神经网络学习实现手写数字识别

前言

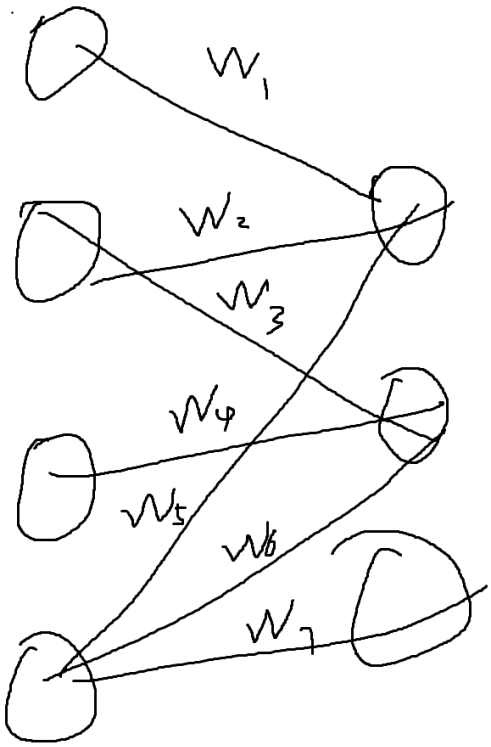
因为想做点相对有意思的东西，所以选择了自己感兴趣的AI方向的内容。整个项目手动实现了所有的所需函数与所需类别，由于个人水平的有限，过程中出过各种奇奇怪怪的bug，然后tensor等结构的实现效率应该是不如前人的。但最后依然取得相对较优的结果，也算是能交上一份合格的答卷。

算法背景

- 神经网络学习是基于生物学中神经系统而创造的。科学家认为可以通过仿制生物体的神经网络来制造人工神经网络系统，但是过去由于算力的不充足，这些都难以实现。伴随现代科学技术的发展，现代家用计算机都有了当年银河一代超级计算机级别的计算力，人工神经网络的思想得以应用，人们发现在很多问题上都取得了优异表现。如今，该算法已广泛应用于各种已知的问题中。

算法思想

如何简单地来描述神经网络的思想？我想我可以用下面这张图作为最好的例子



我们可以认为左右两边是不同层的神经元节点，现在我们从左边向右边传递信号。

左边向右边传递的参数即是我们图中的w，这也叫做权重系数。

当我们传到最后的输出层，我们可以得到一次模拟的结果，当模拟的结果与真实的结果存在差异的时候，我们就希望能修改这些权重系数的值，使下一次的结果更加接近于真实的结果。为了达到这个目的，我们就需要知道应该向什么方向修改这些权重系数。

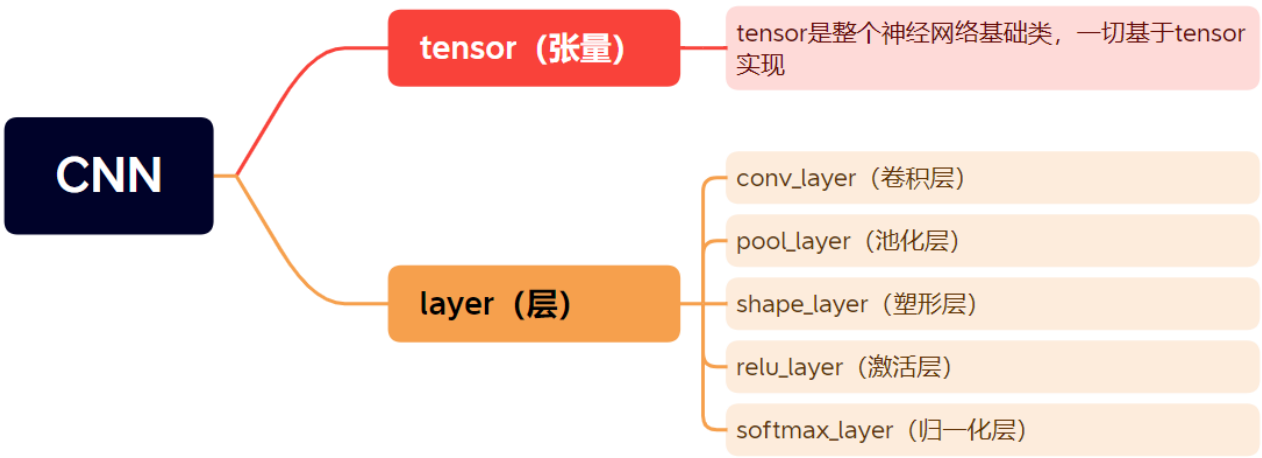
我们这里利用微积分上的偏导概念来实现我们的目的，只要我们知道最终结果对于该权重系数求偏导的值，利于微积分的知识我们知道，只要在对应方向上施加微小的扰动，一定会让最终结果偏向需要的结果，这个过程我们就可以总结为梯度下降。

如何快速地计算偏导就成了新的问题，由微积分中所学习的链式法则，我们可以快速从一个偏导求导另外一个偏导，如果我们能选择一个修改函数，使他能快速求出他对结果的偏导，我们就能借助其一步步反推其他的偏导，这个过程我们就叫做反向传播。

如此一来，我们所有的问题就得到了解决，我们正向传播求出差异，再反向传播求出偏导，利用偏导修正我们的权重系数，我们的训练目的就达到了。

代码实现

代码实现会比简单地叙述思想要相对困难一点



这是代码整体框架的思维导图，下面我们将基于此来讲述我们的实现。

tensor

tensor作为我们的基础变量，基于我们的需求实现了很多操作，包括最基础的矩阵乘法，矩阵转置，矩阵点加。以及为了方便操作实现的各种引用。出于泛用性考虑，最终将tensor设计为了一个可以装载不同类型数据的张量类型

layer_base

这是层类的基础，在该层中，我们设计了所有层类所必须的内容，例如fix_weight的权重修正函数，input,output作为该层的输入输出，deriv作为该层输入的偏导。activate作为正向传播时使用

的函数，deriv_calc作为计算偏导的函数

conv_layer

实现了卷积核对输入数据特征的提取

pool_layer

实现了数据整体特征的概括，提升了整体的运行效率

shape_layer

将二维的输入数据转化为一维的数据

relu_layer

将小于0的数修正为0

soft_max

将最后的得分归一化，使所有数之和为1，成为一个相对得分，最后的最大值即为预测答案

超参数的设置

经过诸多尝试，最后得到的较为不错的超参数设置如下

```
double rate = 0.01;//学习速率

const double up=0.1,down=0;//权值初始随机范围

const int basic_x=28,basic_y=28,baisc_z=1; //初始矩阵大小

const double cutrate=0;

const int extend_flitter_stride=1;//卷积层步幅
const int extend_flitter_size=3;//卷积核大小
const int extend_flitter_len=3;//卷积核数

const int pool_extend_flitter_stride=2;//池化步幅
const int pool_extend_flitter_size=2;//池化矩阵大小
const int pool_extend_flitter_len=1;//池化层数
```

尾声

虽然CNN算法整体的思维难度并不大，但代码实现起来常常会碰见种种问题。例如参数的设置，例如代码的具体实现细节，但我本人依然从项目中学到了相当多的东西。