构建两层神经网络分类器实验报告

1. 数据集

MINIST 手写数字数据集可以从官方下载，需要科学上网加速下载，或者可以通过第三方快速获取，（好像也可以用 python 提供的库中获取），本次实验从第三方下载到本地并解压来进行实验。

整个数据集是按照二进制大端方式存储的，也就是高位存放在高字节处。一共有四个压缩包文件，解压后分别是 train-images-idx3-ubyte: training set images、train-labels-idx1-ubyte: training set labels、t10k-images-idx3-ubyte: test set images、t10k-labels-idx1-ubyte: test set labels。

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

```
[offset] [type]     [value]     [description]`
`0000   32 bit integer 0x00000801(2049) magic number (MSB first)`
`0004   32 bit integer 60000      number of items`
`0008   unsigned byte  ??        label`
`0009   unsigned byte  ??        label`
`........`
`xxxx   unsigned byte  ??        label
The labels values are 0 to 9.
```

训练集标签数据文件如上，前八个字节表示 magic number 和 numbers of items，后面每个 byte 代表一个真实结果，所以在读取文件时就可以分别读入这些数据进行处理。同理，训练集的图片文件存储格式如下，image 信息通过灰度值存储，前 16 个字节表示这个数据集的一些信息，后面的字节就是图片的信息。每个图片都是 28*28 像素，我们一般展开成一维 784 的列向量。

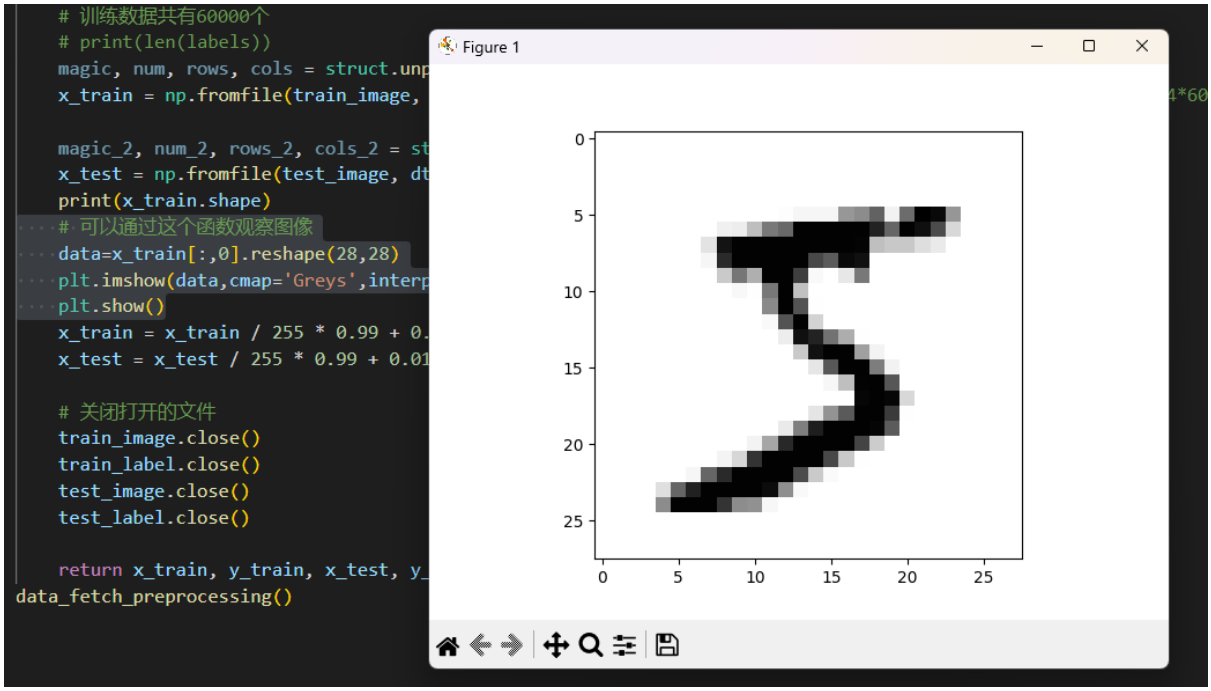
TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

```
[offset] [type]     [value]     [description]`
`0000   32 bit integer 0x00000803(2051) magic number`
`0004   32 bit integer 60000      number of images`
`0008   32 bit integer 28        number of rows`
`0012   32 bit integer 28        number of columns`
`0016   unsigned byte  ??        pixel`
`0017   unsigned byte  ??        pixel`
`........`
`xxxx   unsigned byte  ??        pixel
```

文件解析需要 python 使用 struct 包，具体实现可参考代码实现细节，代码均有注释说明。

读取出来的文件可以通过 matplotlib.pyplot.imshow 来显示二进制图片，同时代码为

了有利于梯度下降的收敛和提高训练速度，把灰度值映射到 0-1 进行归一化处理。



2. 训练神经网络模型

构建两层神经网络，设置输入层为 784 维（28*28 图片拉成一列），中间隐藏层为 200 维，最后输出层为 10 维（分成 0-9 总共 10 个类）。其中输入层和隐藏层的权重矩阵为 w1（200*784），偏差矩阵/向量 b1（200*1），隐藏层和输出层权重矩阵为 w2（10*200），偏差矩阵为 b2（10*1），定义迭代轮次为 5。

前向传播与激活函数：

输入的列向量与权重矩阵乘法运算并加上偏差之后得到一个值就是前向传播，为了保证神经网络非线性，需要使用激活函数，前者的结果通过激活函数之后就是本层的输出。

$$z^{[1]} = W^{[1]} X + b^{[1]}$$

$$a^{[1]} = g(z^{[1]})$$

$$z^{[2]} = a^{[2]} X + b^{[2]}$$

$$a^{[2]} = g(z^{[2]})$$

激活函数通常有 relu、sigmoid、tanh 等，本次实验使用 sigmoid 函数，激活函数表达式为：expit(x) = 1/(1+exp(-x))，这里使用了 scipy.special 库中的 expit，代替 numpy 中的 exp 函数。正向传播和反向传播定义如下，out 保存正向传播的输出，

反向传播用 out 计算。

```
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```

反向传播、loss 以及 L2 正则化、和梯度计算：

反向传播就是链式求导法则，此处不再详细说明，课堂上已经清晰的展示，具体细节可以参考代码和注释。计算 loss 采用的是交叉熵函数，这里采用了 L2 正则化函数来作为损失函数的一部分防止过拟合，所以损失函数按照每一维度求出交叉熵后最后对 60000 个求和取平均得到本次 epoch 的损失，同时本次损失还需要加上两个参数矩阵 w1 和 w2 的二范数的平方和与超参数 lamba 的乘积作为最终的损失误差。求梯度就是对每个列进行求导计算，加上正则化项之后需要每次对参数 w1 和 w2 的修改量 dw 额外进行更新。

学习率下降策略、优化器 SGD

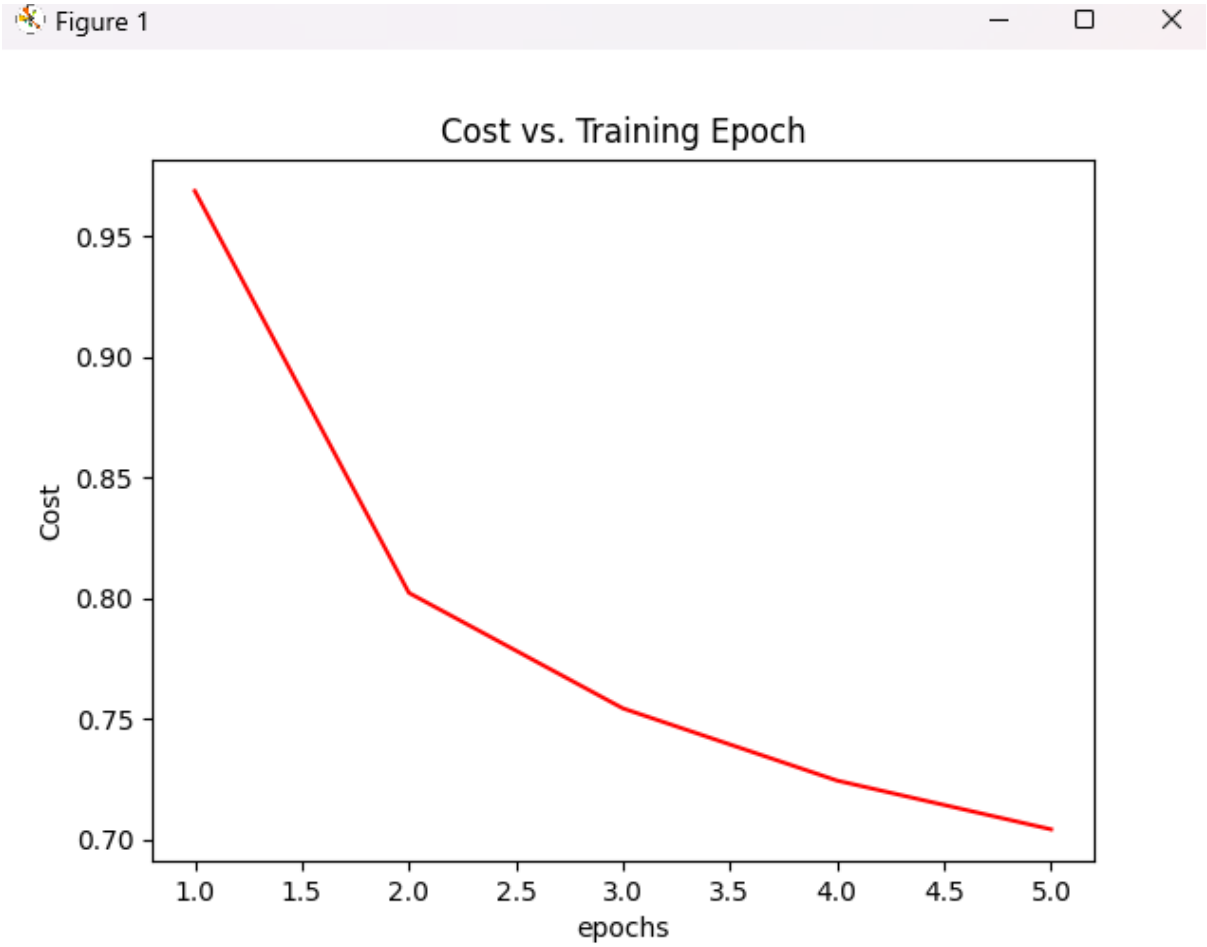
初始化学习率为 0.1，每次 epoch 对学习率乘以 0.9 让学习率下降，即减小步长。通过随机梯度下降算法优化收敛即可，最后保存模型。

3. 测试神经网络模型及结果

训练参数：初始化学习率为 0.2，每次迭代 epoch 降低为原来的 0.9 倍；隐藏层大小为 200 维；正则化强度 lambda 为 0.1（可以修改这个超参数值大小看效果来更新这个超参数）。

通过 60000 个图片样本训练集中训练迭代 5 次得到的模型（w1、w2、b1、b2、lam、learningrate 等参数组成的网络模型）测试测试集中 10000 个样例进行分类输出，与测试集标签进行对比计算准确率分类精度。训练好的模型保存在文件 model_params.txt 中。

可视化结果：训练的 loss 曲线如下，由于测试只跑了一次 10000 个测试集数据，因此只有最终的一个输出结果 loss 和对应的准确性 accuracy，结果如下。



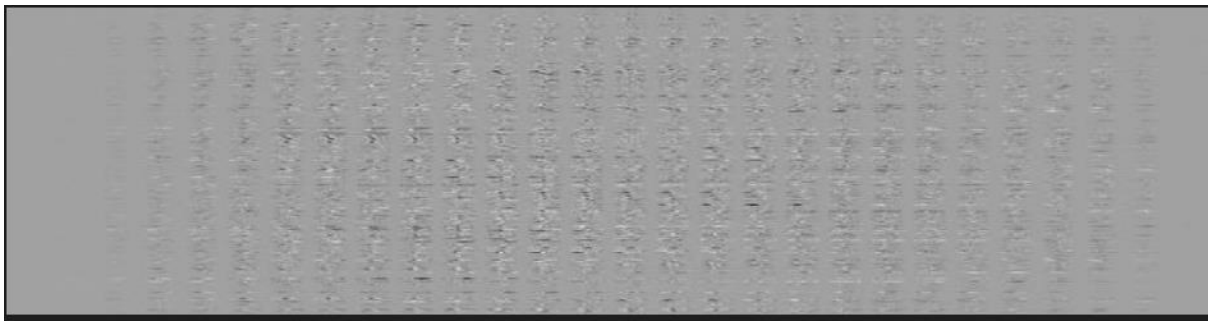

```
[0.9685768433196813, 0.8021341563217997, 0.7542850511599779, 0.7243944889912963, 0.7041923738566711]
start writing...

write success!

准确率: 97%
loss:0.732439
```

可视化每层网络的参数时，注意由于 w1 和 w2 有正有负，直接转化为图像会失真，此处做了转化，把整个矩阵的最大最小值距离归一化，每个矩阵值计算和最小值的差距距离占总距离的比例，用这个比例乘以 255 算出来对应图像上的像素值大小。

w1：



w2：

代码和结果等文件已上传到 github 中，具体可参考链接。
https://github.com/zheng0221/Neural_Network.git
https://github.com/zheng0221/Neural_Network