partial E}{\partial \theta} \tag{47}$$

$$\theta := \theta - \Delta\theta \tag{48}$$

计算得到的梯度不会直接影响参数 $\boldsymbol{\theta}$，而只是微调梯度动量 $\Delta\boldsymbol{\theta}$。这样能够使参数 $\boldsymbol{\theta}$ 沿着更稳定的方向更新，从而加速收敛。如下图所示

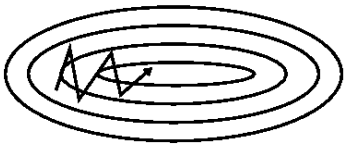

图 2: 带有动量的梯度下降，参数变化示意图

2.8 学习速率衰减

如果使用固定的学习速率，那么网络训练到一定程度后将很难继续训练，损失函数很难继续降低。如果减小学习速率并继续训练，损失函数才能进一步降低。需要使用一个适当的学习速率衰减策略，使得在网络在初期能以较大的步长来训练，而在后期则以较小的步长来训练。一种方法是用迭代次数来划分不同的训练阶段，并在每个训练阶段中设置不同的学习速率。另一种方法是使用指数衰减的方法来使学习速率在每次迭代后都发生少量的衰减。ILSVRC2012 图像识别竞赛中的冠军模型 AlexNet 所使用的衰减策略是，当验证集的错误率不再下降时将学习速率除以 10 作为新的学习速率 [5]。

2.9 Dropout

通过组合不同的模型来进行预测的集成学习方法可以显著提高模型的泛化能力，减小测试的错误率。但是相应带来数倍的训练耗时对于大型神经网络来说是代价高昂的。近几年有一种被称为"Dropout" 的模型组合方式被提出来 [4]。对某个隐层使用"Dropout" 的具体做法是赋予该层每个单元一个参与概率 p。在每次训练迭代中，每个单元有 $(1-p)$ 的概率不参与当次网络训练，也就是临时失效，失效的单元在正向传播中输出值为 0，在反向传播中残差值也是 0。在测试、预测阶段中，采用 Dropout 的层的每个单元的输出值需要乘以 p。

使用了 Dropout 的网络在每次训练迭代中都只有一部分单元生效，网络对每次输入的训练数据都有不同的结构，但是这些所有的权值参数都是共享的，这将提高网络的泛化能力。有研究实验发现取 $p = 0.5$ 时网络具有最好的效果。

2.10 并行计算与 GPU 加速

在大型神经网络的计算过程中，需要进行数量巨大但是结构简单的运算，并且其中的大部分运算都是相互平行互不依赖的。如果将这些运算处理分类到多个计算资源上就可以大大提高网络的运行速度。目前流行的加速方法是使用图形处理器 (GPU) 进行并行计算，这可以将运行速度提高几个数量级。更快的速度不仅是让网络更快地完成训练，更重要的是能显著减小试错成本，加大调试效率。

3 分类应用实例

3.1 MNIST 手写数字图像分类

MNIST 手写数字数据集是含有一个训练数据集和一个测试数据集，分别包含 60000 个和 10000 个手写数字样例，每个样例的输入 $\boldsymbol{X}$ 是一个大小为 28×28 的 8 位色灰度矩阵，其标签是一个 y 是一个 0-9 的整数。本例将标签 y 转换为十类的 1-of-K(one-hot) 编码标签 $\boldsymbol{t}$，对输入的预处理是将每个输入值除以 255 的来，从而伸缩到 0-1 的范围内。

3.1.1 网络结构

第一层是输入层，输出是一个 28×28 的矩阵。

第二层是卷积层，输出是 20 个 24×24 的矩阵，共有 20 个 5×5 的卷积核和 20 个偏置系数。

第三层是采样层，输出是 20 个 12×12 的矩阵，使用最大采样，采样窗大小为 2×2。

第四层是卷积层，输出是 40 个 8×8 的矩阵，共有 $20 \times 40 = 800$ 个 5×5 的卷积核和 40 个偏置系数。

第五层是采样层，输出是 40 个 4×4 的矩阵，使用最大采样，采样窗大小为 2×2。

第六层是全连接层，输出是 200 维的向量，共有 $40 \times 4 \times 4 \times 200 = 128000$ 个权值系数和 200 偏置系数。

第七层是激活层，输出是 200 维的向量，使用的 Rectifier 激活函数。

第八层是全连接层，输出是 10 维的向量，共有 $200 \times 10 = 2000$ 个权值系数和 10 个偏置系数。

第九层是输出层，输出是 10 维的向量 $\boldsymbol{y}$，采用 softmax 函数。

如果要将网络用于预测分类，就需要对网络的输出向量进行规范化，将输出 $\boldsymbol{y}$ 的最大元素设置为 1，其他元素设置为 0，得到预测的标签 $\hat{\boldsymbol{t}}$。

3.1.2 训练策略

采用 batch-size 为 50 的随机梯度下降。

采用 XAVIER 方法来初始化各层的权值参数，用常数 0 来初始化所有偏置参数。

设置全局的学习速率的初始值为 0.01，并且令权值参数的学习速率等于全局学习速率，偏置参数的学习速率等于全局学习速率的两倍。对全局学习速率采用指数衰减的策略，其在每次迭代中都衰减一点，半衰期为 3 个 epochs。

训练前预先从训练集中分割了大小为 1000 的验证集，只用 59000 的训练集进行训练，并且每 200 次迭代都利用验证集对模型进行测试。

设置训练数据集的训练次数上限是 20 个 epochs，并且设置当最近一个 epoch 内的平均交叉熵小于 0.002 时自动停止训练。

3.1.3 结果

在进行第 14 个 epoch 的训练时，损失函数就达到了 0.002 以内。使用大小为 10000 的测试集对模型进行测试评估，分类的正确率为 99.14%。使用 AMD 760K 的 CPU 计算，训练耗时约 3 小时。损失函数值和验证集正确率变化的函数图像见图 3。

如果迭代条件损失函数阈值设置为 0.001，则需要在第 17 个 epoch 的训练时，才能达到停止条件。测试集的分类的正确率为 99.09%。额外的训练没有提高准确率，结合图 3中验证集正确率变化图像来考虑考虑，可以训练已经达到饱和状态。

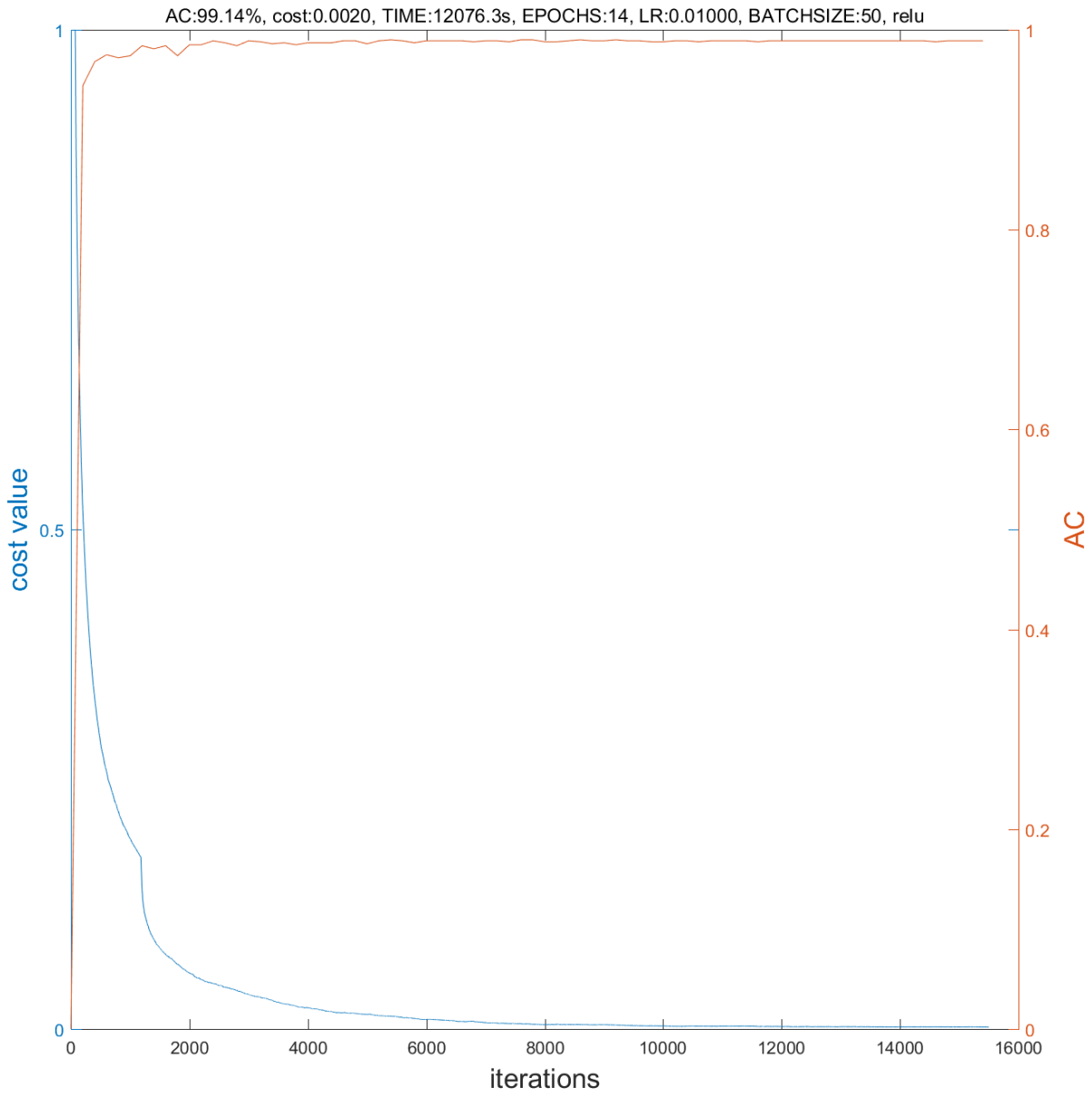

图 3: MNIST 训练过程，损失函数值 (蓝色) 和验证集正确率 (红色) 随迭代次数变化的函数图像

3.2 CIFAR-10 图像分类

CIFAR-10 是图片数据集是 60000 万张大小为 32×32 的彩色图片，每张图片属于十个物体类别中的一类，并且有相应的标签。其中有 50000 张训练图片，有 10000 张测试图片。本例采用 1-of-K 的编码，预处理的方法是将所有输入数据的每个维度都减去训练集相应维度的均值，即基于均值的归一化。由于图片的输入是有红、绿、蓝 (RGB) 三个颜色通道的，因此每个图片的输入数据是 3 个 32×32 的矩阵。

3.2.1 网络结构

本例中的卷积神经网络和上述 MNIST 例子的在细节上略有区别。使用先在四周补 0 扩展再进行卷积，从而使卷积层的矩阵和大小与其输入的矩阵一致。允许重叠的采样，用采样窗大小 f 和采样窗移动步长 s 来控制每个输出元素的采样范围。采样层输出矩阵的边长计算方式为 $d_{out} = \lceil \frac{d_{in}-f}{s} \rceil + 1$。

第一层是输入层，输出是 3 个 32×32 的矩阵。

第二层是卷积层，输出是 24 个 32×32 的矩阵，共有个 $3 \times 24 = 72$ 个 5×5 的卷积核和 24 个偏置系数。

第三层是采样层，输出是 24 个 16×16 的矩阵，采样窗大小为 3×3，采样窗的移动步长为 2。使用最大采样。

第四层是激活层，输出是 24 个 16×16 的矩阵。

第五层是卷积层，输出是 32 个 16×16 的矩阵，共有 $24 \times 32 = 768$ 个 5×5 的卷积核和 32 个偏置系数。

第六层是激活层，输出是 32 个 16×16 的矩阵。

第七层是采样层，输出是 32 个 8×8 的矩阵，采样窗大小为 3×3，采样窗移动步长为 2。使用平均采样。

第八层是卷积层，输出是 64 个 8×8 的矩阵，共有 $32 \times 64 = 2048$ 个 5×5 的卷积核和 64 个偏置系数。

第九层是激活层，输出是 64 个 8×8 的矩阵。

第十层是采样层，输出是 64 个 4×4 的矩阵，采样窗大小为 3×3，采样窗移动步长为 2。使用平均采样。

第十一层是全连接层，输出是 64 维的向量，共有 $64 \times 4 \times 4 \times 64 = 65536$ 个权值系数和 64 个偏置系数。

第十二层是激活层，输出是 64 维的向量。

第十三层是全连接层，输出是 10 维的向量，共有 $64 \times 10 = 640$ 个权值系数和 10 个偏置系数。

第十四层是输出层，输出是 10 维的向量 $\boldsymbol{y}$，采用 softmax 函数。

3.2.2 训练策略

采用 batch-size 为 10 的随机梯度下降。

用标准差为 0.0001,0.0001,0.001,0.01,0.1 的零均值正态分布分别初始化第一、二、三个卷积层和第一、二个全连接层的权值参数。用常数 0 来初始化其余所有的偏置系数。

设置全局的学习速率初始值为 0.0005，并且令权值参数的学习速率等于全局学习速率，偏置参数的学习速率等于全局的两倍。对全局学习速率采用指数衰减的策略，半衰期为 5 个 epochs。

采用带动量的梯度下降，动量参数为 0.9。

训练前从训练集中分割了大小为 1000 的验证集，用剩下大小为 49000 的训练集进行训练。每迭代 4900 次时利用验证集进行一次测试。

设置训练数据集的训练次数上限为 30 个 epochs。

3.2.3 结果

网络在达到迭代次数的上限后停止训练，最后的交叉熵为 0.2572，测试集的正确率为在 74.90%。使用 Intel i7 6800K 的 CPU 进行计算，训练耗时接近 28 小时。损失函数值和验证集正确率变化的函数图像见图 4

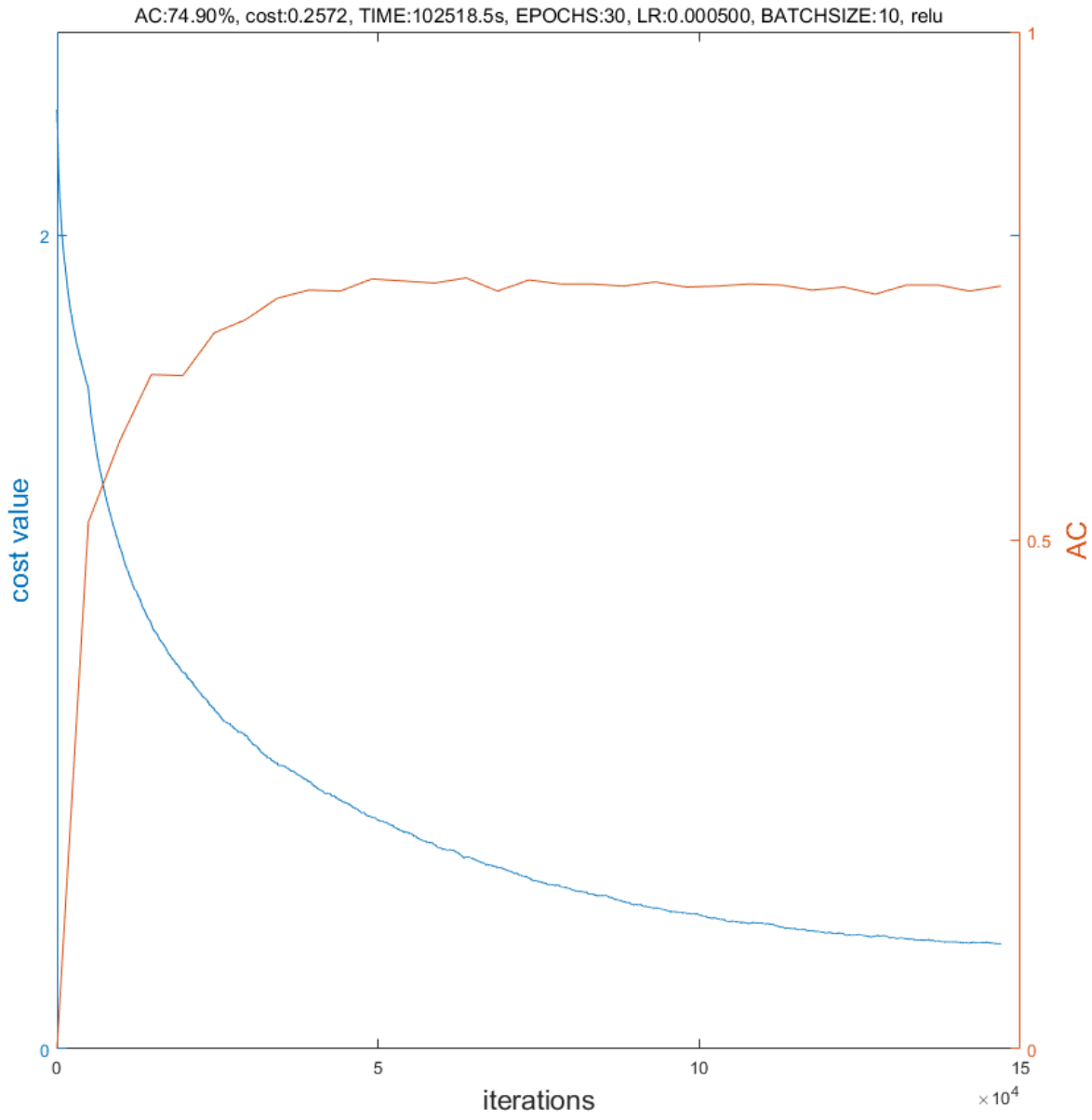

图 4: CIFAR-10 训练过程，损失函数值 (蓝色) 和验证集正确率 (红色) 随迭代次数变化的函数图像

4 结论

普通神经网络在 MNIST 上的测试正确率的上限为 96% 左右，而使用了卷积神经网络可以将正确率提高到 99% 以上，这说明了卷积神经网络的特殊结构确实能带来提升。但是手写数字集过于简单，不足以体现出卷积神经网络的优势。

在 CIFAR-10 上的图像分类问题当中，普通神经网络的测试正确率只能上升到 50% 左右，随着继续训练正确率将开始下降，这说明已经发生了过拟合。一个原因是全连接的方式使其单元之间的每个连接都有一个独立的参数。大型的全连接网络会含有非常大数量的可训练参数，如果训练数据不够庞大，很容易发生过拟合。

卷积神经网络采用了卷积的连接方式，卷积层的每对输入矩阵和输出矩阵之间仅由一个卷积核连接。这种权值共享的连接方式使可训练参数的数量远远低于连接的数量。另一方面，卷积的方式使得卷积层更关注于输入矩阵中每一小块的区域，从而利于提从图像中提取出局部的特征。由于采样的方式，采样结果对输入矩阵中特征的轻微平移和变形是不敏感的。这些结构都有助于提高网络的泛化能力。

在计算方面，卷积神经网络中具有非常大量的连接数，这直接决定了前向传播和反向传播的计算工作量。由于 CPU 的线程数较少，即使高性能的 CPU 也远远无法满足需求大型网络的训练需求。应该用 GPU 来进行并行计算。

参考文献

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[2] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[4] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[6] Yann Lecun, L Eon Bottou, Yoshua Bengio, and Patrick Haaner. Gradient-based learning applied to document recognition. 1998.

[7] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.