

**基于全链接网络的手写数字体识别**

**神经网络与深度学习课程实验报告**

班级：105

姓名：耿翊中

学号：2021213382

2023年10月

1. **实验要求**

数据：[MNIST data set](http://yann.lecun.com/exdb/mnist/" \t "https://ucloud.bupt.edu.cn/uclass/course.html" \l "/student/_blank)

• 本题目考察如何设计并实现一个简单的图像分类器。设置本题目的目的如下：

– 理解基本的图像识别流程及数据驱动的方法（训练、预测等阶段）

– 理解训练集/验证集/测试集的数据划分，以及如何使用验证数据调整模型的超参数

– 实现一个**Softmax**分类器

– 实现一个**全连接神经网络**分类器

– 理解不同的分类器之间的区别，以及使用不同的更新方法优化神经网络

**• 附加题：**

– 尝试使用不同的损失函数和正则化方法，观察并分析其对实验结果的影响 (+5 points)

– 尝试使用不同的优化算法，观察并分析其对训练过程和实验结果的影响 (如batch GD, online GD, mini-batch GD, SGD, 或其它的优化算法，如Momentum, Adsgrad, Adam, Admax) (+5 points)

**• 补充：**[MINST](http://yann.lecun.com/exdb/mnist/" \t "https://ucloud.bupt.edu.cn/uclass/course.html" \l "/student/_blank)是一个手写数字数据集，包括了若干手写数字体及其对应的数字，共60000个训练样本，10000个测试样本。每个手写数字被表示为一个28\*28的向量。

1. **代码成果及实现思路**

**2.1源码文件**

Softmax分类器：soft\_main.py

全连接神经网络分类器：Accuracy.py,Main.py,Method.py,Model.py,Optim.py

使用的数据集：mnist-original.mat

来源网站：<https://www.kaggle.com/avnishnish/mnist-original/download>

**2.2数据集导入**

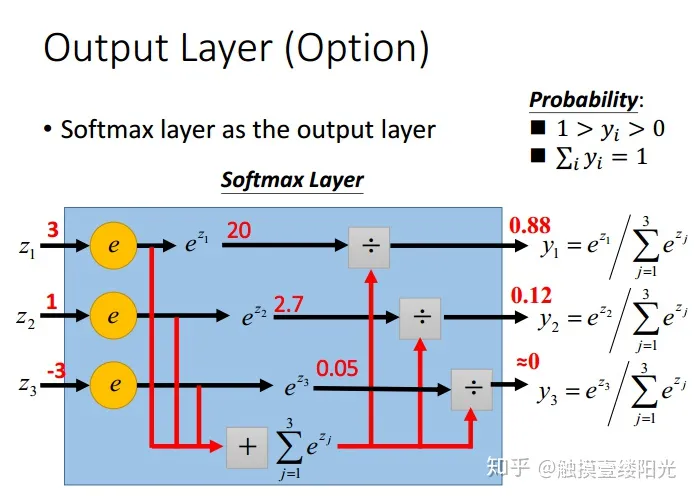
在Softmax和全连接神经网络分类器中都使用了相同的数据集导入方式，数据集来源于网站：<https://www.kaggle.com/avnishnish/mnist-original/download>

，下载到本地后为mnist-original.mat文件，使用**from** scipy.io **import** loadmat 函数将其导入，并划分为60000个训练集和10000个测试集。其中x中每个元素为28\*28的矩阵，代表数字图像像素；y为数字图像像素对应的正确数字标签。

1. **from** scipy.io **import** loadmat
2. **def** loadFile():
3. # 加载数据文件
4. data = loadmat('mnist-original.mat')
5. # 提取数据的特征矩阵，并进行转置，将二维数组转换为一维数组的形式
6. x = data['data']
7. x = x.transpose()
8. # 然后将特征除以255.0，以浮点数的形式重新缩放到[0,1]的范围内，以避免在计算过程中溢出
9. x = x / 255.0
10. # 从数据中提取labels，即x数字像素对应的数字类别y
11. y = data['label']
12. y = y.flatten()
13. # 将数据分割为60,000个训练集
14. x\_train = x[:60000, :]
15. y\_train = y[:60000]
16. # 和10,000个测试集
17. x\_test = x[60000:, :]
18. y\_test = y[60000:]
20. **return** x\_train, y\_train, x\_test, y\_test

**2.3Softmax分类器**

**2.3.1Softmax实现分类原理**



函数角度：softmax函数可以根据输入，输出属于各个类别的可能性大小。

实现方式：使用梯度下降法进行实现，使用交叉熵损失函数作为损失函数。

**2.3.2代码实现解释**

1. 设置超参数及加载数据集
2. # 设置超参数
3. learning\_rate = 0.1
4. num\_epochs = 100
5. batch\_size = 128
6. # 加载数据集
7. x\_train, y\_train, x\_test, y\_test = loadFile()
8. # 训练 softmax 分类器
9. train\_softmax\_classifier(x\_train, y\_train, x\_test, y\_test, learning\_rate, num\_epochs, batch\_size)
10. 定义函数 train\_softmax\_classifier(x\_train, y\_train, x\_test, y\_test, learning\_rate, num\_epochs, batch\_size)；该函数执行Softmax分类器的训练过程。初始化模型参数（权重矩阵 W 和偏置向量 b）。在每个训练周期（epoch）内，使用小批量（mini-batch）梯度下降来更新模型参数。在每个 epoch 结束后，计算测试集上的损失和准确率，并打印出来。
11. **def** train\_softmax\_classifier(x\_train, y\_train, x\_test, y\_test, learning\_rate, num\_epochs, batch\_size):
12. input\_size = x\_train.shape[1]
13. num\_classes = len(np.unique(y\_train))
14. W, b = initialize\_parameters(input\_size, num\_classes)
16. m = x\_train.shape[0]
17. **for** epoch **in** range(num\_epochs):
18. **for** i **in** range(0, m, batch\_size):
19. x\_batch = x\_train[i:i + batch\_size]
20. y\_batch = y\_train[i:i + batch\_size]
21. y\_batch = y\_batch.astype(int)  # 强制转换为整数类型
22. # 计算预测值
23. scores = np.dot(x\_batch, W) + b
24. y\_pred = softmax(scores)
25. # 计算损失
26. loss = cross\_entropy\_loss(y\_batch, y\_pred)
27. # 计算梯度
28. grad = y\_pred
29. grad[np.arange(x\_batch.shape[0]), y\_batch] -= 1
30. grad /= batch\_size
32. dW = np.dot(x\_batch.T, grad)
33. db = np.sum(grad, axis=0)
35. # 更新权重和偏置
36. W -= learning\_rate \* dW
37. b -= learning\_rate \* db
39. # 在每个 epoch 结束后打印损失和准确率
40. y\_test\_pred = softmax(np.dot(x\_test, W) + b)
41. test\_loss = cross\_entropy\_loss(y\_test, y\_test\_pred)
42. test\_accuracy = accuracy(y\_test, y\_test\_pred)
43. **print**(
44. f"Epoch {epoch + 1}/{num\_epochs}, Loss: {loss:.4f}, Test Loss: {test\_loss:.4f}, Test Accuracy: {test\_accuracy:.2%}")
45. 其它辅助函数

定义函数 softmax(x)：该函数实现了Softmax函数，用于将输入向量转换为概率分布。

1. # 定义 softmax 函数
2. **def** softmax(x):
3. e\_x = np.exp(x - np.max(x, axis=-1, keepdims=True))
4. **return** e\_x / e\_x.sum(axis=-1, keepdims=True)

定义函数 initialize\_parameters(input\_size, output\_size)：该函数用于初始化权重矩阵 W 和偏置向量 b，这些参数将在Softmax分类器的训练中学习。

W 被初始化为小的随机值，b 被初始化为全零。

1. # 初始化权重矩阵和偏置向量
2. **def** initialize\_parameters(input\_size, output\_size):
3. W = np.random.randn(input\_size, output\_size) \* 0.001
4. b = np.zeros(output\_size)
5. **return** W, b

定义函数 cross\_entropy\_loss(y\_true, y\_pred)：该函数计算交叉熵损失，用于度量模型的性能。计算交叉熵损失时，将真实标签 y\_true 与模型的预测概率分布 y\_pred 进行比较。

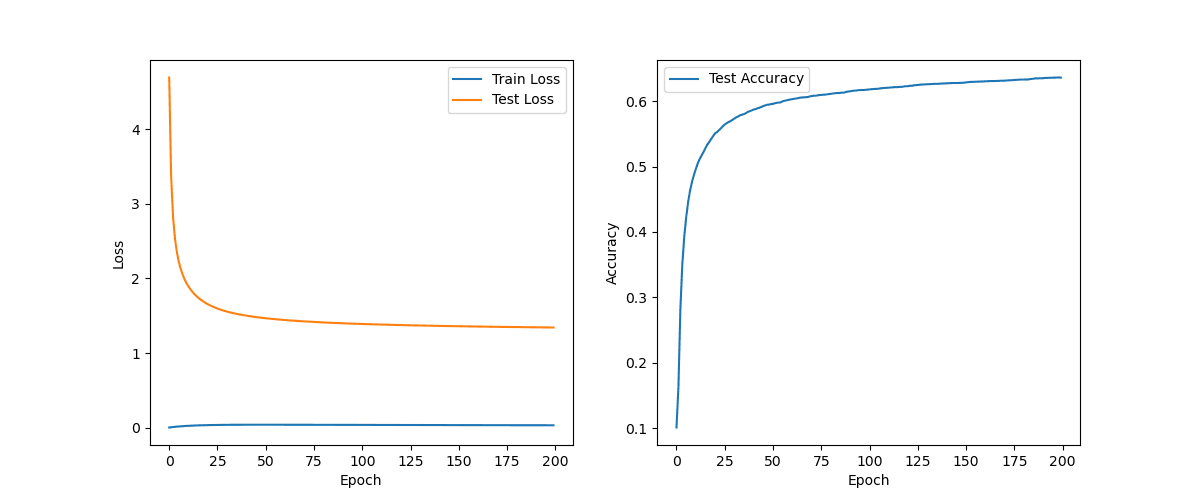
1. # 定义交叉熵损失函数
2. **def** cross\_entropy\_loss(y\_true, y\_pred):
3. m = y\_true.shape[0]
4. y\_true = y\_true.astype(int)  # 强制转换为整数类型
5. **return** -np.sum(np.log(y\_pred[np.arange(m), y\_true])) / m

**2.3.3输出结果**

以学习率为0.1，迭代次数为200，批次为128得到的最终结果是：

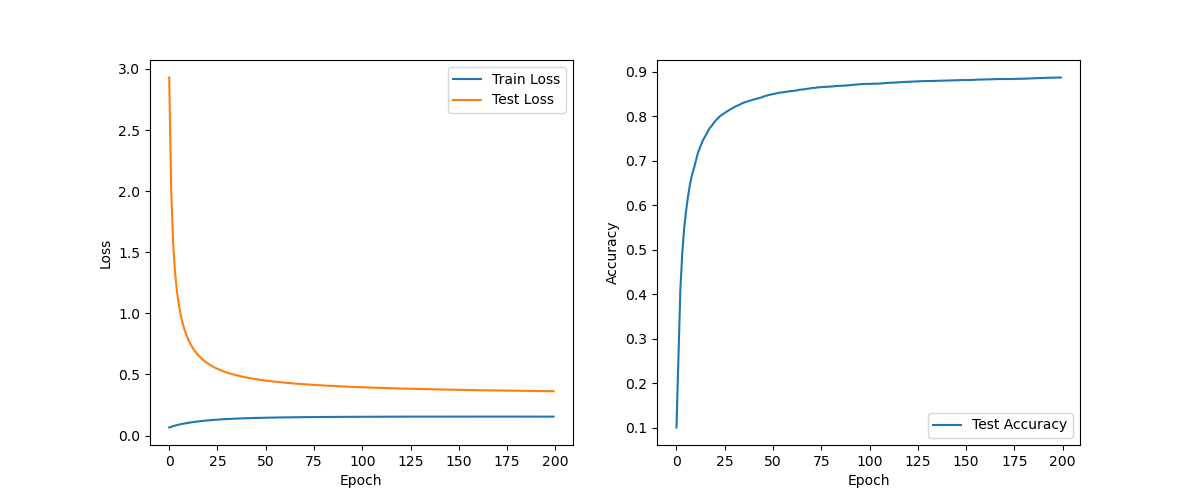
Epoch 200/200, Loss: 0.0343, Test Loss: 1.3429, Test Accuracy: 63.59%

绘出图像为：



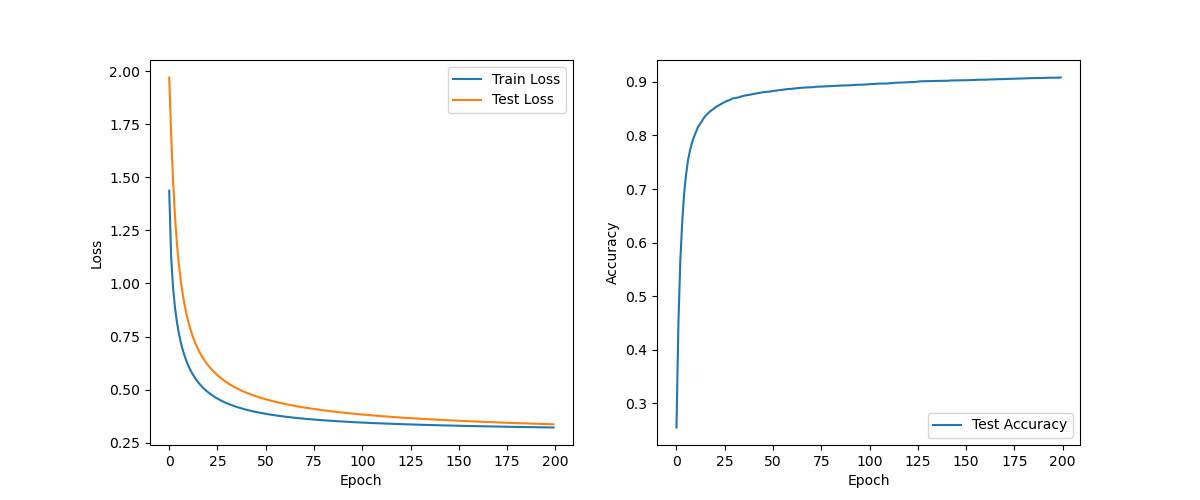
以学习率为0.01，迭代次数为200，批次为128得到的最终结果是：

Epoch 200/200, Loss: 0.1552, Test Loss: 0.3632, Test Accuracy: 88.69%绘出图像为：



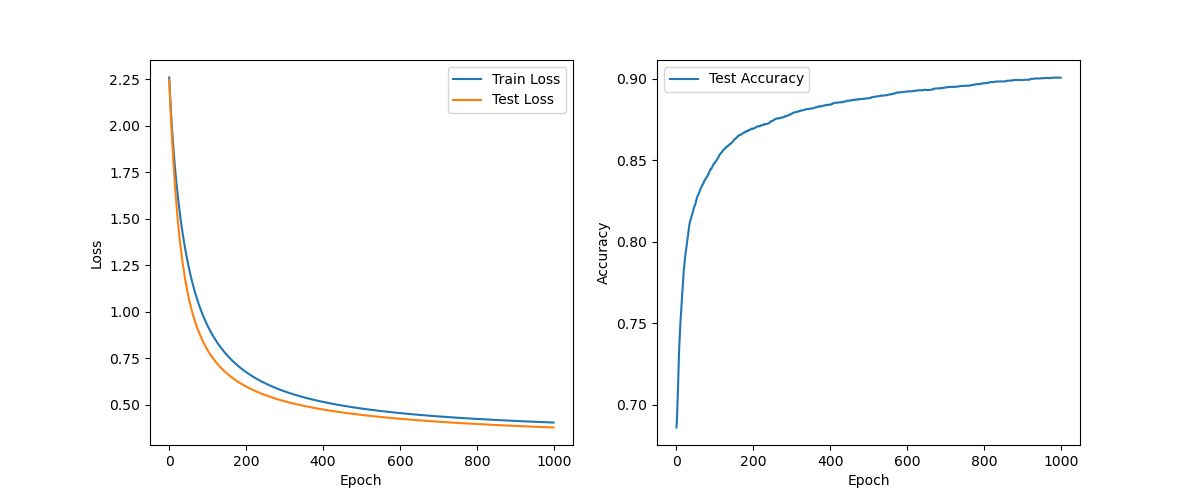
以学习率为0.01，迭代次数为200，批次为128得到的最终结果是：

Epoch 200/200, Loss: 0.3221, Test Loss: 0.3371, Test Accuracy: 90.82%绘出图像为：



以学习率为0.001，迭代次数为1000，批次为128得到的最终结果是：

Epoch 1000/1000, Loss: 0.4052, Test Loss: 0.3785, Test Accuracy: 90.07%绘出图像为：



通过调整学习率可以发现，学习率越低，训练的速度越低，准确率上升越慢，但最终达到的准确率会更高。

**2.4全连接神经网络分类器**

**2.4.1模型结构**