卷积神经网络的复现

作者：王基桥(学号：2019110370)

本系统是基于卷积神经网络(CNN)来实现的对MNIST手写数字进行识别的系统。训练集包含42000张28×28像素的从0到9的黑白手写数字图片。测试集包含28000张28×28像素的从0到9的黑白手写数字图片。通过使用CNN来训练训练集的数据，最终完成对28000张图片的识别。识别准确率：训练集准确率96.1114%，测试集准确率94.442%。

主要工作：复现了一个两层卷积层和三层全连接层的卷积神经网络。

该系统主要分三个部分：系统主函数、系统各部分函数、系统测试函数。

**一、系统主函数文件(digit\_recognize\_cnn.py)：**用于加载训练集和测试集以及初始化参数。

1.训练集：X\_train、Y\_train。

2.测试集：X\_test。

对于X\_train和X\_test二者均除以255以将其取值范围缩小到0到1之间。

对于Y\_train，需要将其转化为one-hot向量。

3.参数：cv\_ft1:第一层卷积层的卷积核(convolution filter)；

cv\_s1:第一层卷积层卷积核移动的步长(convolution stride)；

ac\_fnc\_cv:第一层卷积层卷积完毕后的激活方程(activation function for convolution；

po\_sz1:第一层卷积层之后进行max pool或mean pool时的pooling size；

po\_str1:第一层pooling层采样核移动的步长(pooling stride)；

po\_wy:在进行pooling时的采样方式(pool way)：max pool或mean pool

cv\_ft2:第二层卷积层的卷积核(convolution filter)；

cv\_s2:第二层卷积层移动的步长(convolution stride)；

po\_sz2:第二层卷积层之后进行max pool或mean pool时的pooling size；

po\_str2: 第二层pooling层采样核移动的步长(pooling stride)；

ac\_fnc\_fl:全连接层的激活方程(activation function for fully connected layer；

fl\_w1,fl\_b1,fl\_w2,fl\_b2,fl\_w3,fl\_b3,fl\_w4:全连接层的权重和偏置。由于最后一层使用的是soft max方程，因此最后一层全连接层没有偏置fl\_b4。

4.初始化方法：有两个。第一个：在没有预训练数据时，采用随机初始化的方法初始化各个参数。第二个：在有已训练的数据的时候，加载预训练好的参数来初始化各个参数。

5.然后将各个参数存入python dictionary类型的变量parameter中。

**二、系统各部分函数综合文件(cnn\_functions.py)：**

1.load\_data():用于加载X\_train,Y\_train

2.padding():用在卷积层之前进行zero padding来保持卷积前后的大小。Zero padding:在矩阵的外围添加n圈零。n的大小依据卷积的步长来决定。

3.convolution\_layer\_tf():调用tensorflow内置的tf.nn.conv2d方程来计算卷积。通过此方式来调用GPU内核来加速运算。

4.convolution\_layer\_for\_loop\_version():不使用GPU运算，没有调用tensorflow内置tf.nn.conv2d方程，使用for循环来对每一张图片进行卷积操作。

卷积操作：先将卷积核反转180°然后再乘上矩阵的每一个子矩阵，子矩阵大小和卷积核的大小相同。乘操作是element-wise的乘积，即对应项相乘。跟随卷积步长，自矩阵的左上角开始，子矩阵逐渐往右移动，一行结束后，再往下一行移动。

5.pooling():pooling有两种方式：max pool和mean pool。

max pool:对矩阵的每一个子矩阵（子矩阵大小等于采样核的大小），找出子矩阵中最大的那个值最为采样结果，同时记录下最大值的位置，以方便在反向传播时max pool部分的反向传播计算。跟随卷积步长，自矩阵的左上角开始，子矩阵逐渐向右移动，一行结束后，再往下一行移动。

mean pool:对矩阵的每一个子矩阵（子矩阵大小等于采样核的大小），计算出子矩阵所有项的均值，mean pool不需要记录位置信息。然后跟随卷积步长，子矩阵的左上角开始，子矩阵逐渐向右移动，一行结束后，再往下一行移动。

6.activate():激活方程。激活方程有两个可选的选项：tanh和ReLU。一般都使用ReLU方程，如需添加其他激活方程，可以在本方程中以及之后的activation\_derivative()方程中将该方程的激活方程式和导数方程记录到其中。由求导的链式法则，在梯度反向传播时，只需要更改其中某一步的梯度即可，因此该方法无误。

7.flatten():该方程用于将每一张图片扁平化。

8.soft\_max():该方程用于计算soft max函数。

9.forward\_propagation():该方程用于综合前向传播的各个函数，组成一个总体的框架。

本卷积神经网络的框架为：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 层数 | 本层信息： | 输入大小 | 输出大小 | 计算公式(“\*”代表卷积，“·”代表点积，m为数据集的总大小，当使用mini batch的方法时，m为mini batch的大小) |
| 1 | 步长为1的  3×3卷积层(含zero padding) | (m，28，28，1） | (m，28，28，1） |  |
| 2 | ReLU激活方程 | (m，28，28，1） | (m，28，28，1) |  |
| 3 | 步长为2的  2×2 池化层 | (m，28，28，1） | (m，14，14，1） | 及索引矩阵。max pool的索引矩阵为最大值的位置，mean pool的索引矩阵值均为1 |
| 4 | 步长为1的  3×3卷积层(含zero padding) | (m，14，14，1） | (m，14，14，1) |  |
| 5 | ReLU激活方程 | (m，14，14，1） | (m，14，14，1） |  |
| 6 | 步长为2的  2×2 池化层 | (m，14，14，1） | (m，7，7，1) | 及索引矩阵。max pool的索引矩阵为最大值的位置，mean pool的索引矩阵值均为1 |
| 7 | Flatten层 | (m，7，7，1） | (m，49） | 即，将其扁平化 |
| 8 | 全连接一层 | (m，49) | (m，64) |  |
| 9 | 全连接二层 | (m，64） | (m，32） |  |
| 10 | 全连接三层 | (m，32) | (m，16) |  |
| 11 | soft max层 | (m，16) | (m，10) |  |

10.compute\_cost():该方程用于计算loss。

计算方法： y\_train是一个10维one-hot向量，hypothesis是一个由softmax函数计算出来的10维向量，通过将二者对应项相乘（element-wise multiply）然后将所有项相加，再除以训练数据的个数，得到最终的loss计算公式。(本方法为简化的loss计算方法，完整的方法为：通过交叉熵cost function来计算loss，方程为：，由于在实际运算反向传播时，不需要通过loss来更新参数，loss只是作为一个观测值，在本程序中只需要观测到loss不断下降即可，因此，在本程序计算loss时没有用交叉熵来计算)

11.activation\_derivative():该方程用于计算激活函数的导数。

12.pool\_back():该方程用于计算反向传播时pooling层的反向传播。

假设pooling层反向传播时后一层的梯度为δ：

1. 若使用max pooling，则将δ每一个元素变为一个2×2的小矩阵，然后再将δ与索引矩阵对应项相乘，得到反传结果。
2. 若使用mean pooling，则将δ每一个元素变为一个2×2的小矩阵即可。

13.cv\_back():该方程用于计算卷积层的反向传播。

假设卷积层反向传播时后一层的梯度为δ，设原卷积核为w，卷积层前向传播输入为a，并设卷积层反向传播得到的结果为da和dw，其中da为之后前一层反向传播时需要使用的梯度，dw为卷积核的梯度。则：

14.pad\_back():该方程用于zero padding的反向传播。在卷积层反向传播回来之后，需要保持前后矩阵大小一致，因为在前向传播时，在卷积层之前有zero padding来保持卷积前后矩阵大小一致，因此在反向传播时，由于卷积会改变矩阵的大小，因此也要进行zero padding的反向传播。

15.random\_mini\_batches():该函数用于对数据集随机取样并按照mini batch size的大小来将原数据集切分为若干个mini batch。由于个人电脑的显存不够大，因此，mini\_batch的方法可以有效解决显存太小的问题。

16.backward\_propagation():该函数为反向传播函数。包括全连接层的反向传播以及卷积层和池化层的反向传播。

公式推导：

1. soft max层的梯度公式推导：

设，

首先我们有soft max函数：

并有交叉熵代价函数：

然后我们求该soft max函数对X的导数

(1)当i=j时：

(2)当i≠j时：

综上所述：

因此，

然后我们来推导soft max层的反向传播公式：

反向传播时，是使用loss对x求导，即：

因此，

1. 全连接层的梯度公式推导：

对于全连接第l层，其前向传播公式为：

设全连接关于的梯度为δ，则，关于的梯度为：

换成矩阵的形式即：

本层关于的梯度为：

换成矩阵的形式即：

本层关于的梯度为：

换成矩阵的形式即：

全连接各层的梯度均可由此来计算。

1. 池化层的反向传播：

由于在前向传播时，池化层的输出是输入的下采样结果，因此在反向传播时，梯度反向传播的结果就是输入的梯度上采样的结果。即：

对于最大值池化，最大值放回原来位置，对于均值池化，每个元素变为2×2的小矩阵

1. 卷积层的反向传播：

设卷积层为第l层，则前向传播时的计算公式为：

设关于的梯度为δ，则在反向传播时，关于的梯度为：

关键在于的求法。

举一个例子：假设前向传播时，有：

即：

于是我们对求导可得：

由此可知：关于的导数为进行zero padding之后和翻转180°的卷积核进行卷积的结果，即：

推广到N维矩阵的形式，即：

我们对求导可得：

由此可知，关于的导数为和翻转180°的原输入矩阵进行卷积的结果，即：

推广到N维矩阵的形式，即：

综上，我们得到了卷积层的梯度下降的两个公式：

之后，我们可以根据刚刚已经得到的各层：soft max层、全连接层、池化层和卷积层的梯度下降公式来进行反向传播。根据求导的链式法则，之后的各层的偏导只需要乘上前一层的偏导即可。

从而我们得到如下表格，并可根据此表格编写反向传播的函数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 层数 | 本层信息 | 输入大小 | 输出大小 | 计算公式(“\*”代表卷积，“·”代表点积，m为数据集的总大小，当使用mini batch的方法时，m为mini batch的大小) |
| 11 | soft max层 | (m,10) | (m,16) |  |
| 10 | 全连接三层 | (m,16) | (m,32) |  |
| 9 | 全连接二层 | (m,32) | (m,64) |  |
| 8 | 全连接一层 | (m,64) | (m,49) |  |
| 7 | Flatten层 | (m,49) | (m,7,7,1) |  |
| 6 | 步长为2的  2×2 池化层 | (m,7,7,1) | (m,14,14,1) | upsample即上采样，将dz2\_2变回原来的大小，同时，对于最大值池化，最大值放回原来位置，对于均值池化，每个元素变为2×2的小矩阵 |
| 5 | ReLU激活方程 | (m,14,14,1) | (m,14,14,1) |  |
| 4 | 步长为1的  3×3卷积层(含zero padding) | (m,14,14,1) | (m,14,14,1) | 注：本程序中为了加速运算，在反向传播阶段直接调用了tensorflow的tf.nn.conv2d函数来进行卷积运算，其中可以设置padding=`same`因此没有使用第三个式子。 |
| 3 | 步长为2的  2×2 池化层 | (m,14,14,1) | (m,28,28,1) | upsample即上采样，将da2\_1变回原来的大小，同时，对于最大值池化，最大值放回原来位置，对于均值池化，每个元素变为2×2的小矩阵 |
| 2 | ReLU激活方程 | (m,28,28,1) | (m,28,28,1) |  |
| 1 | 步长为1的  3×3卷积层(含zero padding) | (m,28,28,1) | (m,28,28,1) | 注：本程序中为了加速运算，在反向传播阶段直接调用了tensorflow的tf.nn.conv2d函数来进行卷积运算，其中可以设置padding=`same`因此没有使用第三个式子。 |

17.update\_parameter():该函数为梯度反向传播的更新参数的函数。

需要传入backward\_propagation()中计算出来的梯度值，现在的参数值，以及学习率。然后利用如下公式更新参数：

然后将更新后的参数保存到txt文档中，以方便之后的图像预测以及持续的机器学习。

18.compute\_accuracy():该函数为计算准确率的函数。传入对图像的预测值，找出其最大值然后转变为one-hot向量，再减去图像的真值y（one-hot向量形式），如果相同则为零，不相同则为1，然后将所有的值加起来，再除以图像个数，得到准确率的值。

19.model\_cnn():该函数为总函数。用于将如上所有函数综合起来组成一个完整的系统。传入的数据包括：训练数据、训练参数、训练次数、学习率和mini batch的大小，然后调用以上各函数，依照CNN的模型架构写出一个完整的流程，在每次循环中都打印出loss和准确率，在完成所有循环后，画出epoch-accuracy的曲线图和epoch-loss的曲线图。最后将测试集的预测结果保存到predictions.csv文件中。

注：本程序训练的数据集来自kaggle平台，测试集也来自kaggle平台，如需得到测试集的准确率就需要按照kaggle平台要求的格式将预测结果保存到.csv格式的文件中，因此最后一步是为了得到测试集准确率用的。

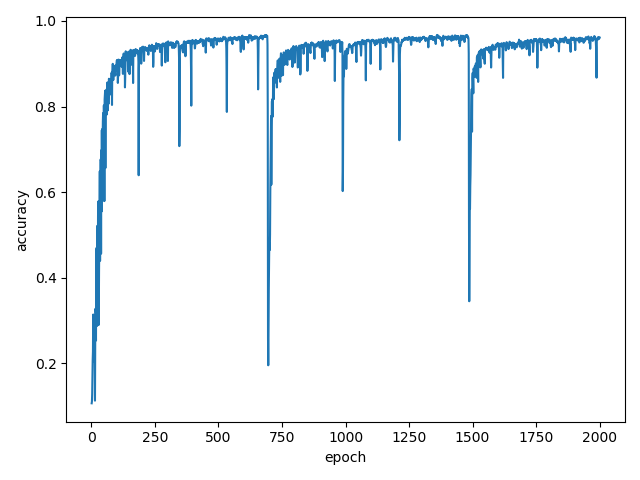
**三、系统测试函数文件(test.py)：**用于查验每张图片的预测结果。

说明：本程序首先加载保存下来的模型参数，然后对随机抽取的每一张图片进行前向传播，然后得到预测结果，同时将其以图片的形式显示出来，以便于查验。

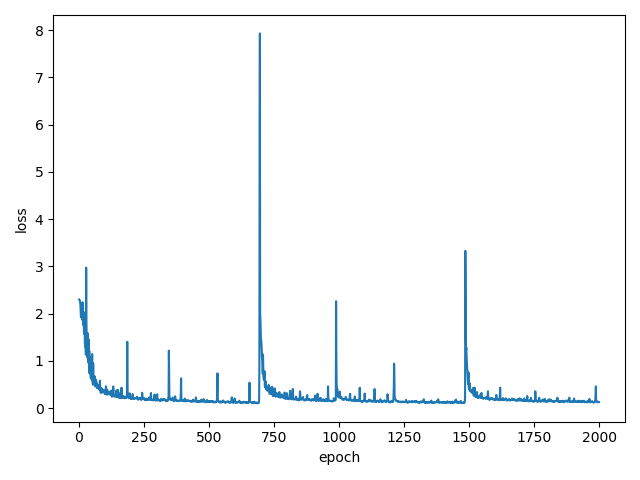
## 实验结果及分析：

**实验结果：**

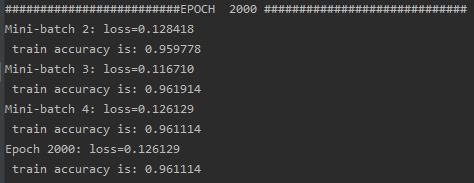
epoch-accuracy曲线图：



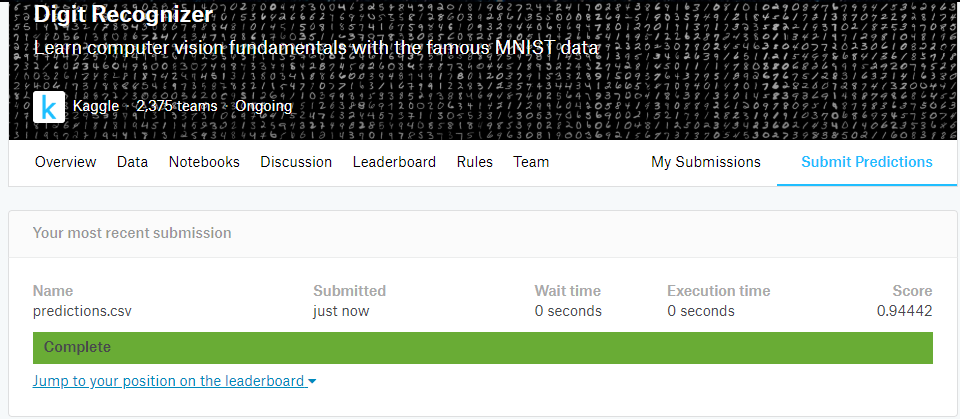
epoch-loss曲线图：



最终得到的损失值和训练集准确率：



最终得到的测试集准确率：



**结果分析：**

本次实验在2080ti显卡上运行了2000epoch，最终得到了训练集准确率96.1114%，测试集准确率94.442%的结果。但由曲线图来看，程序在运行到700epoch和1500epoch时出现了两次大的波折，从实验时观测的数据来看，只需要训练不到700epoch就可以达到训练集准确率96%左右的结果。出现大的波折的原因可能是卷积神经网络的损失函数本身并不是一个凸函数，采用梯度下降的方法很难找到全局最优解，而程序如果从一个局部最优点突然跳到另一个局部最优点的话，就可能出现这样的大的波折。学习率设置太高，参数初始化不太好，都有可能会导致这样的问题。

#### 总结：

本算法只是对卷积神经网络算法的一个复现，主要目的是学习卷积神经网络内部的架构和核心的反向传播算法。从实验结果来看，本次复现还算比较成功，但是仍然出现了训练过程中出现大的波折的问题，一方面，可能是参数初始化以及学习率设置不够恰到好处，另一方面，可能就是我对卷积神经网络的了解依然不够深刻，所以还需要继续努力。

## 参考资料：

1. <https://zhuanlan.zhihu.com/p/37740860>
2. <https://blog.csdn.net/Hearthougan/article/details/72910223>
3. <https://www.deeplearning.ai/>