项目1

用卷积神经网络识别数字

截止日期：1月17日（2023年）23:59

1. 说明

除非注明，这些说明也适用于所有其余项目。请仔细阅读它们。

1. 我们鼓励学生讨论项目。然而，每个学生都需要自己写代码和报告。代码不应该被分享或复制。除非允许，否则不要使用外部代码。显然，看一下代码，然后用非常相似的编码结构重新实现是不行的。
2. 将问题发布到Canvas上，以便每个人都能分享，除非问题是私人问题。如果有类似的问题被发布，请先查看Canvas。对于私人问题，请使用Canvas:Inbox。
3. 当你打算使用自由晚报时，请在第一页写明/写在文章的顶部（或紧随标题之后）。
4. 需要上传两个文件。首先，你的文章（主要文件）必须命名为{你的-SFUID}.pdf并上传到Canvas。第二，必须将一个压缩包上传到Canvas，其目录结构如下（其他项目会有所不同，但也会类似）。
    - {SFUID}/
        - 生态
            - ec.m
        - Matlab/
            - col2im_conv.m
            - col2im_conv_matlab.m
            - conv_layer_backward.m
            - conv_layer_forward.m
            - conv_net.m
            - convnet_forward.m
            - get_lenet.m
            - get_lr.m
            - im2col_conv.m
            - im2col_conv_batch.m
            - init_convnet.m

        - inner_product_backward.m
        - inner_product_forward.m
        - load_mnist.m
        - mlrloss.m
        - pooling_layer_backward.m
        - pooling_layer_forward.m
        - 芦笙.m
        - relu_backward.m
        - sgd_momentum.m
        - test_components.m
        - test_network.m
        - train_lenet.m
        - vis_data.m
        - lenet_pretrained.mat
        - 硕士所有.mat
    - 项目1有13分。
5. 文件路径。确保你使用的任何文件路径都是相对的，而不是绝对的，这样我们就可以很容易地在我们这一端运行代码。例如，你不能写 "imread('/some/absolute/path/data/abc.jpg')"。写 "imread('./data/abc.jpg') "代替。

2. 概述

在这项任务中，你将实现一个卷积神经网络（CNN）。你将建立一个在MNIST数据集上训练的数字字符识别系统。

我们首先简要介绍一下结构和功能。关于更多的细节，你可以参考在线资源，如 http://cs231n.stanford.edu。请注意，这项作业的编码量比其他作业要少得多。我们不会提供详细的说明，希望大家能在网上搜索和/或逆向工程模板代码。

一个典型的卷积神经网络有四个不同类型的层。

全连接层/内部产品层（IP）

全连接层或内积层是构成神经网络的最简单层。该层的每个神经元都与上一层的所有神经元相连（见图1）。在数学上，它是通过矩阵乘法和增加一个偏置项来模拟的。对于一个给定的输入x，全连接层的输出由以下公式给出。

$$f(x) = W x + b$$

W、b是该层的权重和偏置。W是一个m×n大小的二维矩阵，其中n是上一层的维度，m是这一层的神经元数量。b是一个大小为m×1的向量。

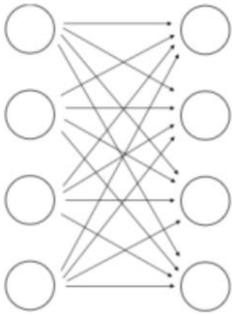


Figure 1: Fully connected layer

卷积层

这是CNN的基本构建块。在我们深入了解什么是卷积层之前，让我们对卷积做一个简单的回顾。

正如我们在讲座中所看到的，卷积是用一个k×k的滤波器/内核和一个W×H的图像进行的。卷积操作的输出是一个特征图。这个特征图可以根据所使用的滤波器而具有不同的含义--例如，使用高斯滤波器会导致图像的模糊版本。在x和y方向上使用索贝尔滤波器会给我们相应的边缘图作为输出。

术语：滤波器中的每个数字都将被称为滤波器权重。例如，3x3高斯滤波器有以下9个滤波器权重。

$$W = \begin{pmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & 0.6193 & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{pmatrix}$$

当我们进行卷积时，我们决定我们要使用的确切的过滤器类型，并相应地决定过滤器的权重。CNN试图从数据中学习这些过滤器的权重和偏差。我们试图为每个卷积层学习一组过滤器。

一般来说，使用卷积层而不是全连接（FC）层（如神经网络中使用的）有两个主要动机

1. **参数的减少**

   在FC层中，一个层中的每个神经元都与前一个层中的每个神经元相连。这导致了大量的参数需要估计--这导致了过度拟合。CNN通过共享权重来改变这种情况（同一滤波器在整个图像上被转换）。
2. **它利用了空间结构**

   图像具有固有的二维空间结构，当我们将图像展开为一个矢量并将其送入一个普通的神经网络时，就会失去这种结构。卷积就其本质而言是一种二维操作，对空间上接近的像素进行操作。

实施细节。一般的卷积操作可以用以下公式表示。

$$f(X, W, b) = X * W + b$$

其中W是大小为$k \times k \times Ci$的滤波器，X是大小为$Ni \times Ni \times Ci$的输入量，b是1×1元素。各个术语的

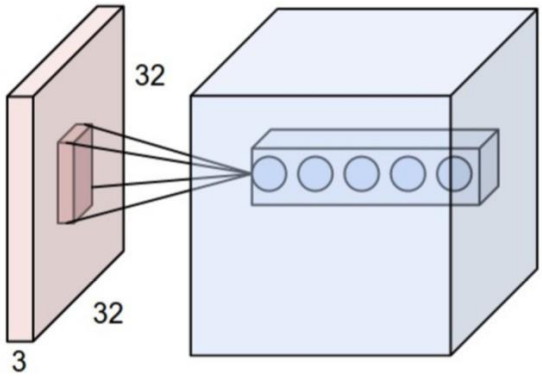


Figure 2: Input and output of a convolutional layer (Image source: Stanford CS231n

含义如下所示。

在下面的例子中，下标i是指该层的输入，下标o是指该层的输出。

- Ni - 输入图像的宽度
- Ni - 输入图像的高度（图像有一个正方形）。
- Ci - 输入图像中的通道数
- ki - 滤波器的宽度
- si - 卷积的步长
- pi - 输入图像的填充像素的数量
- num - 要学习的卷积滤波器的数量

一幅灰度图像有1个通道，这就是图像体积的深度。对于一个有c_i通道的图像，我们将学习n个大小为$k_i \times k_i \times c_i$的滤波器。与每个滤波器进行卷积的输出是一个高度和宽度为N_o的特

$$N_o = \frac{N_i - k_i + 2p_i}{s_i} + 1$$

征图，其中

如果我们把num特征图堆叠起来，我们可以把卷积的输出当作另一个三维体积/图像，c_o = num通道。

总之，卷积层的输入是一个尺寸为$N_i \times N_i \times c_i$的体积，输出是一个尺寸为$N_o \times N_o \times$num的体积。图2显示了一个图形化的图片。

集合层

池化层一般用于卷积层之后，以减少特征图的大小。池化层分别对每个特征图进行操作，并将特征图的一个局部区域替换为一些聚集的统计数据，如最大值或平均值。除了减少特征图的大小外，它还使网络对小的平移不敏感。这意味着，当物体稍微移动时，该层的输出不会改变。

在这个任务中，我们将只使用图3所示的MAX池化层。这种操作的方式与卷积相同，但我们不是应用滤波器，而是在每个核中找到最大值。让k代表核的大小，s代表跨度，p代表填充。那么，应用于填充后的特征图X的池化函数f的输出由以下方式给出。

$$f(X, i, j) = \max_{x \in [i-k/2, i+k/2], y \in [j-k/2, j+k/2]} (X[x, y])$$

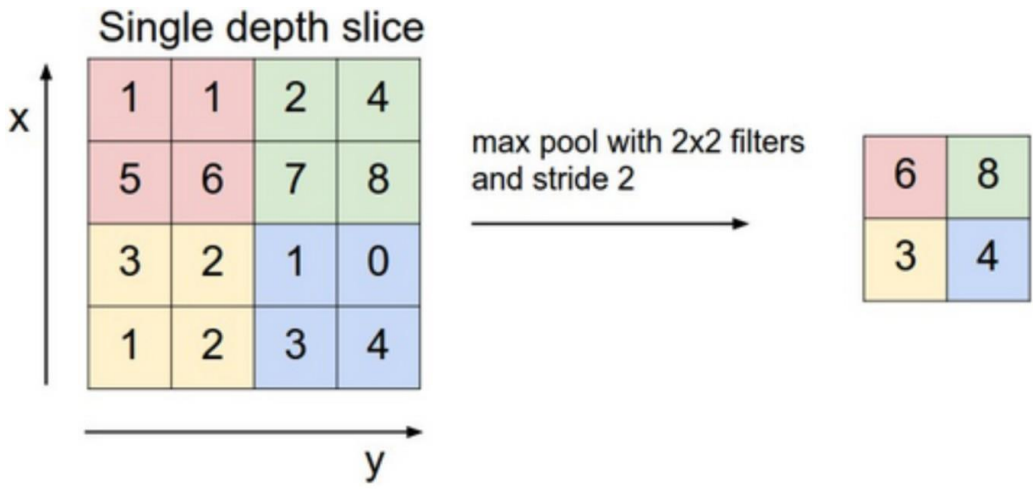


Figure 3: Example MAX pooling layer

激活层--ReLU--整顿线性单元

激活层在网络中引入了非线性，并赋予其学习复杂函数的能力。最常用的非线性函数是ReLU函数，定义如下。

$$f(x) = \max(x, 0)$$

ReLU函数对前一层的每个输出进行操作。

损失层

损失层有一个全连接层，其神经元的数量与类的数量相同。然后，为了将输出转换为概率分数，使用了一个softmax函数。该操作由以下公式给出。

$$p = \text{softmax}(W x + b)$$

其中，W的大小为C×n，n是前一层的维度，C是问题中的类的数量。

该层还计算损失函数，该函数在训练过程中要最小化。实践中最常用的损失函数是交叉熵和负对数可能性。在这个任务中，我们将只是最小化给定标签的负对数概率。

建筑学

在这项作业中，我们将使用一个基于非常流行的网络的简单架构，称为LeNet（http://ieeexplore.ieee.org/abstract/document/726791/）。

- 输入 - 1×28×28
- 卷积 - k = 5, s = 1, p = 0, 20个过滤器
- リオンドライト
- MAXPooling - k=2, s=2, p=0
- 卷积 - k = 5, s = 1, p = 0, 50个过滤器
- リオンドライト
- MAXPooling - k=2, s=2, p=0
- 全连接层 - 500个神经元
- リオンドライト
- 损失层

请注意，所有类型的深度网络都对其隐藏层使用非线性激活函数。如果我们使用线性激活函数，那么隐藏层对最终结果没有影响，这将成为输入值的线性（仿射）函数，可以用一个没有隐藏层的简单2层神经网络来表示。

文献中使用了很多标准的卷积神经网络架构，例如，AlexNet、VGG-16或GoogLeNet。它们在参数的数量和配置上都有所不同。

3. 编程

实现CNN的大部分基本框架已经被提供。你将需要填入一些函数。在进入实现之前，你将需要了解代码中使用的数据结构。

数据结构

我们定义了四个主要的数据结构来帮助我们实现卷积神经网络，这些结构将在下一节中解释。每个层由一个数据结构定义，其中字段类型决定了该层的类型。这个字段可以取

值为DATA、CONV、POOLING、IP、RELU、LOSS，对应于数据。

分别是卷积层、最大集合层、内积/全连接层、ReLU和损失层。每个层中的字段将取决于层的类型。

输入是以一个带有以下字段的结构传递给每一层的。

- height - 特征图的高度
- width - 特征图的宽度
- 通道 - 通道/特征图的数量
- batch size - 网络的批次大小。在这个实现中，你将实现迷你批次的随机梯度下降来训练网络。这背后的想法非常简单，我们不是在每张图片后计算梯度和更新参数，而是在看完一批图片后再做。这个参数的批次大小决定了在更新参数之前它要看多少张图像。
- data - 存储在各层之间传递的实际数据。它的大小应该是[高度×宽度×通道，批量大小]。你可以在计算过程中调整这个结构的大小，但要确保将它恢复为一个
  二维矩阵。数据是按列大顺序存储的。接下来是行，最后是通道。
- diff - 存储相对于数据的梯度，它的大小与数据相同。每个层的参数都存储在一个结构param中。在前向传递中，你不需要接触这个。
- w - 该层的权重矩阵
- b - 偏见

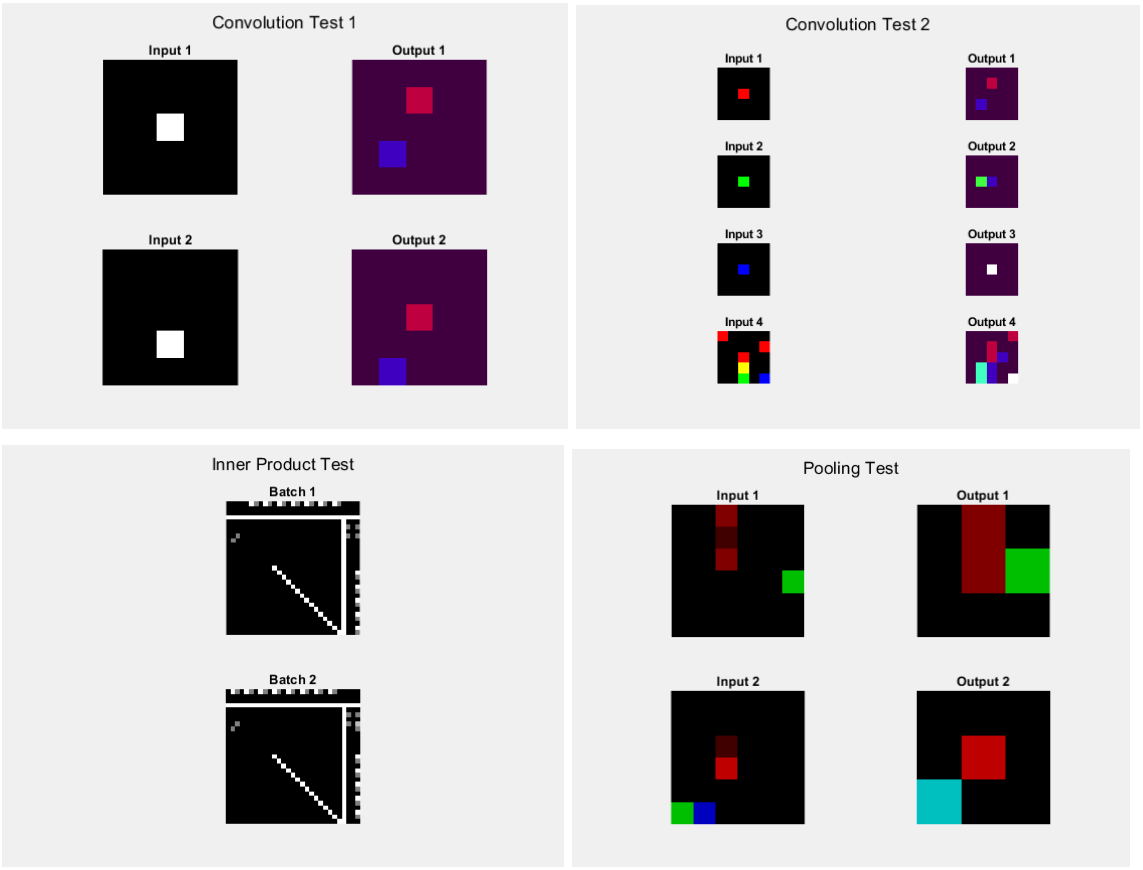
param_grad用于存储每层耦合的梯度，其属性如下。

- w - 存储损失与w有关的梯度。
- b - 存储相对于偏置项的损失梯度。

第一部分。正面传球

现在我们将开始实现网络的前向传递。每个层都有一个非常相似的原型。每个层的前向函数都以输入、层、参数作为参数。输入存储的是输入数据以及关于其形状和大小的信息。层存储了该层的规格（例如，对于一个conv层，它将有k、s、p）。params是一个可选的参数，传递给有权重的层。它包含用于计算输出的权重和偏置。在每一个前向传递函数中，你都应该使用这些参数并计算输出。你应该在从函数返回之前填写输出的高度、宽度、通道、批量大小、数据字段。还要确保数据字段已被重塑为一个二维矩阵。

在过去，我们要求为每一个单一的步骤提供一些可视化的结果。然而，在我们实现所有层的前向功能之前，是没有意义的可视化的。一旦你实现了所有的层，运行test_components.m，然后把可视化的结果复制/粘贴到报告中。这些图像应该看起来像下面这样。(test_components.m是由SFU2019秋季班的**Matthew Marinets**提供的）。可视化的结果不完全相同也没关系。



Q 1.1 内部产品层 - 1分

全连接层的内积层应该用以下定义来实现

[output] = inner_product_forward(input, layer, param)

问题1.2 集合层 - 1分

编写一个实现池化层的函数，其定义如下。[output] = pooling_layer_forward(input, layer)

输入和输出是有数据的结构，层结构有该层特有的参数。本层有以下字段。

- pad - 要对输入层进行的填充
- stride - 该层的跨度
- k - 核的大小（假设是方形核）。

Q 1.3 卷积层 - 1分

用以下定义实现一个卷积层。

[output] = conv_layer_forward(input, layer, param)

卷积层的层具有与集合层相同的字段，参数具有与该层对应的权重。不要担心一个字段 "group"，它在这个赋值中被设置为1。卷积层有一个字段 "num"，它是内核的数量，等于输出通道的数量。

问题1.4 ReLU - 1分

用以下定义实现ReLU函数。

[输出] = relu_forward(输入)

第2部分 背面传播

在实现前向传播后，我们将使用链式规则实现反向传播。让我们假设第i层计算了一个参数为w_i的函数f_i，那么最终的损失可以写成以下形式。

$$l = f_i(w_i, f_{i-1}(w_{i-1}, \dots))$$

为了更新参数，我们需要计算损失的梯度，即每个参数的梯度。

$$\frac{\partial l}{\partial w_i} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial w_i}$$

$$\frac{\partial l}{\partial h_{i-1}} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial h_{i-1}}$$

其中，$h_i = f_i(w_i, h_{i-1})$。

每一层的反向传播函数以输入、输出、层、param为输入，并返回param_grad和input_od。output.diff存储$\frac{\partial l}{\partial h_i}$。你要用它来计算$\frac{\partial l}{\partial w}$并存储在param_grad.w中，$\frac{\partial l}{\partial b}$，并存储在param_grad.b中。你要也希望在input_od中返回$\frac{\partial l}{\partial h_{i-1}}$，这是损失的梯度，与输入层有关。

问题2.1 ReLU - 1分

在relu_backward.m文件中实现Relu层的后向传递。这个层没有任何参数，所以你不需要返回param_grad结构。

问题2.2 内部产品层 - 1分

在inner_product_backward.m中为Inner product层实现后向传递。

组建网络

这一部分已经为你完成了，在函数convnet forward中可用。这个函数接收参数、层和输入数据，并在网络的每一层生成输出。它还返回图像属于每个类别的概率。我们鼓励你研究这个函数的代码，以了解数据是如何被传递来执行前向传递的。

第三部分 培训

函数conv_net将前向和后向通道放在一起并训练网络。这个函数也已经实现了。

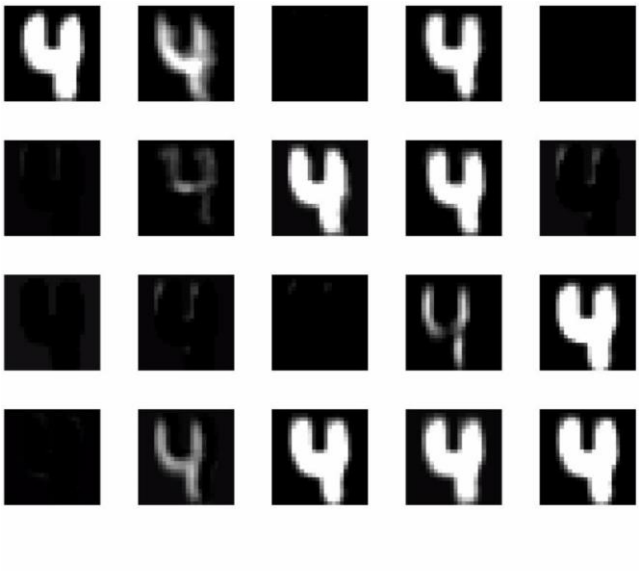


Figure 4: Feature maps of the second layer

问题3.1 培训 - 1分

脚本train_lenet.m定义了优化参数并对网络进行实际更新。这个脚本加载了一个预训练的网络，并对网络进行了3000次迭代训练。再训练3000次后，在你的文章中报告获得的测试精度。将改进后的网络权重保存为lenet.mat，其格式与lenet_pretrained.mat相同。准确率应该在95%以上。

问题3.2 测试网络 - 1分

脚本test_network.m已经提供，它通过网络运行测试数据并获得预测概率。修改这个脚本以生成混淆矩阵，并对前两个混淆的类对进行评论（为什么它们会混淆等等）。

问题3.3 真实世界的测试 - 1分

获得真实世界的数字例子。在至少5个例子上展示你的系统的结果，这些例子是你自己获得的（例如，从互联网上下载，自己涂鸦，自己拍摄图像。这里不要使用第5部分的样本）。对于这一步，请手动裁剪一个包含每个数字的边界框，而不是像第5部分那样，要求你自动找到一个数字。

第4部分 视觉化

问题4.1 - 1分

写一个脚本vis_data.m，它可以从数据中加载一个样本图像，将第二和第三层（即CONV层和ReLU层）的输出可视化。在一个图文件中显示每层的20张图像（使用子图，并以4×5的格式组织它们--如图4）。澄清一下，你取一张图像，通过你的网络运行，并在CONV层和ReLU层将该图像的20个特征可视化。

问题4.2 - 1分

将特征图与原始图像进行比较，并解释其中的差异。

第5部分 图像分类 - 2分

我们现在将尝试使用经过充分训练的网络来完成光学字符识别的任务。在图像文件夹中为你提供了一组真实世界的图像。编写一个ec.m脚本，它将读取这些图像并识别手写的数字。

你训练的网络需要一个灰度图像，每个图像中都有一个数字。在给定的真实图像中，有许多方法可以获得这个数字。下面是一个可能的方法的概要。

1. 通过执行简单的操作如阈值化，将每个像素分类为前景或背景像素。
2. 找到连接的组件，在每个字符周围放置一个边界框。你可以使用matlab的一个内置函数来做这个。
3. 拿出每个边界框，如果有必要，将其垫高并调整为28×28的大小，然后通过网络传递。

识别中可能有错误，在报告中报告你的网络的输出。对于这一部分，你可以使用*graythresh*、*adaptthresh*、*bwconncomp*、*bwlabel*和*regionprops*等内置函数。

附录。项目中所有文件的清单

- col2im_conv.m 帮助函数，如果需要，你可以使用它
- col2im_conv_matlab.m 帮助函数，如果需要，你可以使用这个。
- conv_layer_backward.m - 请勿修改

- conv_layer forward.m - 为了实现
- conv_net.m - 请勿修改
- convnet_forward.m - 请勿修改
- get_lenet.m - 请勿修改。拥有架构。
- get_lr.m - 获取每个迭代的学习率。
- im2col_conv.m 帮助函数，如果需要，你可以使用它
- im2col_conv_batch.m 帮助函数，如果需要，你可以使用它
- init_convnet.m 初始化网络权重
- inner_product_backward.m - 用于实现
- inner_product_forward.m - 为了实现
- load_mnist.m - 加载训练数据。
- mlrloss.m - 执行损失层。
- pooling_layer_backward.m 已实现，请勿修改
- pooling_layer_forward.m - 为了实现
- relu_backward.m - 为了实现
- relu_forward.m - 为了实现
- sgd_momentum.m - 请勿修改。有更新的方程
- test_network.m - 测试脚本
- train_lenet.m - Train script
- vis_data.m - 添加代码以实现过滤器的可视化。
- lenet_pretrained.mat - 训练过的权重
- mnist_all.mat - 数据集

笔记

这里有一些你在实施时应该牢记的要点。

- 上面的所有方程式都描述了各层在单个数据点上的运作。你的实现将不得不在一小部分输入上工作，称为 "批次"。
- 始终确保每层的输出.数据已经被重塑为一个二维矩阵。