

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **实验1** |
| **姓 名：** | **梁棋棋** |
| **学 号：** | **20120376** |
| **上课类型：** | **平台课/专业课** |
| **日 期：** | **2020.7.23** |

## 一、实验内容

#### 1.1 PyTorch基本操作实验

1. 使用 𝐓𝐞𝐧𝐬𝐨𝐫 初始化一个 𝟏 × 𝟑 的矩阵 𝑴 和一个 𝟐 × 𝟏 的矩阵 𝑵，对两矩阵进行减法操作（要求实现三种不同的形式），给出结果并分析三种方式的不同（如果出现报错，分析报错的原因），同时需要指出在计算过程中发生了什么
   1. 利用 𝐓𝐞𝐧𝐬𝐨𝐫 创建两个大小分别 𝟑 × 𝟐 和 𝟒 × 𝟐 的随机数矩阵 𝑷 和 𝑸 ，要求服从均值为0，标准差0.01为的正态分布；
   2. 对第二步得到的矩阵 𝑸 进行形状变换得到 𝑸 的转置 𝑸𝑻 ；
   3. 对上述得到的矩阵 𝑷 和矩阵 𝑸𝑻 求内积
2. 给定公式 𝑦3 = 𝑦1 + 𝑦2 = 𝑥2 + 𝑥3，且 𝑥 = 1。利用学习所得到的Tensor的相关知识，求𝑦3对𝑥的梯度，即𝑑𝑦3/𝑑𝑥 。要求在计算过程中，在计算 𝑥3 时中断梯度的追踪，观察结果并进行原因分析

#### 1.2 Logistic 回归实验

1. 要求动手从0实现 logistic 回归（只借助Tensor和Numpy相关的库）在人工构造的数据集上进行训练和测试，并从loss以及训练集上的准确率等多个角度对结果进行分析

（可借助nn.BCELoss或nn.BCEWithLogitsLoss作为损失函数，从零实现二元交叉熵为选作）

1. 利用 torch.nn 实现 logistic 回归在人工构造的数据集上进行训练和测试，并对结果进行分析，并从loss以及训练集上的准确率等多个角度对结果进行分析

#### 1.3 Softmax 回归实验

1. 要求动手从0实现 softmax 回归（只借助Tensor和Numpy相关的库）在Fashion-MNIST数据集上进行训练和测试，并从loss、训练集以及测试集上的准确率等多个角度对结果进行分析

（要求从零实现交叉熵损失函数）

1. 利用torch.nn实现 softmax 回归在Fashion-MNIST数据集上进行训练和测试，并从loss，训练集以及测试集上的准确率等多个角度对结果进行分析

## 二、实验设计

### 2.1 Logistic 回归实验

#### 2.1.1 从0实现Logistic 回归实验

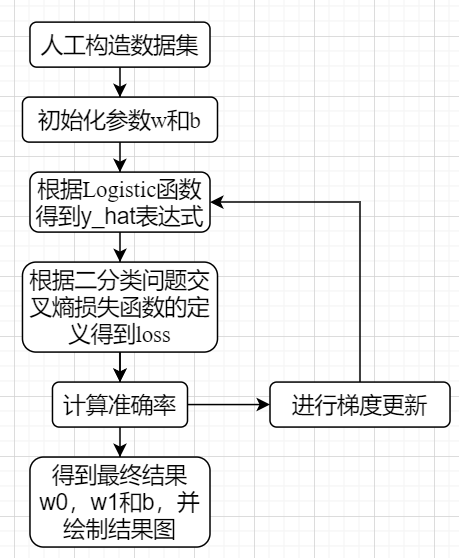


图2.1 从0实现Logistic 回归实验流程图

#### 2.1.2 利用torch.nn实现Logistic 回归实验

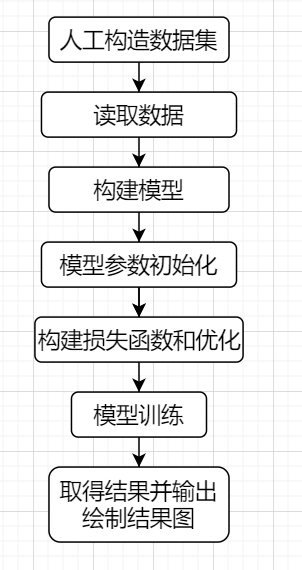


图2.2 利用torch.nn实现Logistic 回归实验流程图

### 2.2 Softmax 回归实验

#### 2.2.1 从0实现Logistic 回归实验

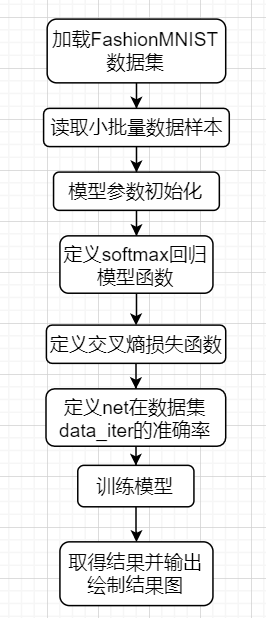


图2.3 从0实现Softmax 回归实验流程图

**2.2.2 利用torch.nn实现Softmax 回归实验**

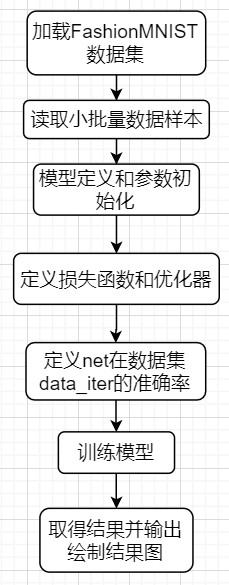


图2.2 利用torch.nn实现Softmax 回归实验流程图

## 三 、实验环境及实验数据集

表3-1 实验环境

|  |  |
| --- | --- |
| 操作系统 | Windows 10 |
| 使用语言 | Python3.6.10 |
| 使用IDE | Jupyter notebook |
| 开发平台 | Pytorch1.5.1 |

实验数据集1：人工构造数据集

实验数据集2：Fashion-MNIST数据集

## 四、实验过程

#### 4.1 PyTorch基本操作实验

1. 初始化两个矩阵并使用三种形式对矩阵进行减法操作



图4.1.1 实验一第1题代码图

1. 创建两个矩阵，其中对一个矩阵矩阵进行转置并求二者的乘法

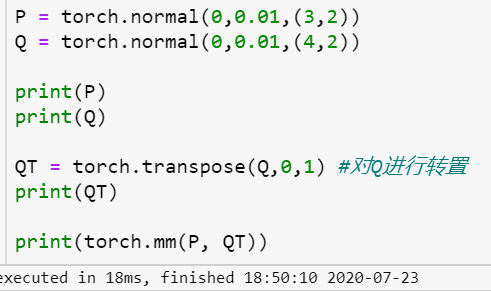


图4.1.2 实验一第2题代码图

1. 先创建一个矩阵x，由于题目要求中断y2梯度的追踪，所以y2部分的代码需要用with torch.no\_grad()包裹，再使用y3.backward(x)进行求梯度运算

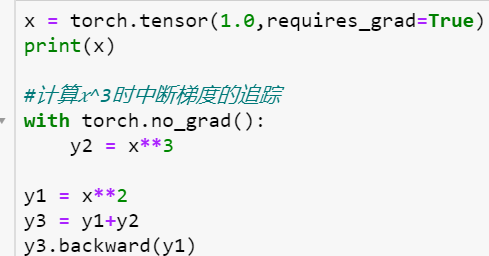


图4.1.3 实验一第3题代码图

#### 4.2 Logistic 回归实验

##### 4.2.1 从0实现

1. 人工构造数据集，为了防止收敛速度过快不好观察实验结果，故正负样本设置为各1000个。

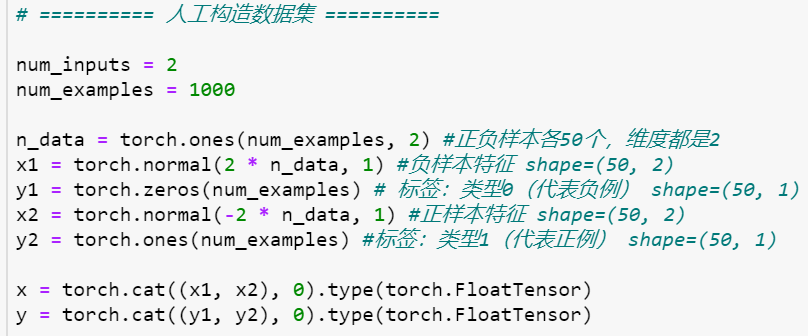


图4.2.1 人工构造数据集代码图

1. 初始化需要被更新的参数w和b，其中w的维度是[2,1]，b的维度是[1,1]，requires\_grad都设置为True用于更新。然后设置学习率和迭代次数。

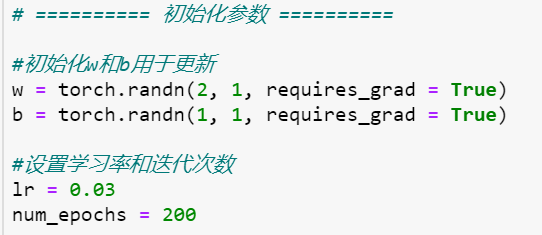


图4.2.2 初始化参数代码图

1. 进行梯度更新，由于grad在反向传播过程中是累加的，所以需要将w和b的梯度清零（注意此处要使用torch.view函数将y的维度和yhat的维度保持一致）。然后记录下每个更新的loss和精度，只有前10个epoch才逐个打印出结果，由于后续的Loss和Acc的变化不大，所以采取每10个epoch打印一次。

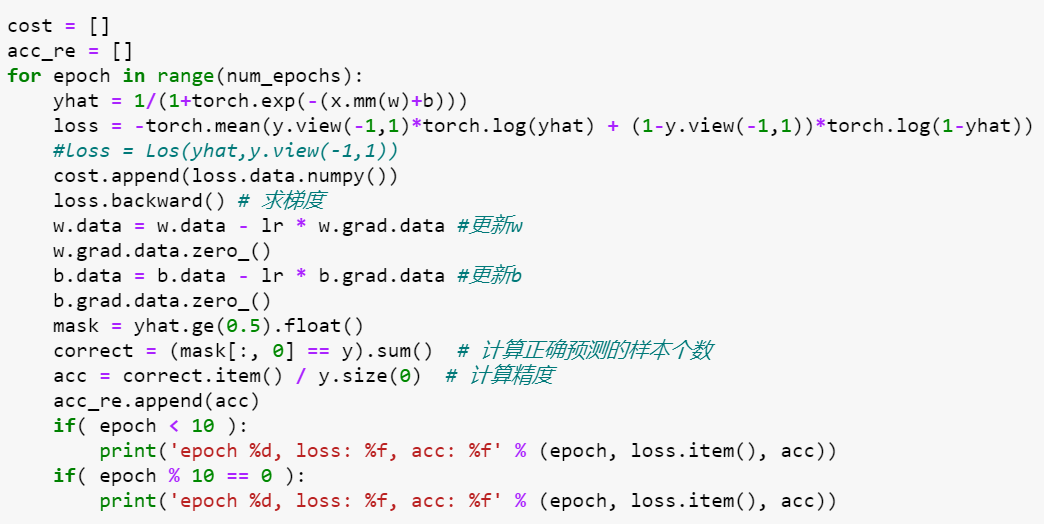


图4.2.3 梯度更新代码图

1. 从w和b中提取出w0、w1和b用于绘制结果图

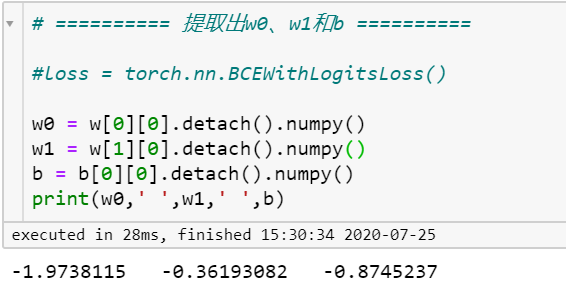


图4.2.4 提出结果代码图

1. 绘制结果

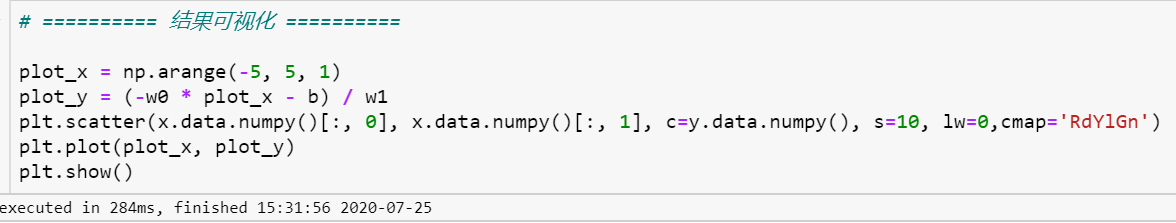


图4.2.5 绘制结果代码图

##### 4.2.2 利用torch.nn实现

1. 人工构造数据集，为了防止收敛速度过快不好观察实验结果，故正负样本设置为各1000个。

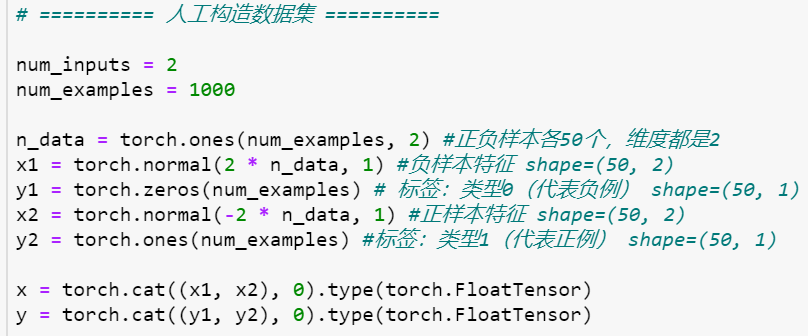


图4.2.6 人工构造数据集代码图

1. 设置学习率为0.03，batch\_size为50。将训练数据的特征和标签组合，把dataser放入DataLoader中，使用Data随机读取包含10个数据样本的小批量。

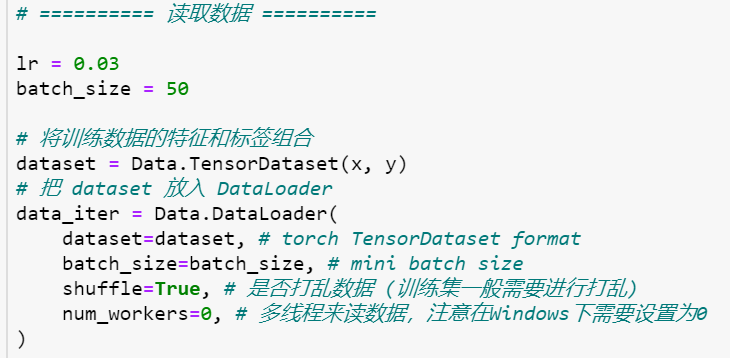


图4.2.7 读取数据代码图

1. 继承nn.Module构建自己的网络，其中包含了Linear层和前向传播方法

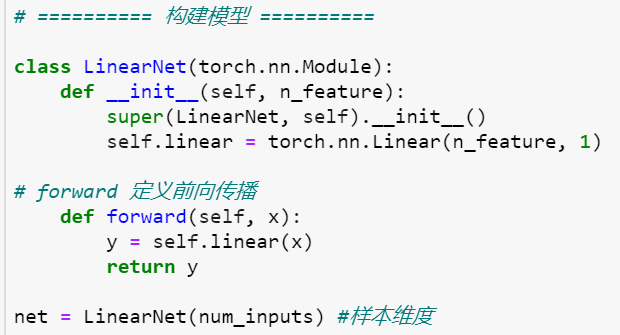


图4.2.8 构建模型代码图

1. 构建损失函数和优化器，使用torch.nn.BCEWithLogitsLoss()作为损失函数，使用小批量随机梯度下降算法进行优化。

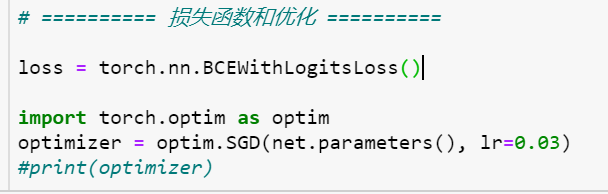


图4.2.9 构建损失函数和优化器代码图

1. 此处使用的是50为一个batch\_size的数据，optimizer.step()通常用在每个mini-batch之中，顺便记录下每个epoch的loss和acc。Output.ge(0.5)的意思是将结果大于0.5的归类为1,结果小于0.5的归类为0，通过该分类与预测输出做比较计算正确预测的个数，再通过正确个数与总个数的比值得到精度。

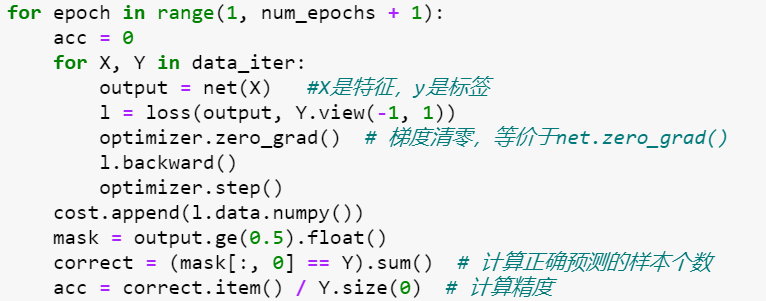


图4.2.10 模型训练代码图

1. 在net的linear层中得到w0，w1和偏正项b，通过这三个参数绘制直线，并结合到x，y图中看是否被正确分类。

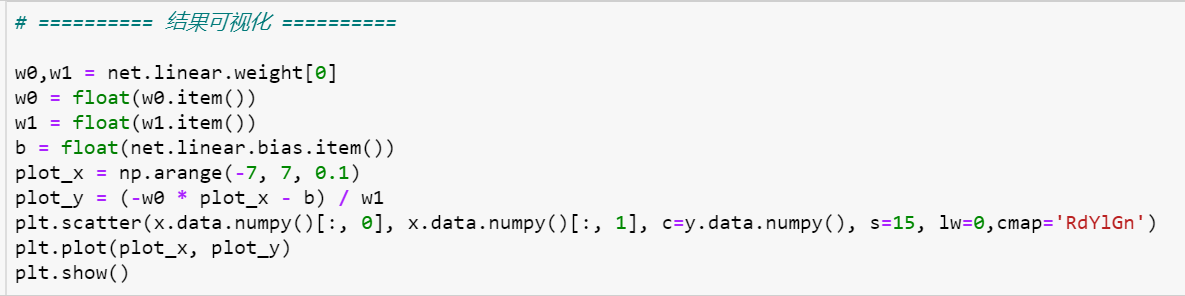


图4.2.11 绘制结果代码图

#### 4.3 Softmax 回归实验

##### 4.3.1 从0实现

1. Softmax回归的线性函数为，而经过Softmax运算后就成了

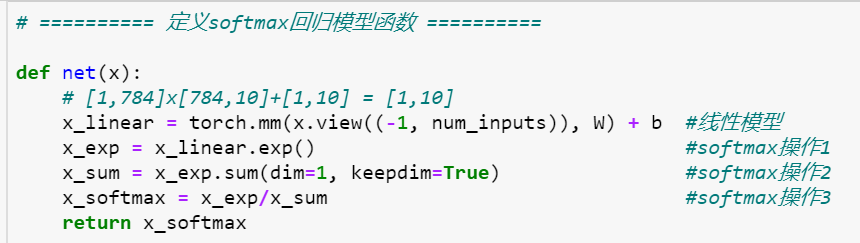


图4.3.1 定义softmax回归模型函数代码图

1. 由于样例的分类代码就是只有一个1，其余都是0，那么y\_hat表示的是某个样本每类的概率分布（加起来为1），y\_hat.gather的意思是找到那个分类的概率（有点类似于下标查找），根据交叉熵损失函数的定义

由于yn只有一个1，所以只需要计算yn是1的那部分即可

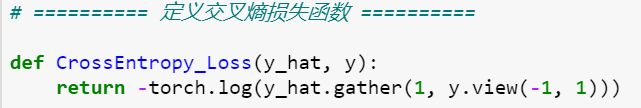


图4.3.2 定义交叉熵损失函数代码图

1. net(X)的维度是[256,10]，y的维度是[256,1]，也就是说net(X).argmax(dim=1)表示net(X)中按行查找的最大概率的下标，，y就是它的分类下标，如果二者相等则可以加1，把256个样例中正确的加起来（对应.sum().item()）得到该样本预测正确的个数，然后除以n得到准确率

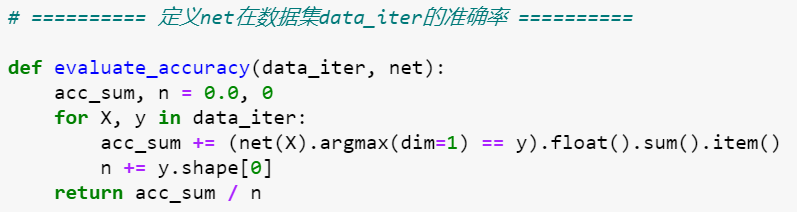


图4.3.3 定义net在数据集上的准确率代码图

1. 首先使用net(X)函数得到y\_hat，也就是一次train\_iter（有256个样例）中256个样例的分类概率，维度为[256,10]；使用交叉熵损失函数得到这256个样例的损失和，进行梯度更新，其中W和b参数的更新采用的是小样本随机梯度下降算法；在此基础上得到训练集的准确率和测试集的准确率。

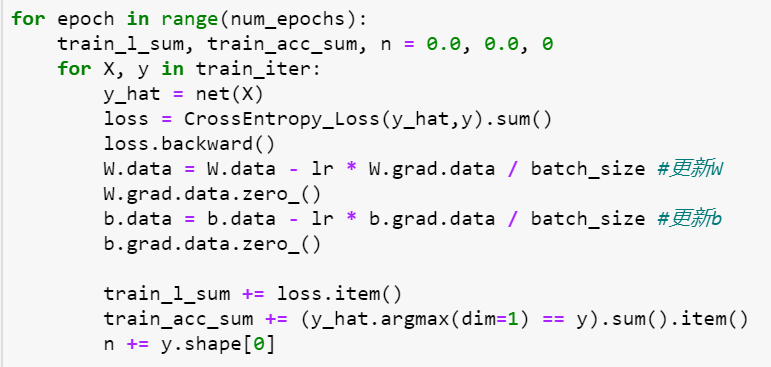


图4.3.4 训练模型代码图

##### 4.3.2 利用torch.nn实现

1. 模型定义部分：和Logister的模型定义类似，只是这次的输出不是只有一个数字而是代表着10个类别的输出概率，所以此处的输出是num\_outputs；在前向传播中x的大小应该是[,784]，但是原始x的大小是[,28,28]，所以我们需要使用torch.view转换一下维度。然后使用torch.nn.init.normal\_和torch.nn.init. constant\_ 初始化W和b。

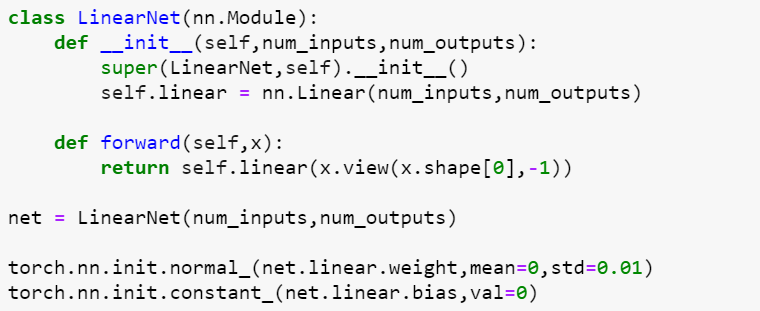


图4.3.5 构建模型和参数初始化代码图

1. 使用torch.nn自带的CrossEntropyLoss()交叉熵损失函数，以及定义一个优化器

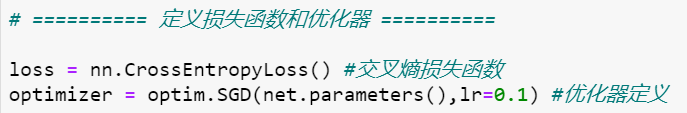


图4.3.6 定义损失函数和优化器代码图

1. 和Logister类似，计算损失，进行梯度更新，然后更新梯度，然后记录下损失，计算准确率。

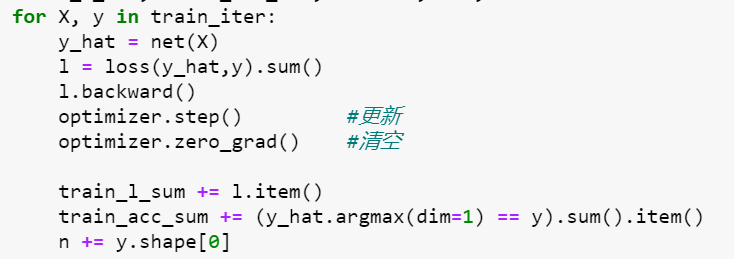


图4.3.7 训练模型代码图

## 五、实验结果

实验结果包括程序运行结果以及对结果的分析，尽量用图表展示实验结果，并且通过结果进行相关的分析。

#### 5.1 PyTorch基本操作实验

**(1)**

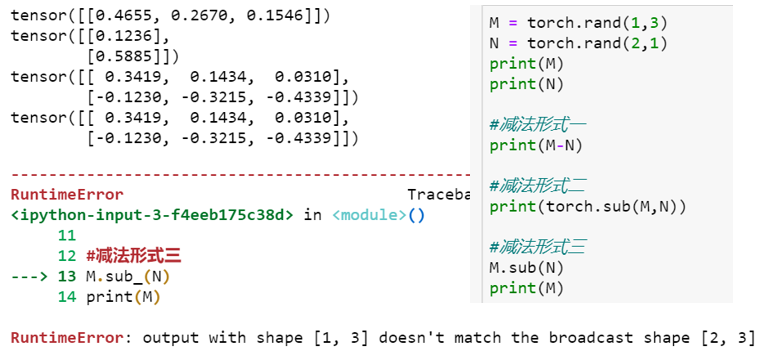


图5.1.1 实验一第1题代码和结果图

可以看出减法形式一和形式二的计算结果相同，减法形式三报错。直接相减和使用torch.sub()方法会自动进行维度的扩展，M将第一行复制到第二行扩展成2x3的矩阵，N将第一列复制到第二第三列也扩展成2x3的矩阵，二者扩展成相同维度后进行减法运算，得到结果相同。而对矩阵使用.sub\_()方法是没有办法将维度进行扩展的，括号中的N将维度扩展成了2x3，但是M没有办法将维度也扩展成2x3，导致运行报错。

**(2)**

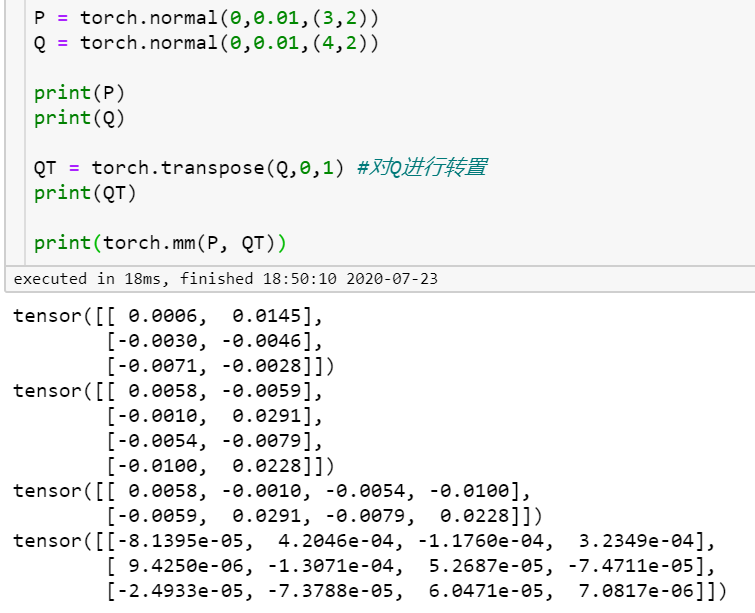


图5.1.2 实验一第2题代码和结果图

使用torch.normal方法创建随机正态分布矩阵P和Q；对Q使用torch.transpose方法进行转置，其中需要注意设置进行转置的维度，代码中设置0就是第一维度4，1就是第二维度2，合并起来就是将4和2换位置，也就是变成2x4维度的矩阵；使用torch.mm方法对P和QT进行矩阵乘法，torch.mm和torch.mul不同，torch.mul要求矩阵对应维度必须相等，是对应位相乘的函数，而torch.mm是做的矩阵乘法，即行乘以列的形式。

**(3)**

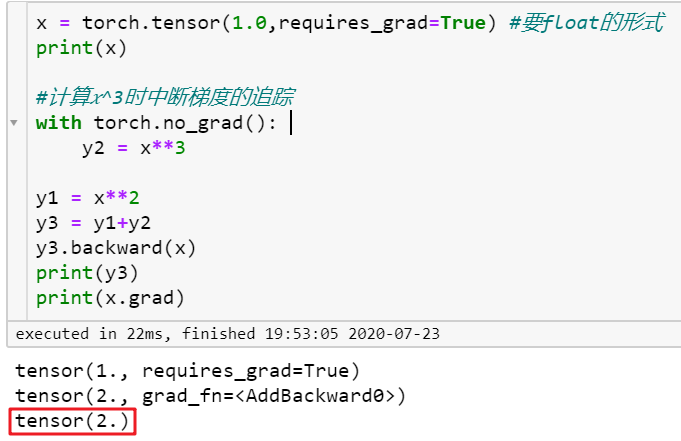


图5.1.3 实验一第3题代码和结果图

可以看到y的结果是2。原因在于y2 = x\*\*3是在with torch.no\_grad()下的，y2的梯度信息被中断，不会回传，所以相当于y3只得到了y1的梯度信息，也就是y3只对x\*\*2求梯度，结果就是2x = 2。

#### 5.2 Logistic 回归实验

##### 5.2.1 从0实现Logistic 回归实验结果

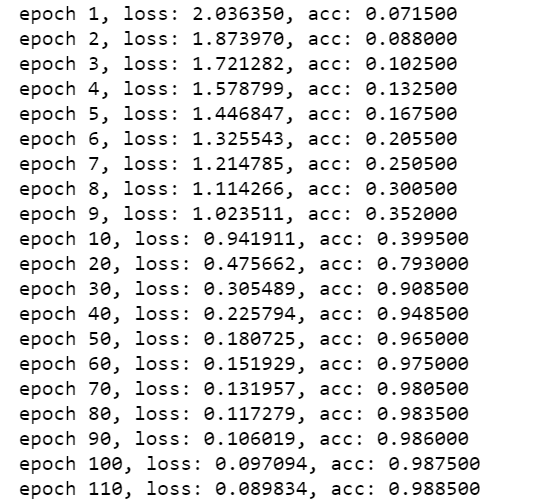


图5.2.1 随epoch的loss和acc结果数字化图

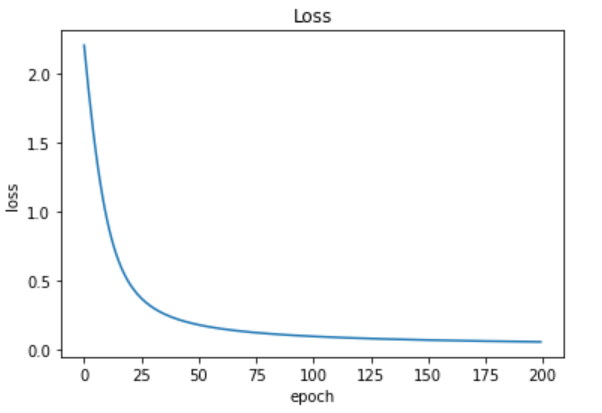


图5.2.2 随epoch变化的Loss结果图

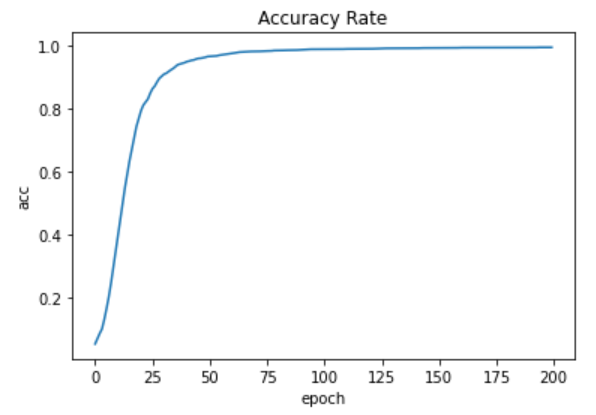


图5.2.3 随epoch变化的准确率结果图

由图中可以看出，随着epoch次数增加，loss损失是逐渐变小的，在200个epoch的时候loss已经降到0.059336了；acc准确率是逐渐升高的，最开始准确率不足10%，在200个epoch的时候准确率已经升到0.993000（也就是99.3%）了。

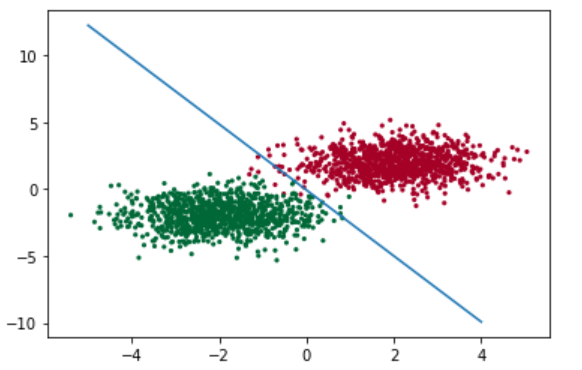


图5.2.4 最终拟合效果图

将数据集和拟合后的直线放在一个图内结合成最终结果，可以看出这条直线可以很好的将图形分类，符合准确率99.3%的数据，通过更多次数的迭代更新，分类效果还会更好。

##### 5.2.1 利用torch.nn实现Logistic 回归实验结果

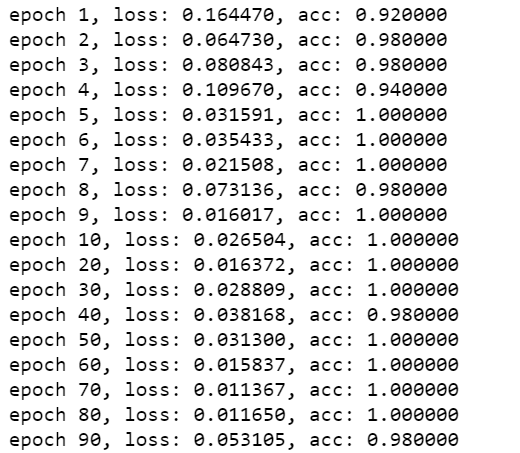


图5.2.5 随epoch的loss和acc结果数字化图

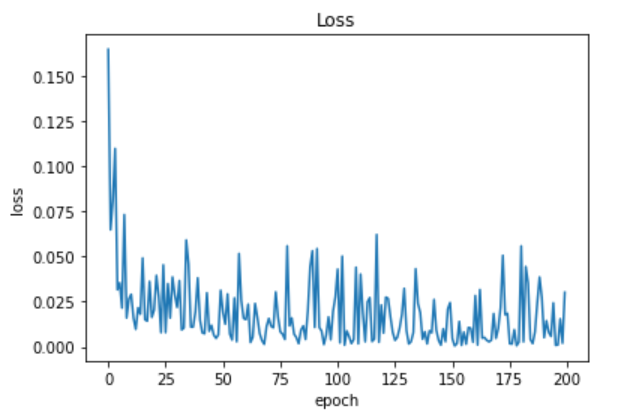


图5.2.6 随epoch变化的Loss结果图

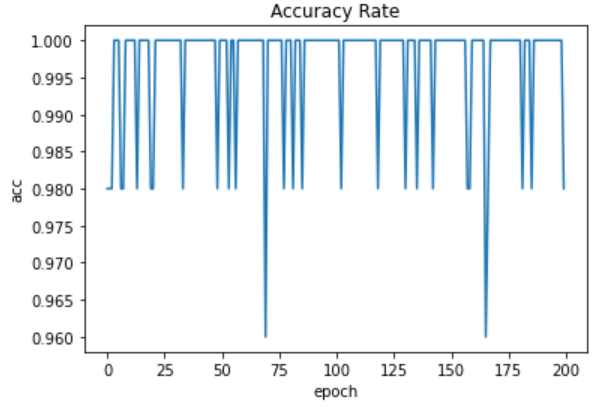


图5.2.3 随epoch变化的准确率结果图

由图中可以看出，随着epoch次数增加，loss损失是整体上是变小的，因为从一开始loss损失就已经是0.16了，经过200个epoch损失也才降到0.005，整体上下降，但实际上没什么变化；acc准确率也无变化，在一开始92%的准确率升到100%之后随着次数的增多准确率开始震荡，可能是学习率设置得太高，样本数量不足或者batch\_size太小导致的。

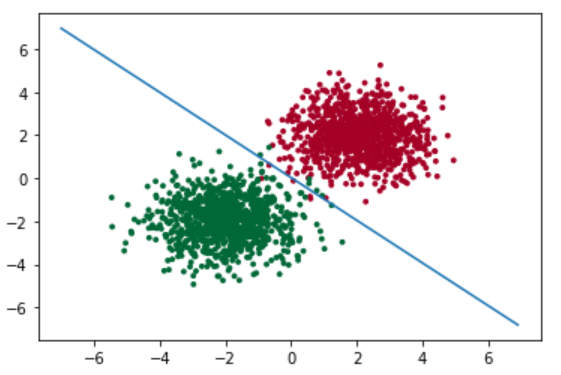


图5.2.4 最终拟合效果图

将数据集和拟合后的直线放在一个图内结合成最终结果，可以看出这条直线可以很好的将图形分类，通过更多次数的迭代更新，分类效果还会更好。

#### 5.3 Softmax 回归实验

##### 5.3.1 从0实现Softmax回归实验结果

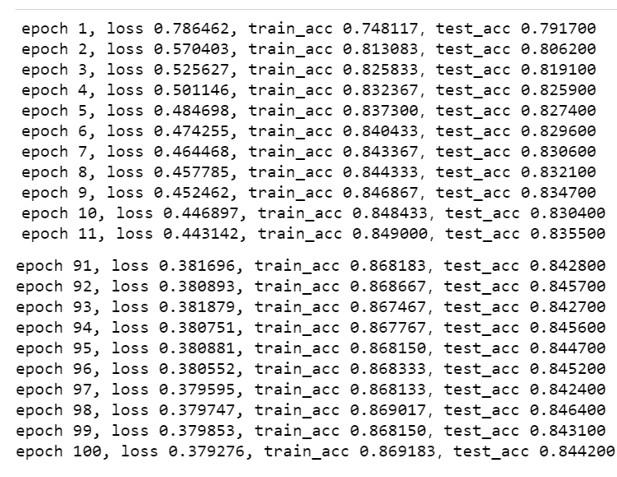


图5.3.1 随epoch变化的的损失、测试集、训练集准确率的数字化图

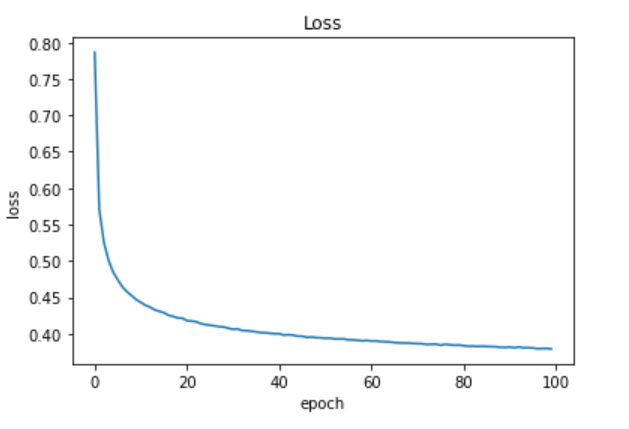


图5.3.2 随epoch变化的Loss（损失）结果图

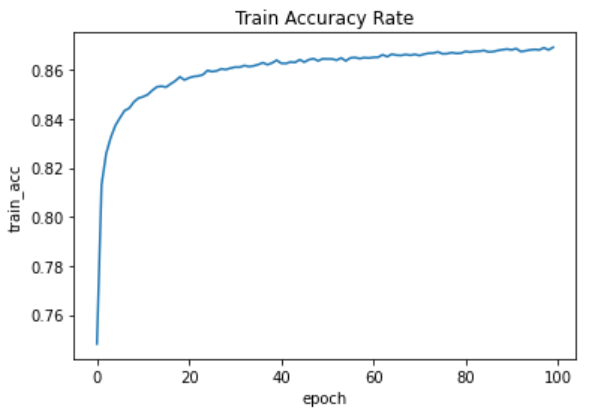


图5.3.3 随epoch变化的train\_acc（训练集准确率）结果图



图5.3.4 随epoch变化的test\_acc（测试集准确率）结果图

从图中可以看出loss是不断下降的，损失在第100次epoch附近已经下降到0.38左右；训练集的准确率是不断上升的，准确率在第100次epoch附近已经上升到87%左右；测试集的准确率也是不断上升的，准确率在第100次epoch附近来回波动，大约保持在84%左右。

##### 5.3.2 利用torch.nn实现Softmax回归实验结果

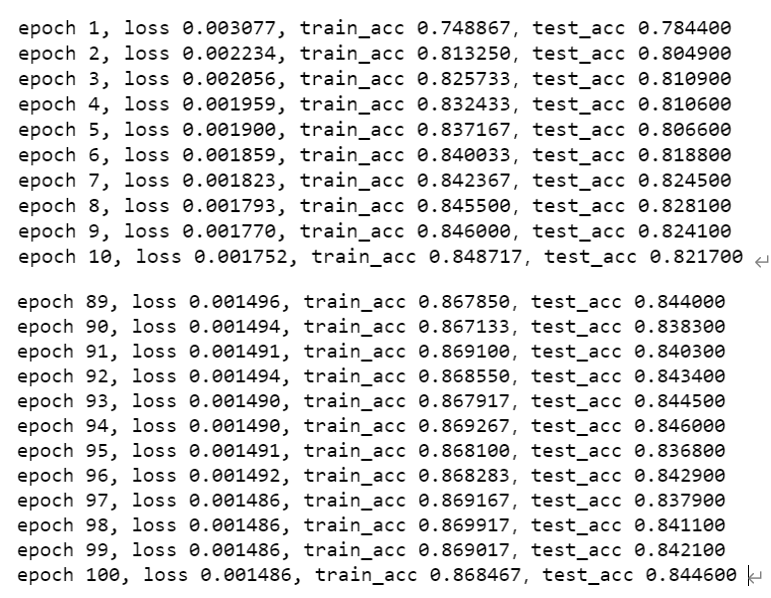


图5.3.5 随epoch变化的的损失、测试集、训练集准确率的数字化图

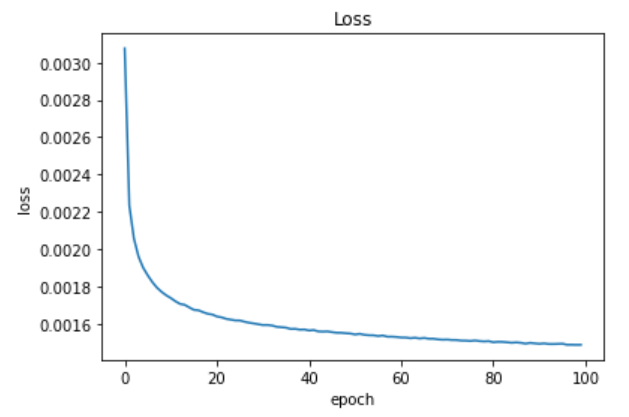


图5.3.6 随epoch变化的Loss（损失）结果图

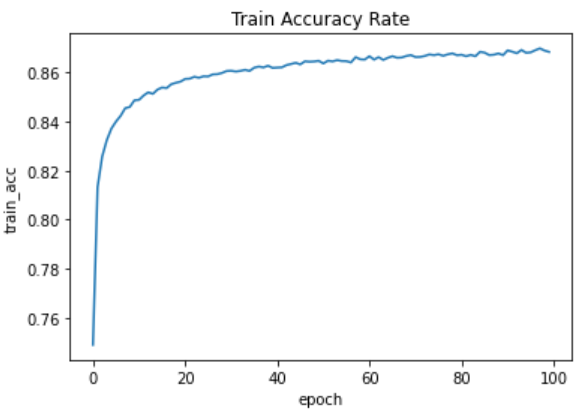


图5.3.7 随epoch变化的train\_acc（训练集准确率）结果图

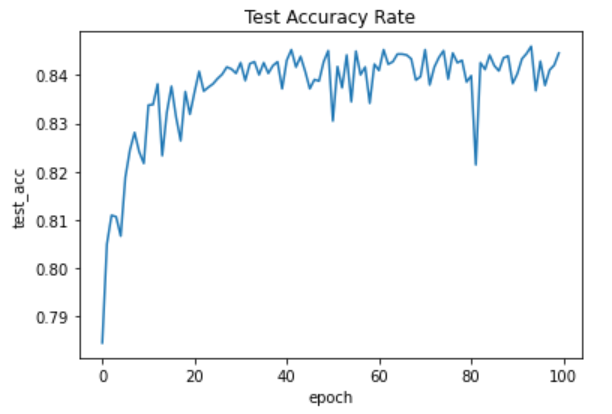


图5.3.3 随epoch变化的test\_acc（测试集准确率）结果图

从图中可以看出loss在第一个epoch的时候就已经很小的，是0.003，自己从0实现的Softmax在100次epoch后损失才0.3，随着epoch增加不断的减小。

训练集的准确率是不断上升的，第一次epoch的训练集准确率和从0实现的Softmax差不多，准确率在第100次epoch附近已经上升到87%左右。

测试集的准确率也是不断上升的，第一次epoch的训练集准确率和从0实现的Softmax差不多，准确率在第100次epoch附近来回波动，大约保持在84%左右。

## 六、实验心得体会

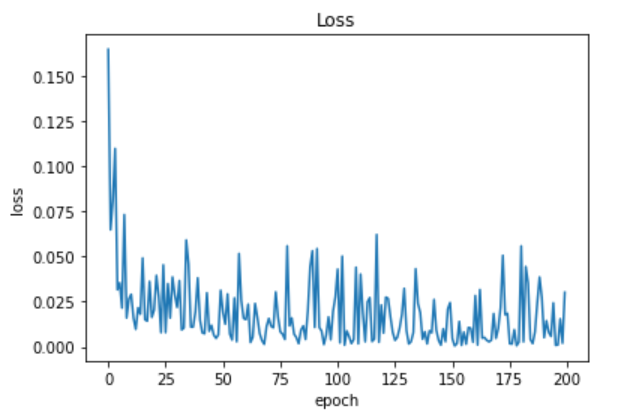
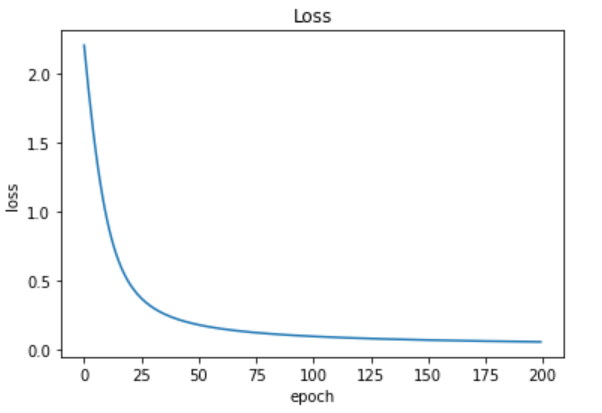
1. torch.mm和torch.mul的区别：torch.mul要求矩阵对应维度必须相等，是对应位相乘的函数，而torch.mm是做的矩阵乘法，即行乘以列的形式。
2. 通过torch.view可以改变torch的维度（类似于reshape），如果第一个参数为-1意味着让电脑帮我们计算，比如说y.shape = [2000]，那么y.view(-1,1)就是2000/1=2000，即[2000,1]；y.view(-1,5)就是2000/5=400，即[400,5]
3. 由于w和b是会被追踪更新的，所以在使用数据的时候不能直接使用w. numpy()，需要用w. detach().numpy()来将其从追踪记录中分离出来才可以使用tensor中的数据。
4. 使用torch.item()可以得到torch的元素值，比如说x = torch.randn(1)，那么print(x) = tensor([-0.4464])，而print(x.item()) = -0.44643348455429077，如果x不是只含一个元素就不可以用item，但是可以访问对应下标用item
5. Torch.argmax(dim=1)表示输出每行中最大值的index，而dim=0表示输出每列中最大值的index
6. 特别需要注意的是维度的问题！！！！！（每句代码要注意的维度相关的已经备注在代码中了，故此不再赘述）

## 七、参考文献

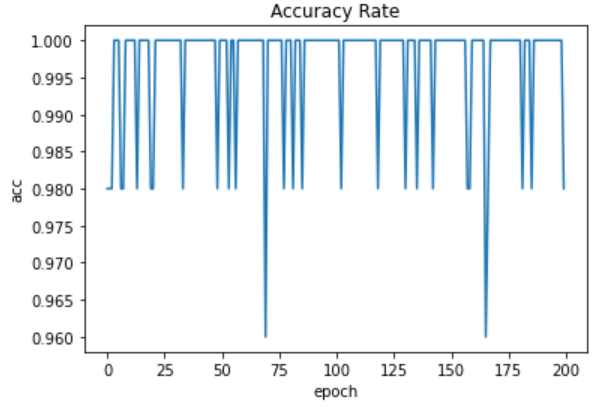
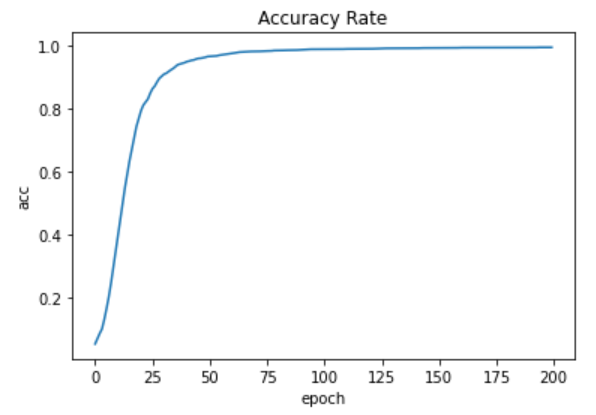
PyTorch版《动手学深度学习》

## 八、附录

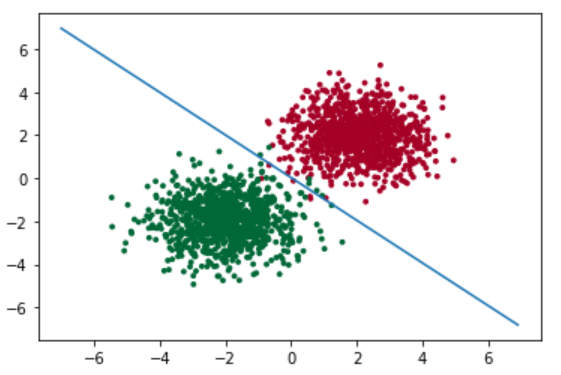
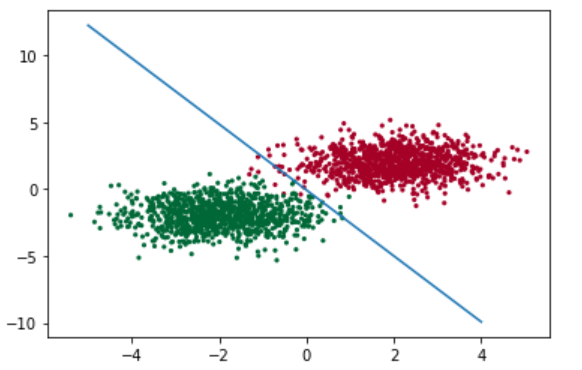
Logister回归实现的Loss结果对比，左图从0实现，右图使用torch.nn



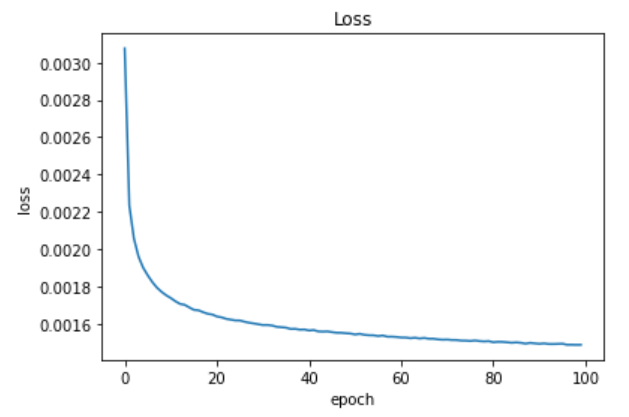
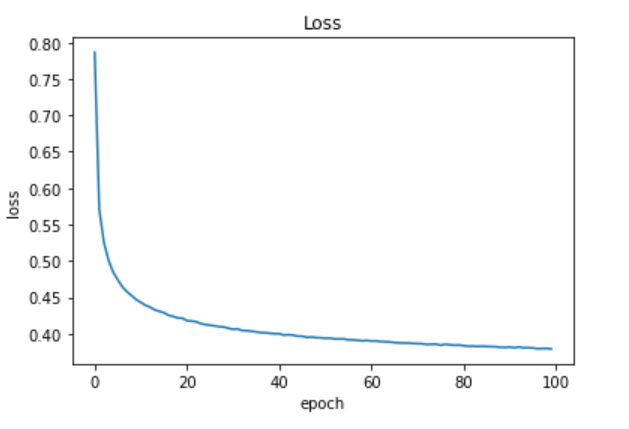
Logister回归实现的Acc结果对比，左图从0实现，右图使用torch.nn



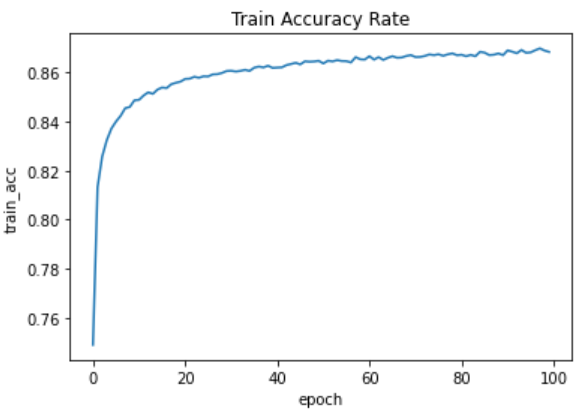
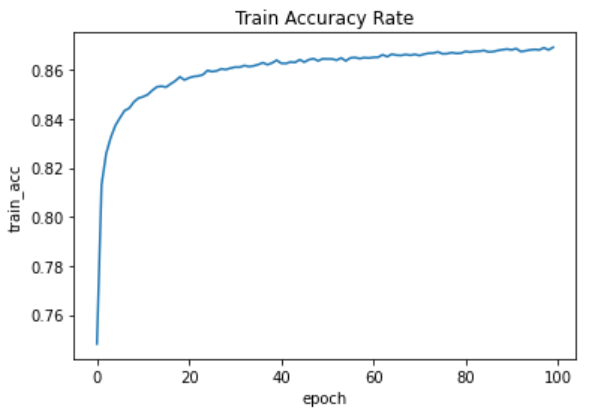
Logister回归实现的拟合结果对比，左图从0实现，右图使用torch.nn



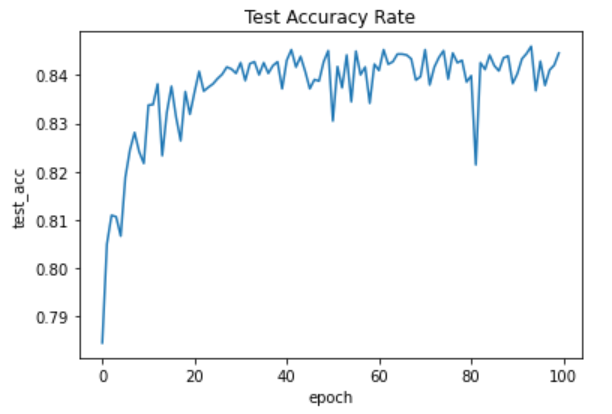
Softmax回归实现的Loss结果对比，左图从0实现，右图使用torch.nn



Softmax回归实现的训练集Acc结果对比，左图从0实现，右图使用torch.nn



Softmax回归实现的测试集Acc结果对比，左图从0实现，右图使用torch.nn



**实验报告编写要求**

1. 正文要求小四号宋体，行间距1.5倍；

2. 英文要求小四号Times New Roman；

3. 在实验内容、实验过程、实验结果三部分需要针对当次实验不同的实验内容分别填写（模版以实验一为例），实验设计中如有必要也可以分开填写；

4. 实验报告配图的每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；

5. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前0.5倍行距。表格中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；

6. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图1、表2、公式(5)等。