人工神经网络实验报告
卷积神经网络 (CNN)

张钰晖
2015011372, yuhui-zh15@mails.tsinghua.edu.cn, 185-3888-2881

目录

1 问题描述

机器学习数字分类领域广泛使用 MNIST 训练集，该集合包含 60000 个训练样例和 10000 个测试样例。每一个样例是 784×1 的列向量，该列向量可以被还原为原始的 28×28 的灰度图。下面展示了其中的几张样本。

图 1: MNIST 数据样本

在这次作业中，笔者将使用卷积神经网络 (CNN) 来完成数字分类任务，通过不同参数的设置来提高任务的准确率。

2 卷积神经网络

卷积神经网络是最重要的一类神经网络，在图像处理与信号处理等领域发挥着极其重要的作用。

在图像处理任务中，卷积神经网络 (CNN) 相比多层感知器 (MLP) 而言，具有以下优势：

- 参数数量大幅减少：由于多层感知器为全连接网络，网络参数极多。而卷积神经网络通过局部连接与权值共享，极大地减少了网络参数。通过池化，进一步减少参数数量，提升模型鲁棒性。
- 利用像素之间的位置信息：对于图像识别任务而言，每个像素与其周围像素的信息比离得较远的像素的信息重要很多，卷积神经网络通过局部连接解决了这个问题。

卷积神经网络主要由卷积层、池化层和全连接层构成，下图展示了卷积神经网络的基本结构：

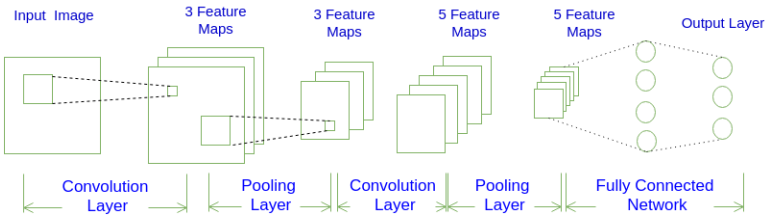


图 2: 卷积神经网络结构

3 细节实现

CNN 的基本特性是局部连接与权值共享。

卷积神经网络 (CNN) 的实现相比多层感知器 (MLP) 要复杂很多，需要清楚地理解原理。因为作业提供了较为完美的实验框架，实现了绝大部分代码，本次任务只需修改相应的函数即可。

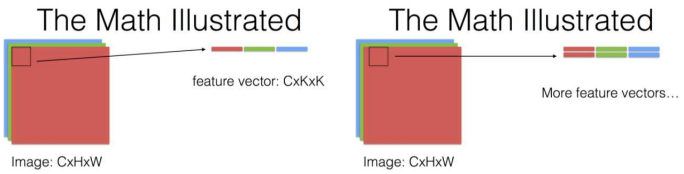
由于这部分不作为重点，且向量化推导将占去大段篇幅，故仅进行了简要介绍。笔者在代码注释中详细记录了每一步矩阵的大小变换，读者可以参考代码去理解。

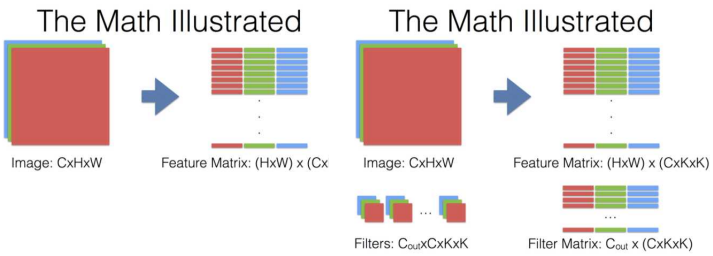
3.1 卷积层的实现

3.1.1 向前传递

- 输入：input($n * c\_in * h\_in * w\_in$), W($c\_out * c\_in * k * k$), b(c_out), kernel_size, pad
- 输出：output($n * c\_out * h\_out * w\_out$)

借助 im2col 实现，最终转化为矩阵乘法，im2col 的原理如下：





尽管笔者采用了 for 循环的方式实现了 im2col，但通过 cython 进行了加速，最终效率较高。

3.1.2 向后传递

- 输入：input($n*c\_in*h\_in*w\_in$), grad_output($n*c\_out*h\_out*w\_out$), W(c_out*c_in*k*k), b(c_out), kernel_size, pad
- 输出：grad_input($n*c\_in*h\_in*w\_in$), grad_W(c_out*c_in*k*k), grad_b(c_out)

先借助 im2col 实现梯度计算，再借助 col2im(im2col 的逆变换) 变换回原始矩阵，原理不再赘述。值得注意的是，由于偏导数计算需要，col2im 是累加实现，而不是完全还原 im2col。

3.2 池化层的实现

3.2.1 向前传递

- 输入：input($n*c\_in*h\_in*w\_in$), kernel_size, pad
- 输出：output($n*c\_in*h\_out*w\_out$)

借助 numpy 向量变换实现，通过对向量进行 resize 后对特定维度取 mean，非常简洁的实现了前向传递。

3.2.2 向后传递

- 输入：input($n*c\_in*h\_in*w\_in$), grad_output($n*c\_out*h\_out*w\_out$), kernel_size, pad
- 输出：grad_input($n*c\_in*h\_in*w\_in$)

同借助 numpy 向量变换实现，通过对向量进行 repeat 后除以 kernel_size 的平方，非常简洁的实现了后项传递。

3.3 损失函数的实现

3.3.1 向前传递

- 输入：input($batch\_size * output\_layer\_size$), target($batch\_size * output\_layer\_size$),
- 输出：output($1 * 1$)

SoftmaxCrossEntropyLoss 的定义如下：

$$SoftmaxCrossEntropyLoss = -\frac{1}{N}\sum_{k}^{N} t_k \log p_k$$

其中

$$p_k = \frac{e^{y_k}}{\sum_m e^{y_m}}$$

3.3.2 向后传递

- 输入：input($batch\_size * output\_layer\_size$), target($batch\_size * output\_layer\_size$),
- 输出：output($batch\_size * output\_layer\_size$)

对损失函数进行求导，利用线性代数相关的知识，将损失函数的表达式高度向量化。

4 训练曲线

4.1 训练曲线

训练曲线刻画了准确率 Accuracy 与损失 Loss 随训练次数变化的曲线。下图展示了双层卷积神经网络的训练曲线：

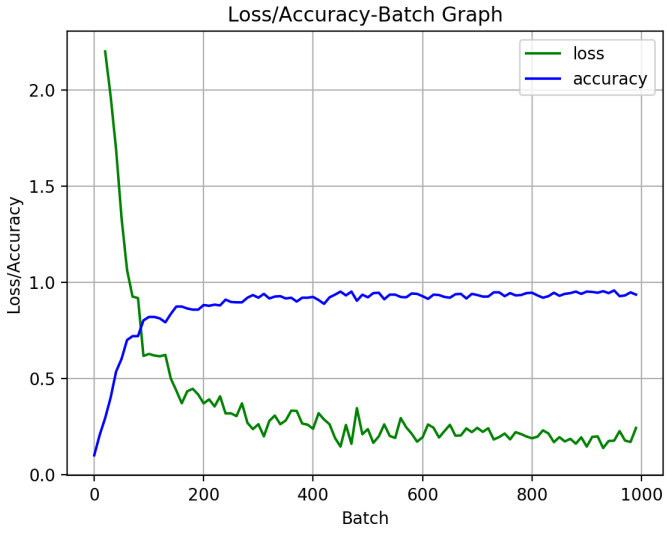


图 3: 1000 个 Batch 的训练曲线（准确率为训练集准确率）

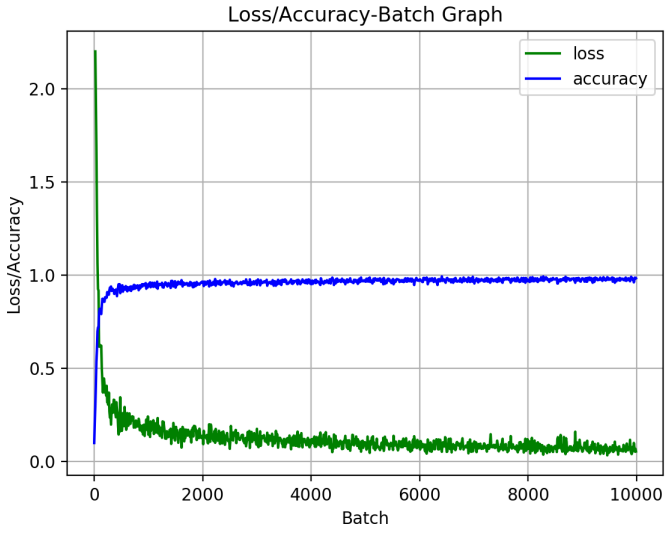


图 4: 10000 个 Batch 的训练曲线（准确率为训练集准确率）

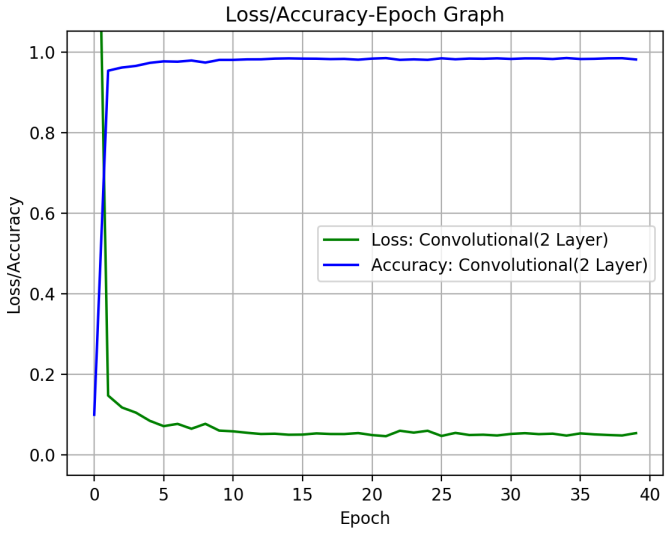


图 5: 40 个 Epoch 的训练曲线（准确率为测试集准确率）

4.2 网络参数

对应的网络参数如下：

表 1: 网络结构

层	参数	规模
Convolutional	C In: 1	输入: $N * 1 * 28 * 28$
	C Out: 4	输出: $N * 4 * 28 * 28$
	Kernel Size: 5	
	Padding: 2	
Relu		输入: $N * 4 * 28 * 28$
		输出: $N * 4 * 28 * 28$
AveragePool	Kernel Size: 2	输入: $N * 4 * 28 * 28$
	Padding: 0	输出: $N * 4 * 14 * 14$
Convolutional	C In: 4	输入: $N * 4 * 14 * 14$
	C Out: 8	输出: $N * 8 * 14 * 14$
	Kernel Size: 3	
	Padding: 1	
Relu		输入: $N * 8 * 14 * 14$
		输出: $N * 8 * 14 * 14$
AveragePool	Kernel Size: 2	输入: $N * 8 * 7 * 7$
	Padding: 0	输出: $N * 8 * 7 * 7$
Linear		输入: $N * 392$
		输出: $N * 10$

表 2: 网络参数

激活函数	Relu
损失函数	SoftmaxCrossEntropyLoss
其他参数	Learning Rate: 1e-1
	Weight Decay: 1e-3
	Momentum: 1e-3
	Batch Size: 50

4.3 结果分析

这是所有测试的参数中结果最好的一组网络，其最终准确率达到了 98.54%（30 个 Epoch 之内），较好的完成了图像识别任务。

从图中可以看出，CNN 收敛速度很快（快于 MLP），迭代次数较少时损失 Loss 下降迅速，准确率 Accuracy 提升迅速。随着迭代次数的增加，损失 Loss 下降速度变得平缓，准确率 Accuracy 提升速度也变得平缓，并逐步收敛，最终稳定在收敛值附近。

5 与 MLP 对比

5.1 对比结果

下面这张表格从网络结构（包含了网络连接信息，舍去了部分不太重要的信息，如超参数）、参数数量、训练结果三方面对比了三种网络，所有网络均训练 40 个 Epoch：

表 3: 网络参数

	双层 CNN	**双层 MLP**	**单层 MLP**
网络结构	(卷积, 激活, 池化) 双层卷积网络	(784, 400, 200, 10) 双隐层网络	(784, 400, 10) 单隐层网络
激活函数	Relu	Relu	Relu
损失函数	SoftmaxCrossEntropy	SoftmaxCrossEntropy	SoftmaxCrossEntropy
参数数量	4330	396210	318010
准确率	98.54%	98.33%	98.00%
训练速度	很慢 (约 180s/epoch)	较快 (约 8s/epoch)	很快 (约 5s/epoch)
收敛速度	快	快	快
收敛性	好	好	好

5.2 训练曲线

下面展示了三种网络的训练曲线，所有网络均训练 40 个 Epoch：

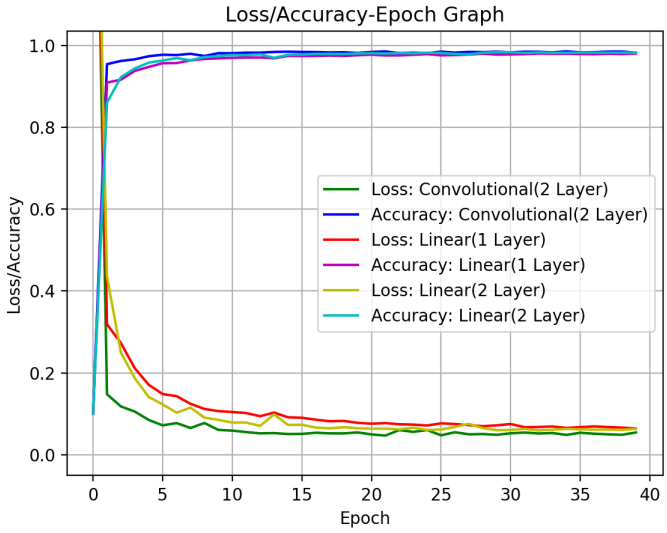


图 6: 三种网络训练曲线（准确率为训练集准确率）

5.3 结果分析

以上选取了三种网络结构进行对比，均采用 Relu 激活函数，这是因为 Relu 函数作为激活函数有以下优势：

- 速度快：Relu 函数计算简单，只需和 0 计算 max 即可。
- 减轻梯度消失问题：Sigmoid 函数导数最大值是 0.25，用 Sigmoid 函数作为激活函数会导致梯度越来越小，对于深层网络训练是个很大的问题，而 Relu 函数的导数是 1，不会导致梯度减小。

通过不同方面的对比，我们可以清楚地看到 CNN 与 MLP 的异同。

- 从参数数量可以看出，由于局部连接、权值共享和池化，处理同样的问题 CNN 参数数量和 MLP 相比大为下降，远远少于 MLP。以双层 CNN 计算为例，参数数量为 $4*1*5*5+4+8*4*3*3+8+392*10+10=4330$，而双层 MLP 参数数量为 $784*400+400+400*200+200+200*10+10=396210$。
- 从训练结果可以看出，CNN 和 MLP 的准确率都较高，收敛性都较好，收敛速度都较快（此处收敛速度是指准确率稳定时所需的训练次数），相对而言 CNN 的收敛速度快于 MLP。

- 从准确率可以看出，CNN 高于双层 MLP 高于单层 MLP，在 40 个 Epoch 时，CNN 和 MLP 训练都并未充分（训练 100 个 Epoch 时双层 MLP 准确率可达 98.69%），但由于训练速度很慢，故只训练了 40 个 Epoch。
- 从训练速度可以看出（此处训练速度是指训练一个 1Epoch 所需时间），CNN 训练速度相比 MLP 而言非常慢，这是因为 CNN 卷积计算复杂度很高，需要进行大量计算。

6 可视化

可视化中间状态能帮助我们更好的理解卷积神经网络，以下对第一层卷积后经过 Relu 激活的中间状态进行可视化。

6.1 输出结果

6.1.1 训练 0 个 Epoch

测试集准确率：7.4%

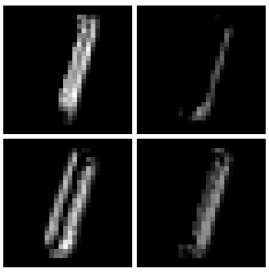


图 7: 1(预测 8)

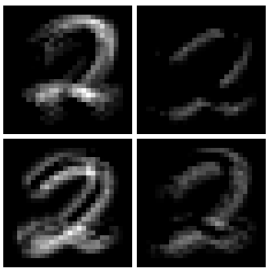


图 8: 2(预测 8)

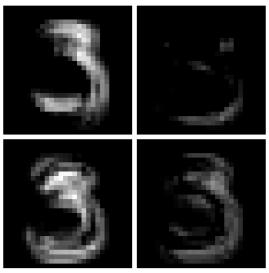


图 9: 3(预测 8)

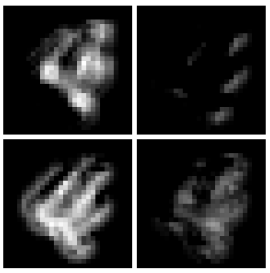


图 10: 4(预测 8)

6.1.2 训练 5 个 Epoch

测试集准确率 62.4%

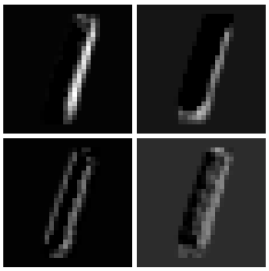


图 11: 1(预测 8)

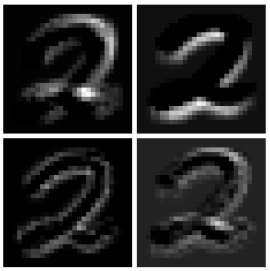


图 12: 2(预测 2)

图 13: 3(预测 3)

图 14: 4(预测 8)

6.1.3 训练 25 个 Epoch

测试集准确率：98.5%

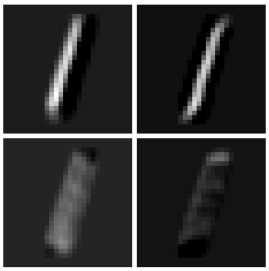


图 15: 1(预测 1)

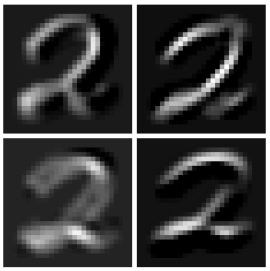


图 16: 2(预测 2)

图 17: 3(预测 3)

图 18: 4(预测 4)

6.2 结果分析

训练卷积神经网络的本质是训练卷积核的过程，卷积核用于正确的提取图像信息。从可视化结果可以非常清晰地看出，随着训练次数的增多，卷积核提取图像信息的能力不断增强，从而完成识别。

6.3 其他部分可视化

以下对第二层卷积后经过 Relu 激活的中间状态进行可视化。(训练 25 个 Epoch)

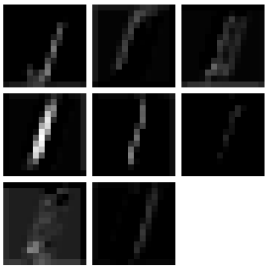


图 19: 1(预测 1)

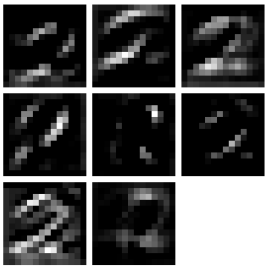


图 20: 2(预测 2)

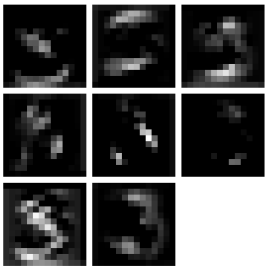


图 21: 3(预测 3)

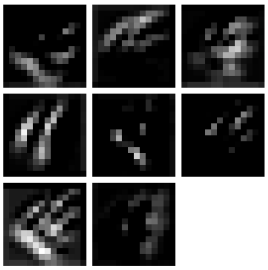


图 22: 4(预测 4)

7 实验心得

通过本次实验, 我大大加深了对卷积神经网络 (CNN) 的理解, 尽管实现过程是痛苦的, 但在动手推导与实现卷积层的前向传播与后向传播算法时, 我感受到了数学与算法的美, 初步掌握了 CNN 的搭建与参数选择技巧, 更加深入的理解了神经网络和机器学习。通过并不是足够充分训练, 识别准确率可以达到 98.54%。

CNN 的局部连接与权值共享可以联系到生物的感受野, 这让我对生物学也产生了及其浓厚的兴趣。

笔者最初通过 for 循环实现了 im2col, 发现速度极慢, 后来尝试使用了 cython 进行加速, 仍然很慢。最终笔者发现 array[a,b,c,d] 的 numpy index 方法远远快于 array[a][b][c][d], 这一微小的改动使得速度快了至少 10 倍！但通过对比 Tensorflow 和 Theano 的 CNN 运行速度, 我也感受到机器学习框架的底层不知道经历了多少了多少精妙绝伦的优化才能达到这样的速度, 训练速度和这些主流框架相比, 真是贻笑大方。

对我而言, 这依旧只是一个入门, 还需要进一步的练习与实践。

由于时间和精力有限,后期可以进行更充分的训练、尝试更多的参数以及对比 RNN、LSTM 等不同的神经网络。

参考文献

[1] 人工神经网络, 黄民烈, 2017.

[2] 零基础入门深度学习, *Han Bingtao*, 2017.

[3] Caffe 计算卷积, 贾扬清, 2015.