智能计算系统实验

实验指导书

安徽工业大学计算机科学与技术学院

实验　一	基于三层神经网络实现手写数字分类
实验　二	基于 VGG 实现图像分类
专业班级	智能 202
学　　号	209074458
姓　　名	张建才
指导教师	李静

实验 1：基于三层神经网络实现手写数字分类

实验目的

1) 神经网络的模块化实现；
2) Python 中 numpy 库的使用；
3) 理解神经网络的训练过程；

背景知识

1) 概述：本实验中的三层神经网络由三个全连接层构成，在每两个全连接层之间会插入 ReLU 激活函数引入非线性变换，最后使用 Softmax 层计算交叉熵损失。
2) 基本单元：全连接层；ReLU 激活函数；Softmax 损失函数；
3) 神经网络训练过程：前向计算；反向计算；
4) Numpy 中关于矩阵函数举例：numpy.dot 方法可以支持矩阵相乘；可以使用 numpy.zeros 方法确保维度相同；numpy.shape 可以读取矩阵的长度等其他方法。
5) 更新参数：初始化，前向计算时不更新参数；反向计算时根据学习率和参数的梯度更新参数。
6) 计算公式：

   全连接层公式：全连接层的输入为一维向量 X，维度为 M；输出为一维向量 Y，维度为 N；权重 W 是二维 矩阵，维度为 M × N，偏置 B 是一维向量 X，维度为 N。给定神经网络

损失函数 L 对当前全连接层的输出 Y 的偏导

前向传播: $\boldsymbol{y} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$

反向传播: $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \nabla_{\boldsymbol{W}} L$; $\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta \nabla_{\boldsymbol{b}} L$;

$\nabla_{\boldsymbol{b}} L = \nabla_{\boldsymbol{y}} L$; $\nabla_{\boldsymbol{x}} L = \boldsymbol{W}^T \nabla_{\boldsymbol{y}} L$; $\nabla_{\boldsymbol{W}} L = \boldsymbol{x} \nabla_{\boldsymbol{y}} L^T$

计算反向传播时，第二层要计算损失函数对（前向传播）第一层输出的偏导，$\nabla_{\boldsymbol{h}} L = \boldsymbol{W}^{(2)^T} \nabla_{\boldsymbol{z}} L$；h 为前向传播第一层的输出；z 为第二层的输出；可以看到这是对上面公式的运用。

ReLU 激活函数层: $\boldsymbol{y}(i) = \max(0, \boldsymbol{x}(i))$

损失层函数: $L = -\sum_i \boldsymbol{y}(i) \ln \hat{\boldsymbol{y}}(i)$; $\nabla_{\boldsymbol{x}} L = \frac{\partial L}{\partial \boldsymbol{x}} = \hat{\boldsymbol{y}} - \boldsymbol{y}$ （此时的 x 是损失层输入的量全连接层经过激活层后的输出 y）

实验环境

使用运算硬件：CPU

软件环境：python3.11；numpy 库；本实验不需要 TensorFlow

实验步骤

1） MNIST 数据集函数 load_mnist()和 load_data()

2） 全连接层类：初始化；前向传播计算；反向传播计算；参数更新；参数加载。

3） ReLU 激活函数类：前向传播计算；反向传播计算。

4） Softmax 损失函数类：计算损失；前向传播计算；反向传播

计算。

5） 三层神经网络类。

6） 主要代码:

```
def backward(self, top_diff):  # 反向传播的计算
    # TODO: 全连接层的反向传播，计算参数梯度和本层损失
    self.d_weight = np.dot(self.input.T, top_diff)
    self.batch_size = top_diff.shape[0]
    self.d_bias =np.dot(np.ones(shape=(1,self.batch_size)),top_diff)
    bottom_diff = np.dot(top_diff, self.weight.T)
    return bottom_diff
def update_param(self, lr):  # 参数更新
    # TODO: 对全连接层参数利用参数进行更新
    self.weight = self.weight - lr * self.d_weight
    self.bias = self.bias - lr * self.d_bias
def backward(self, top_diff):  # 反向传播的计算
    # TODO: ReLU 层的反向传播，计算本层损失
    bottom_diff = np.zeros_like(self.input)
    mask = self.input >= 0
    np.putmask(bottom_diff, mask, top_diff)
    return bottom_diff
def backward(self):  # 反向传播的计算
    # TODO: softmax 损失层的反向传播，计算本层损失
    bottom_diff = (self.prob - self.label_onehot) / self.batch_size
    return bottom_diff
```

实验思考

如果在写代码之前没有分清楚各个模块所包含的功能，在牵扯到过多模块之后；模块之间的关系会混乱。所以，理清楚大的三个模块：各个函数基本的代码；主函数；调用各个功能函数使其成为一体的过渡模块。然后再在各个函数基本代码中进行小的类的分类：（神经网络的三个主层）全连接层，损失函数，激活函数。

初始化网络，只有在反向传播时，初始化的参数才会更新。而且，参数更新的公式与损失函数有关。

实验二：基于 VGG 实现图像分类

实验目的

1） 实现卷积层，最大池化层。

2） 理解卷积核，池化层实现的数学原理。

3） 基于 python 实现。

背景介绍

1） 卷积层中的卷积运算：对输入子矩阵和卷积核做矩阵内积。

2） 前向传播计算时，为了保证卷积之后的有效输出尺寸与输入尺寸一致，首先对卷积层的输入 X 做边界扩充。

3） $$\boldsymbol{Y}(n, c_{out}, h, w) = \sum_{c_{in}, k_h, k_w} \boldsymbol{W}(c_{in}, k_h, k_w, c_{out}) \boldsymbol{X}_{pad}(n, c_{in}, hs + k_h, ws + k_w) + \boldsymbol{b}(c_{out})$$

$\boldsymbol{Y}$ 的维度为 $N \times C_{out} \times H_{out} \times W_{out}$

C_{out} 为输出特征图的通道数

卷积核张量 $\boldsymbol{W}$ 用四维矩阵表示，维度为 $C_{in} \times K \times K \times C_{out}$

特征图 $\boldsymbol{X}_{pad}$ 为扩充后的特征图。

反向传播与实验一类似。只不过多了一个池化层。如下：

4） 池化层前向传播计算时，输出特征图 Y 中某一位置的值是输入特征图 X 的对应池化窗口内的最大值。

5） 由于最大池化层在前向传播后仅保留池化窗口内的最大值，因此在反向传播时，仅将后一层损失中对应该池化窗口的值传递给池化窗口内最大值所在位置，其他位置值置为 0。

实验环境

硬件环境：CPU

软件环境：python；Numpy；不使用 TensorFlow。

数据集：ImageNet。

实验步骤

1） 整体与实验一相似。有以下新的模块：

卷积层；池化层；flatten 层（扁平化层）；注意：扁平化层属于全连接层。

2） 卷积层：

```
for idxn in range(self.input.shape[0]):
    for idxc in range(self.channel_out):
        for idxh in range(height_out):
            for idxw in range(width_out):
                # TODO: 计算卷积层的前向传播，特征图与卷积核的内积再加偏置
                #(3.3)
                hs = idxh * self.stride
                ws = idxw * self.stride
                self.output[idxn, idxc, idxh, idxw] = np.sum(self.weight[:, :, :, idxc] * \
                self.input_pad[idxn, :, hs:hs+self.kernel_size, ws:ws+self.kernel_size]) + \
                self.bias[idxc]
return self.output
```

3） 池化层：

```
for idxn in range(self.input.shape[0]):
    for idxc in range(self.input.shape[1]):
        for idxh in range(height_out):
            for idxw in range(width_out):
                # TODO:  计算最大池化层的前向传播，  取池化窗口内的最大值
                hs = idxh * self.stride
                ws = idxw * self.stride
                self.output[idxn, idxc, idxh, idxw] = np.max(self.input[idxn, idxc, hs:hs+self.kernel_size, ws:ws+self.kernel_size])
return self.output
```

实验思考

1） 卷积层和池化层构成特征提取器，而扁平层是分类器。

2） 本实验仅需要对给定的一张图像进行分类，因此给定一张 预处理好的图像，执行网络前向传播函数即可获得 VGG19 预测的 1000 个类别的分类概率，然后取其中概率最大的类别作为最终预测的分类类别。