**计算机科学与技术学院神经网络与深度学习课程实验报告**

|  |  |  |
| --- | --- | --- |
| 实验题目：了解卷积神经网络的架构 | | 学号：201900150221 |
| 日期：10.28 | 班级： 19智能 | 姓名： 张进华 |
| Email：zjh15117117428@163.com | | |
| **实验目的：**  了解卷积神经网络的架构网络并通过数据训练这些模型获得实践。实现两个子任务：构建卷积神经网络模型和应用，使用 Keras 构建残差网络（可选） | | |
| **实验软件和硬件环境：**  Visual studio Code python 3.9.7  Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz | | |
| **实验步骤**  **步骤一 Convolutional Neural Networks: Step by Step**  一个卷积层将一个输入转换成一个不同大小的输出体积，首先实现两个辅助函数：一个用于零填充，另一个用于计算卷积函数本身。  **1.1 - Zero-Padding**  填充的主要好处如下：  允许使用 CONV 层而不必缩小体积的高度和宽度。 这对于构建更深的网络很重要，否则当进入更深的层时，高度/宽度会缩小。 一个重要的特殊情况是“相同”卷积，其中高度/宽度在一层之后完全保留。  在图像的边界保留更多信息。 如果没有填充，下一层的值很少会受到作为图像边缘的像素的影响。  零填充的实现代码关键部分如下：      **1.2 - Single step of convolution**  实现一个卷积步骤，将过滤器应用到输入的单个位置，用于构建一个卷积单元，实现代码关键部分如下：    **1.3 - Convolutional Neural Networks - Forward pass**  在前向传递中，采用许多过滤器并将它们与输入进行卷积。 每个“卷积”都会提供一个 2D 矩阵输出，将堆叠这些输出以获得 3D 张量，其中输出部分的矩阵维度计算公式如下：    开始 for loop，4 个 for 循环分别遍历  ①样本，②输出图像的高度，③输出图像的宽度，④每个通道，即当前层卷积核的个数  先定位当前的切片位置，得到 vert\_start 和 vert\_end。  定位好之后，在输入的 a 图像中取出要进行加卷积的部分，并执行单步卷积，保存到结果 Z 中，实现代码关键部分如下：    **1.4 - Pooling layer**  池化（POOL）层减少了输入的高度和宽度，有助于减少计算。  **1.4.1 - Forward Pooling**  由于没有填充，将池化的输出形状绑定到输入形状的公式为：    池化层前向传播过程实现代码如下，有两种模式，最大池化和平均池化，开始 for loop，4 个 for 循环分别遍历①样本，②输出图像的高度，③输出图像的宽度，④每个通道，即当前层卷积核的个数，先定位当前的切片位置，得到 vert\_start 和 vert\_end。定位好之后，在输入的 a 图像中取出要进行池化操作的部分，并执行池化操作，保存到结果 Z中，实现代码关键部分如下：    **1.5 - Backpropagation in convolutional neural networks**  了解卷积网络中的反向传播  **1.5.1 - Convolutional layer backward pass**  实现 CONV 层的反向传递  **1.5.1.1 - Computing dA:**  相对于特定过滤器 W\_c 和给定训练示例的成本计算 dA 的公式：    **1.5.1.2 - Computing dW:**  关于损失计算 dW\_c（dW\_c 是一个滤波器的导数）的公式：    a\_slice对应于用于生成激活 Z\_{ij}的切片。因此，提供了 W 相对于该切片的梯度。由于是相同的 W，将所有这些梯度相加得到 dW。  **1.5.1.3 - Computing db:**  计算 db 相对于某个过滤器 W\_c的成本的公式：    根据公式实现代码如下：      **1.5.2 Pooling layer - backward pass**  实现池化层的反向传递，从 MAX-POOL 层开始。 虽然池化层没有要更新的反向传播参数，但需要通过池化层反向传播梯度，以便计算池化层之前的层的梯度。  **1.5.2.1 Max pooling - backward pass**  在进入池化层的反向传播之前，构建一个名为 create\_mask\_from\_window()的辅助函数，执行以下操作：创建了一个“掩码”矩阵，用于跟踪矩阵的最大值在哪里。 True （1）表示最大值在 X 中的位置，其他条目为 False (0)。 平均池化的反向传递将与此类似，但使用不同的掩码，实现代码关键部分如下：    为什么要跟踪最大值的位置？ 这是因为这是最终影响输出和成本的输入值。 反向传播是根据成本计算梯度，所以任何影响最终成本的东西都应该有一个非零梯度。 因此，反向传播会将梯度“传播”会影响成本的特定输入值。  **1.5.2.2 - Average pooling - backward pass**  在最大池化中，对于每个输入窗口，对输出的所有“影响”都来自单个输入值——最大值。 在平均池化中，输入窗口的每个元素对输出都有相同的影响。 因此，要实现反向传播，实现一个反映这一点的辅助函数，关键代码如下：    **1.5.2.3 Putting it together: Pooling backward**  在两种模式（max和average）下实现 pool\_backward（） 函数。 使用 4 个 for 循环（迭代训练示例、高度、宽度和通道）。如果它等于 average，使用上面实现的distribute\_value()函数来创建一个与 a\_slice形状相同的矩阵。 否则，模式等于max，使用create\_mask\_from\_window()创建一个掩码并将其乘以dZ的相应值，实现代码关键部分如下：    **步骤2 Convolutional Neural Networks: Application**  **2.1 - Create placeholders**  TensorFlow 要求在运行会话时输入模型的输入数据创建占位符，实现代码如下：    **2.2 - Initialize parameters**  初始化权重W1和W2，使用tf.get\_variable(name, shape=None, initializer=None)来定义变量时，可以利用变量初始化函数来实现对 initializer 的赋值。在神经网络中，最常权重赋值方式是**正态随机赋值**和 **Xavier赋值。**  常用变量初始化函数如下  **正态**tf.random\_normal\_initializer(shape,mean=0.0,stddev=1.0,dtype=tf.float32,name=None))  **均匀**tf.random\_uniform\_initializer(shape,minval=0,maxval=None,dtype=tf.float32)  **常量**tf.constant\_initializer(obj,shape)  **全0** tf.zeros\_initializer(shape,dtype)  **全1** tf.ones\_initializer(shape,dtype)  **Xavier初始化器**  如果深度学习的权重初始化的太小，那么信号将在每一层传递时逐渐缩小而难以产生作用，但如果初始化得太大，那信号将在每层传递时逐渐放大并导致发散和失效，Xavier初始化器就是让权重不大不小，刚好合适。会根据某一层网络的输入、输出节点的数量自动调整最合适的分布。  tf.contrib.layers.xavier\_initializer( uniform=True, seed=None, dtype=tf.float32)  uniform参数: 使用uniform或者normal分布来随机初始化，实现代码如下：      **2.3 - Forward propagation**  由提示实现前向传播函数，关键代码如下：    **2.4 - Compute cost**  计算softmax的损失函数，这个函数既计算softmax的激活，也计算其损失，实现代码如下：    **2.5 Model**  最后，合并在上面实现的辅助函数以构建模型，在 SIGNS 数据集上训练它，实现代码如下：      完成并构建了一个模型，此时效果如下：    可以看到最终在训练集上的准确率是 0.887，测试集上的准确率是 0.767  **步骤三 sidual Networks**  使用残差网络 (ResNets) 构建非常深的卷积网络，理论上，非常深的网络可以表示非常复杂的功能，但在实践中，很难训练。  非常深的网络的主要好处是可以表示非常复杂的功能，还可以学习许多不同抽象级别的特征，从边缘（在较低层）到非常复杂的特征（在更深层）。然而，使用更深的网络并不总是有帮助，训练的一个巨大障碍是梯度消失。  非常深的网络通常有一个梯度信号，会很快变为零，从而使梯度下降慢得难以忍受。更具体地说，在梯度下降过程中，当你从最后一层反向传播到第一层时，你在每一步都乘以权重矩阵，因此梯度可以呈指数快速下降到零（或者，在极少数情况下，呈指数增长快速并“爆炸”以获取非常大的值）。  因此，在训练期间，可能会看到，随着训练的进行，较早层的梯度幅度（或范数）会非常迅速地减小为零，而在 ResNets 中，“shortcut”或“skip connection”允许梯度直接反向传播到较早的层  **3.1 - The identity block**  The identity block是 ResNets 中使用的标准块，对应于输入激活（比如 a^[l]）与输出激活（比如 a^[l+2 ]）。 为了充实 ResNet 的身份块中发生的不同步骤，实现代码如下：  第一层实现代码如下：    第二层实现代码如下：    第三层实现代码如下：    最后相加并同Relu,实现代码如下：      **3.2 - The convolutional block**  已经实现了 ResNet 身份块，接下来，ResNet“卷积块”是另一种类型的块。 当输入和输出维度不匹配时，可以使用这种类型的块，与identity block的区别在于快捷路径中有一个CONV2D层。  第一层实现代码如下：    第二层实现代码如下：    第三层实现代码如下：    SHORTCUT PATH实现代码如下：    最后一步相加并通过Relu层，实现代码如下：      **3.3 - Building your first ResNet model (50 layers)**  构建卷积神经网络。现在已经构建好需要用到的残差块了，现在根据这个 50 层的神经网络原理图构建卷积神经网络。第一步实现代码如下：    第二步实现代码如下：    第三部实现代码如下：    第四步实现代码如下：    第五步实现代码如下：    均值池化部分代码如下：    **3.4 构建模型、训练、测试**  建立模型并导入数据集，得到手势数据集维度如下：    训练模型过程如下：    最后测试效果如下：    然后使用训练好的模型进行测试，实现效果如下：    可以发现使用别人训练很长时间的权重的模型效果比刚才自己只两次的模型效果好很多，在测试集上的准确率达到了 0.867 | | |
| **结论分析与体会：**  **Convolutional Neural Networks**  学习了卷积神经网络的构建方法，了解了卷积神经网络主要有卷积模块和池化模块组成。  卷积模块，包含了以下函数：  使用 0 扩充边界、卷积窗口、前向卷积、反向卷积（可选）  池化模块，包含了以下函数：  前向池化、创建掩码、值分配、反向池化（可选）  有了这两个模块后，根据网络结构原理图，构建了 CNN 神经网络。  **CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN -> FULLYCONNECTED**  构建了 CNN 模型，包括以下步骤  创建占位符、初始化参数、前向传播、计算成本、反向传播、创建优化器、创建一个 session 运行模型  Residual Networks  非常深的“普通”网络在实践中不起作用，因为它们由于梯度消失而难以训练，skip-connections 有助于解决渐变消失问题，还使 ResNet 块很容易学习恒等映射函数。 | | |
| **就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟1－3道问答题：**  1.tensorflow 需要用到对应版本才能够很好的使用，对于 python 环境，使用 anconda 可能会更为方便快捷，  建立专门的虚拟环境，运行会更加方便一些，直接改自己的配置，又没法做其他课的实验，还得修改回来 | | |