

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **前馈神经网络实验** |
| **姓 名：** | **梁棋棋** |
| **学 号：** | **20120376** |
| **上课类型：** | **专业课** |
| **日 期：** | **2020.8.10** |

## 一、实验内容

实验内容包含要进行什么实验，实验的目的是什么，实验用到的算法及其原理的简单介绍。

1、手动实现前馈神经网络解决上述回归、二分类、多分类任务

分析实验结果并绘制训练集和测试集的loss曲线

2、利用torch.nn实现前馈神经网络解决回归、二分类、多分类任务

分析实验结果并绘制训练集和测试集的loss曲线

3、在多分类实验的基础上使用至少三种不同的激活函数

对比使用不同激活函数的实验结果

4、在多分类任务中评估隐藏层层数和隐藏单元个数对结果的影响

使用不同的隐藏层层数和隐藏单元个数，进行对比实验并分析实验结果

5、在多分类任务实验中分别手动实现和用torch.nn实现dropout

探究不同丢弃率对实验结果的影响（可用loss曲线进行展示）

6、在多分类任务实验中分别手动实现和用torch.nn实现L2正则化

探究惩罚项的权重对实验结果的影响（可用loss曲线进行展示）

7、采用10折交叉验证评估回归、二分类、多分类任务实验结果

要求除了最终结果外还需以表格的形式展示每折的实验结果

## 二、实验设计

若实验内容皆为指定内容，则此部分则可省略；若实验内容包括自主设计模型等内容，则需要在此部分写明设计思路、流程，并画出模型图并使用相应的文字进行描述。

#### 1 手动实现前馈神经网络

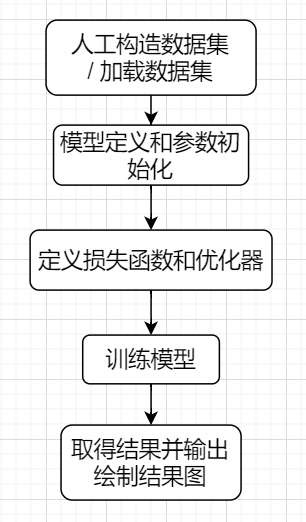


图2.1 手动实现前馈神经网络流程图

#### 2 利用torch.nn实现前馈神经网络

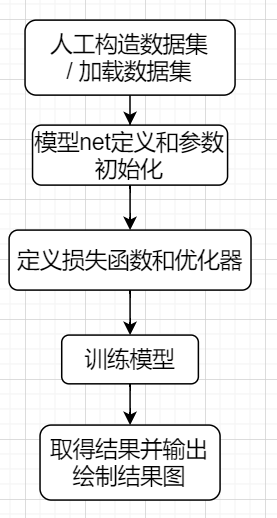


图2.2 利用torch.nn实现前馈神经网络流程图

#### 3 使用不同激活函数实现多分类任务

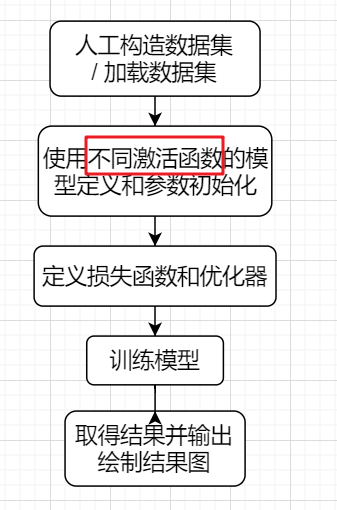


图2.3 使用不同激活函数实现多分类任务流程图

#### 4 使用不同隐藏层层数和隐藏单元个数实现多分类任务

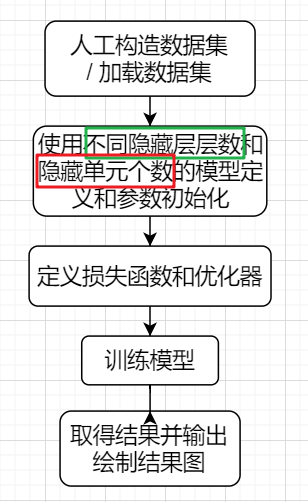


图2.4 使用不同隐藏层层数和隐藏单元个数实现多分类任务流程图

#### 5 在多分类任务实验中分别手动实现和用torch.nn实现dropout

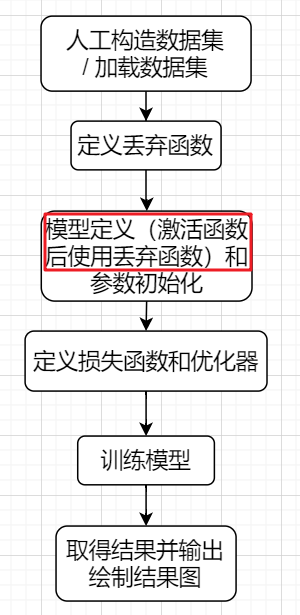


图2.5 手动实现和用torch.nn实现dropout流程图

#### 6 在多分类任务实验中分别手动实现和用torch.nn实现L2正则化

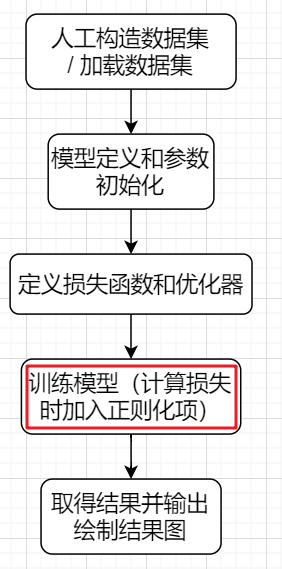


图2.6 手动实现和用torch.nn实现L2正则化流程图

#### 7 采用10折交叉验证评估回归、二分类、多分类任务实验结果

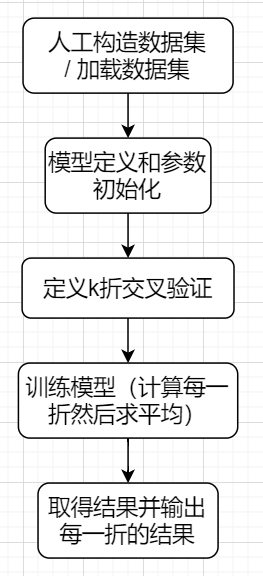


图2.7 采用10折交叉验证评估回归、二分类、多分类任务流程图

## 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：手动生成回归任务的数据集

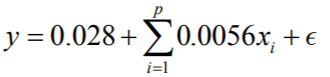
实验数据集2：手动生成二分类任务的数据集

实验数据集3：Fashion-MNIST数据集

## 四、实验过程

#### 4.1 手动实现前馈神经网络

##### 4.1.1 解决回归问题

1、构造数据集：训练集大小为7000，测试集大小为3000，数据集的样本特征维度p为500，且服从如下的高维线性函数：

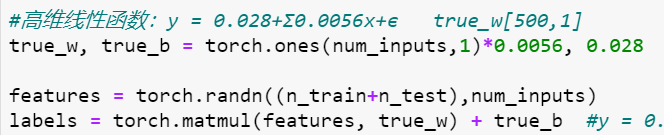


图4.1.1 构造数据集代码图

2、定义模型和损失函数：模型使用 y = Wx + b模型，损失函数使用均方误差

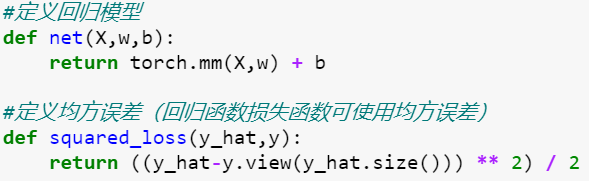


图4.1.2 模型和损失函数定义代码图

3、训练模型：batch\_size设置为128，迭代100次，学习率设置为0.003

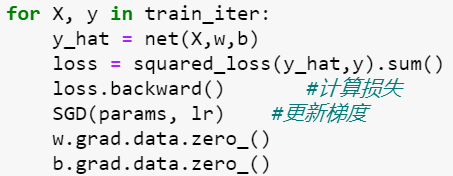


图4.1.3 训练模型代码图

**4.1.2 解决二分类问题**

1、构造数据集：两个训练集大小为7000，测试集大小为3000，两个数据集的样本特征维度p为200，且分别服从均值互为相反数且方差相同的正态分布，两个数据集的样本标签分别为0和1。

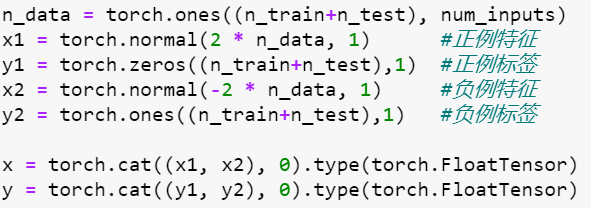


图4.1.4 构造数据集代码图

2、定义模型和损失函数：模型分为输入层、一层隐藏层和输出层，隐藏层单元个数为256，损失函数使用二分类交叉熵损失函数



图4.1.5 模型和损失函数定义代码图

3、训练模型：batch\_size设置为256，迭代100次，学习率设置为0.003

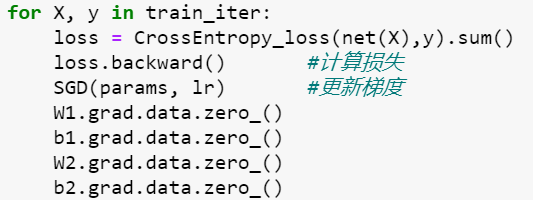


图4.1.6 训练模型代码图

**4.1.3 解决多分类问题**

1、使用MNIST手写体数据集： 该数据集包含60,000个用于训练的图像样本和10,000个用于测试的图像样本



图4.1.7 使用数据集代码图

2、定义模型和损失函数：模型分为输入层、一层隐藏层和输出层，隐藏层单元个数为256，经过隐藏层后使用relu激活函数。损失函数使用交叉熵损失函数，由于CrossEntropyLoss自带Softmax，所以输出层不用加Softmax。

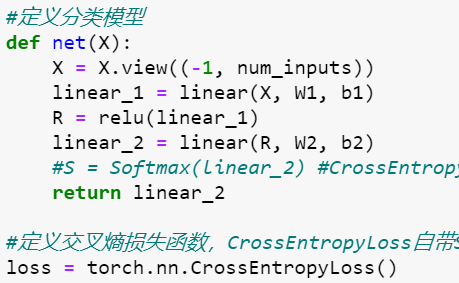


图4.1.8 模型和损失函数定义代码图

3、训练模型：batch\_size设置为256，迭代20次，学习率设置为0.01

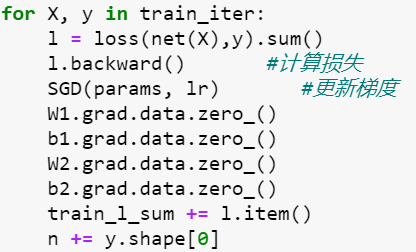


图4.1.9 训练模型代码图

#### 4.2 利用torch.nn实现前馈神经网络

**4.2.1 解决回归问题**

1、定义模型：通过继承torch.nn.Module来定义，模型包括输入层和输出层，由于回归模型是线性的，故不加激活函数

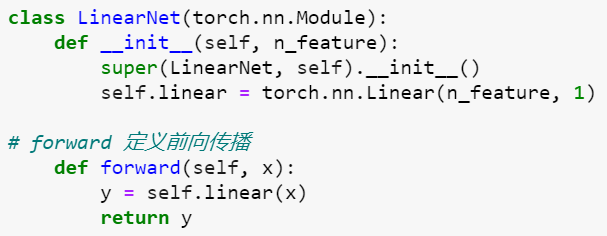


图4.2.1 模型定义代码图

2、定义损失函数和优化器：torch.nn中的均方误差损失函数为MSELoss()

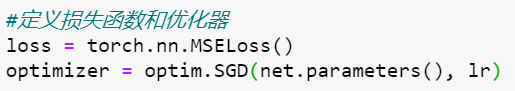


图4.2.2 损失函数和优化器定义代码图

3、训练模型：batch\_size设置为128，迭代100次，学习率设置为0.003

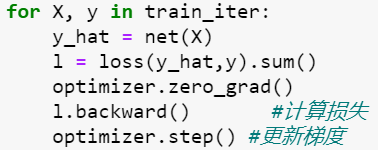


图4.2.3 训练模型代码图

**4.2.2 解决二分类问题**

1、定义模型：通过继承torch.nn.Module来定义，模型包括输入层、一层激活曾和输出层，激活层使用ReLU激活函数，输出层使用Sigmoid激活函数

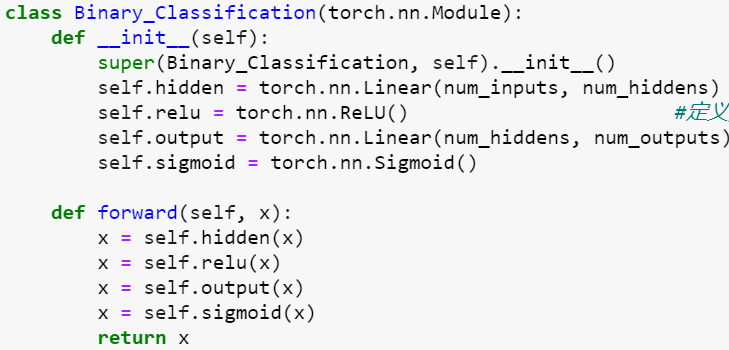


图4.2.4 模型定义代码图

2、定义损失函数和优化器：torch.nn中的二分类交叉熵损失函数为BCELoss ()，由于在定义模型时已经增加Sigmoid激活函数，所以损失函数只需用BCELoss ()。如果模型没加Sigmoid，那损失函数就用BCEWithLogitsLoss()

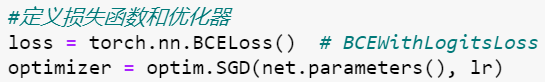


图4.2.5 损失函数和优化器定义代码图

3、训练模型：batch\_size设置为128，迭代100次，学习率设置为0.003

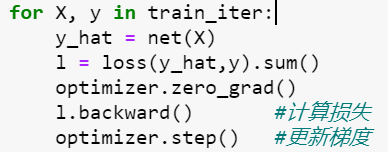


图4.2.6 训练模型代码图

**4.2.3 解决多分类问题**

1、定义模型：通过继承torch.nn.Module来定义，模型包括输入层、一层激活曾和输出层，激活层使用ReLU激活函数

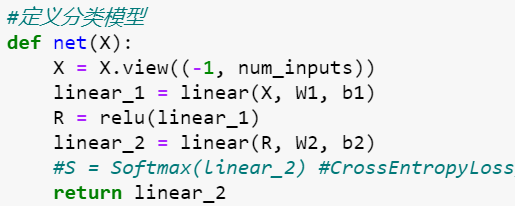


图4.2.7 模型和损失函数定义代码图

2、定义损失函数：torch.nn中的多分类交叉熵损失函数为CrossEntropyLoss ()，由于CrossEntropyLoss自带Softmax，所以输出层中不用加Softmax



图4.2.8 损失函数定义代码图

3、训练模型：batch\_size设置为256，迭代20次，学习率设置为0.01

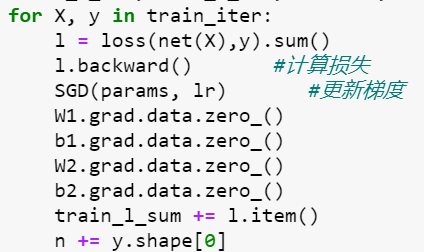


图4.2.9 训练模型代码图

#### 4.3 在多分类基础上使用不同激活函数

1、ReLU激活函数

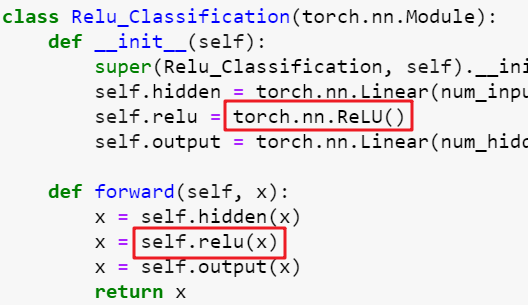


图4.3.1 使用ReLU函数代码图

2、LeakyReLU激活函数

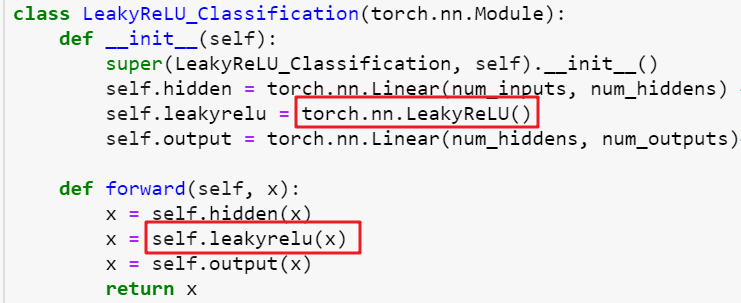


图4.3.2 使用LeakyReLU函数代码图

3、Tanh激活函数

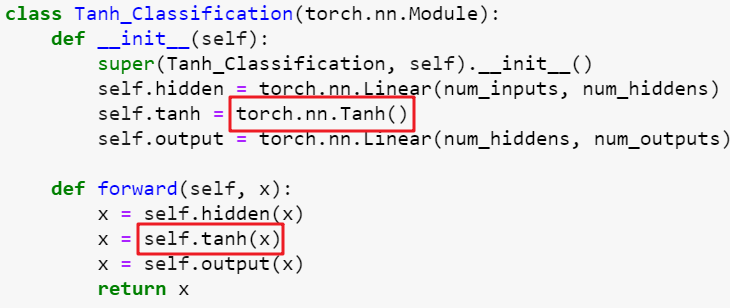


图4.3.2 使用Tanh函数代码图

#### 4.4 在多分类基础上使用不同隐藏层层数和隐藏单元个数

1、一层隐藏层

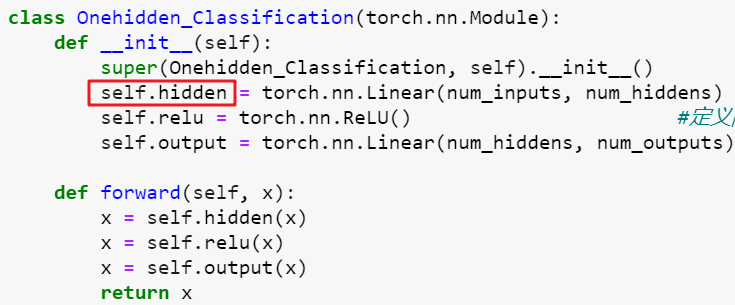


图4.4.1 一层隐藏层代码图

2、两层隐藏层

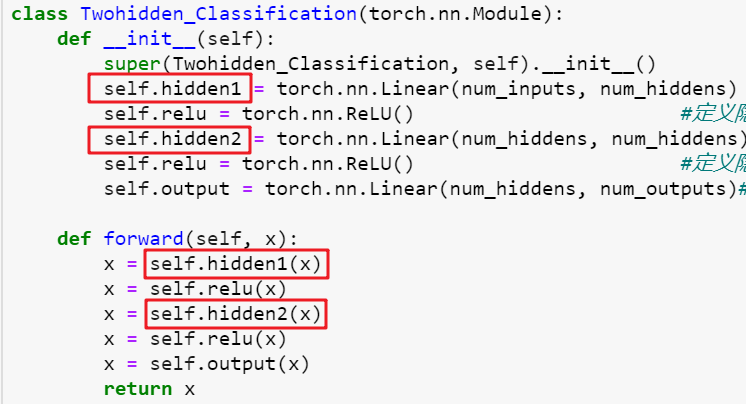


图4.4.2 两层隐藏层代码图

3、三层隐藏层

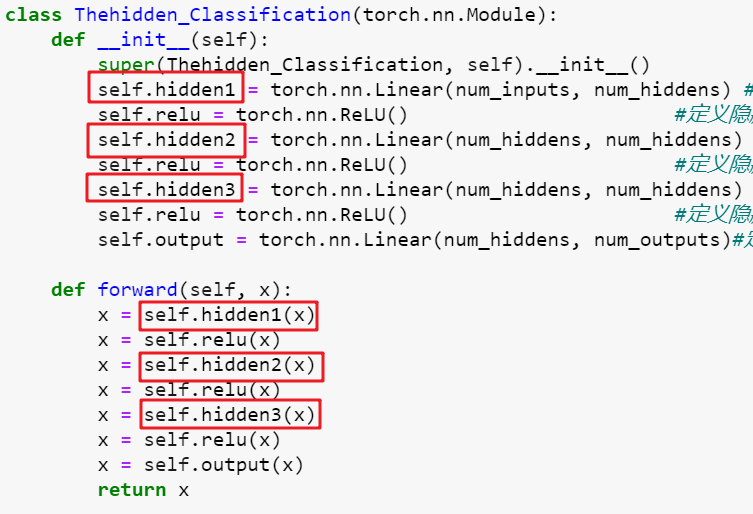


图4.4.3 三层隐藏层代码图

#### 4.5 在多分类基础上实现Dropout

##### 4.5.1 手动实现

1、定义丢弃函数：assert类似于一个报错函数，如果输入的丢弃率不在0 ~ 1之间则报错。torch.zeros\_like(X)是返回一个全0的Tensor，其维度与X一样

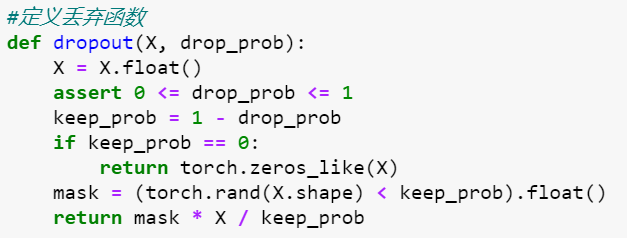


图4.5.1 丢弃函数定义代码图

2、定义模型：在模型中增加is\_training参数，如果is\_training=True则说明需要丢弃，如果is\_training=False（比如在获得测试集loss时）则不需要丢弃

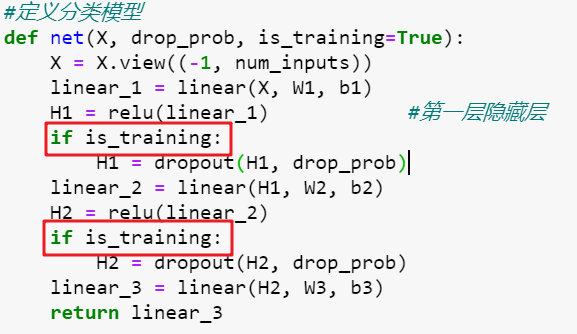


图4.5.2 模型定义代码图

##### 4.5.2 利用torch.nn实现

1、定义模型：在激活函数之后增加Dropout层，参数为丢弃率

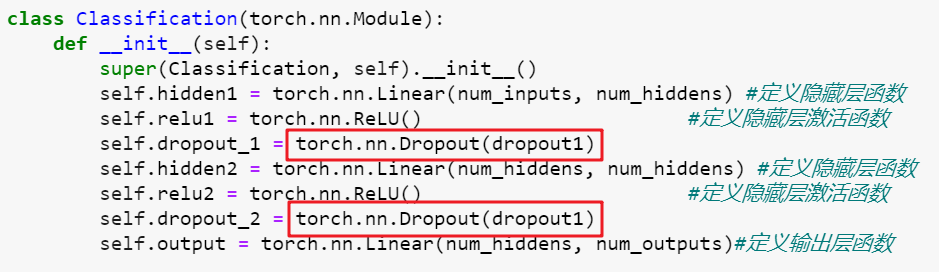


图4.5.3 模型定义代码图

**4.6 在多分类基础上实现L2正则化**

**4.6.1 手动实现**

1、定义L2正则化函数： 权值向量w中各个元素的平方和然后再除以2

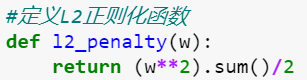


图4.6.1 L2正则化函数定义代码图

2、训练模型：在求loss时增加L2正则化项，其中lambd表示惩罚权重

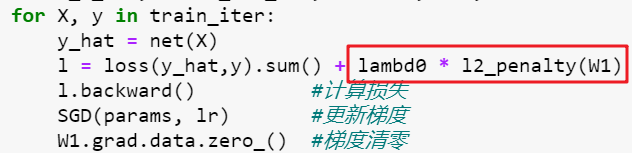


图4.5.2 训练模型定义代码图

**4.6.2 利用torch.nn实现**

1、定义优化器：在优化器中增加weight\_decay设置权重衰减，L2正则化的目的就是为了让权重衰减到更小的值，在一定程度上减少模型过拟合的问题，所以权重衰减也叫L2正则化。

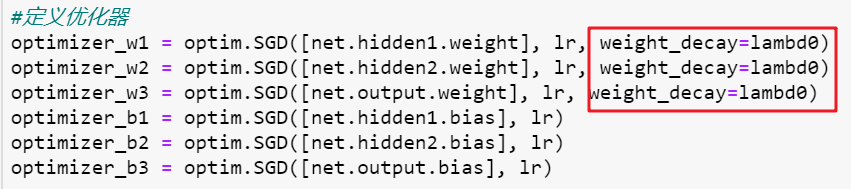


图4.5.3 优化器定义代码图

#### 4.7 实现十折交叉验证

**4.7.1 解决回归问题**

1、定义k折交叉验证：fold\_size是每一折的大小，通过输入的参数i来决定哪一折作为测试集，剩下的作为训练集，X为全部特征，y为全部标签

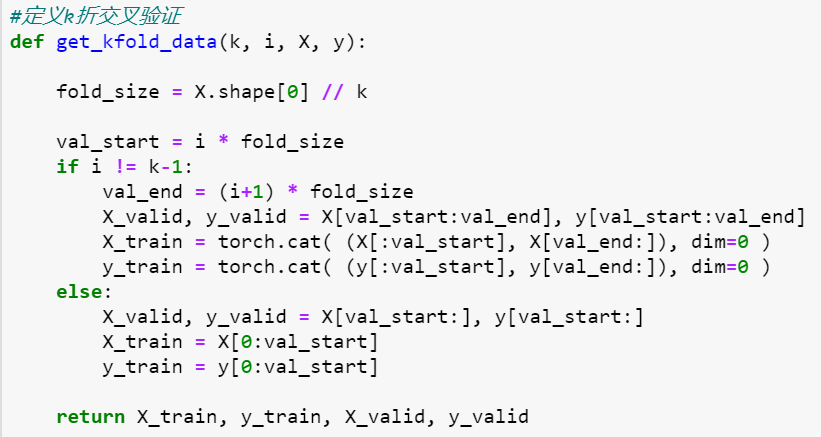


图4.7.1 k折交叉验证定义代码图

2、每一折结果计算函数：通过get\_kfold\_data获得该次折数的训练集和测试集，然后通过训练函数获得每一折的loss和accuracy并输出

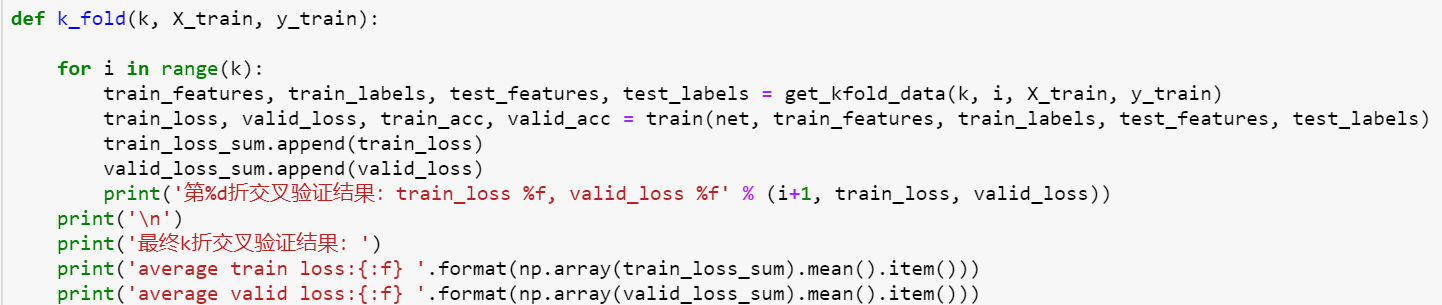


图4.7.2 每一折结果计算函数代码图

3、绘制表格：使用pandas. DataFrame来进行表格的绘制，data中是列的集合，使用为{“列名”：列的集合}，index是索引（默认从0开始）

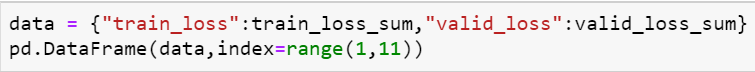


图4.7.3 训练模型代码图

**4.7.2 解决二分类问题**

1、数据集构造：获得正负例之后直接将二者cat起来再进行十折交叉验证

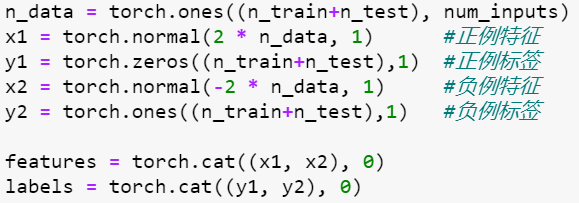


图4.7.4 二分类任务数据集构造代码图

2、准确率定义：由于二分类问题输出层的激活函数使用的Sigmoid，只会输出一个结果，如果这个结果大于0.5则将该类分为正例，如果小于0.5则分为负例，所以使用net(X\_train, W1, W2, b1, b2)>=0.5判断是True还是False（不是1和0的形式，而是True和False的形式），为了使得==两边能比较，所以将y\_train改成y\_train>0.5，如果y\_train=1>0.5说明是True，反之是False。



图4.7.5 准确率定义代码图

**4.7.3 解决多分类问题**

1、数据集构造：获得FashionMNIST之后我们需要对数据集进行重构来使得数据集满足十折交叉验证输入的类型。CrossEntropyLoss()标签部分输入的Tensor必须是Long类型，所以使用torch.LongTensor(labels)改成Long类型

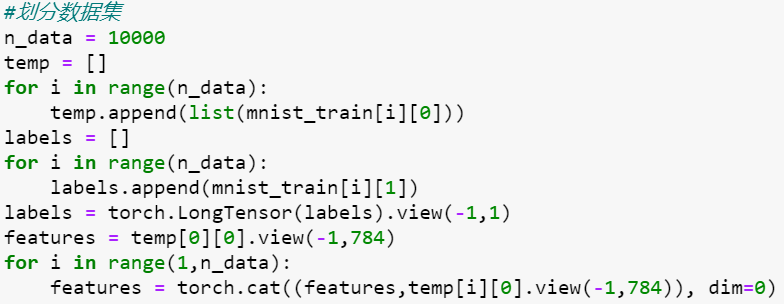


图4.7.6 多分类任务数据集构造代码图

## 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 手动实现前馈神经网络**

1、回归任务结果图：最开始test\_loss和train\_loss都在19左右，经过两个epoch迅速降到0.2，经过三个epoch降到0.02，在第6个epoch之后趋于稳定

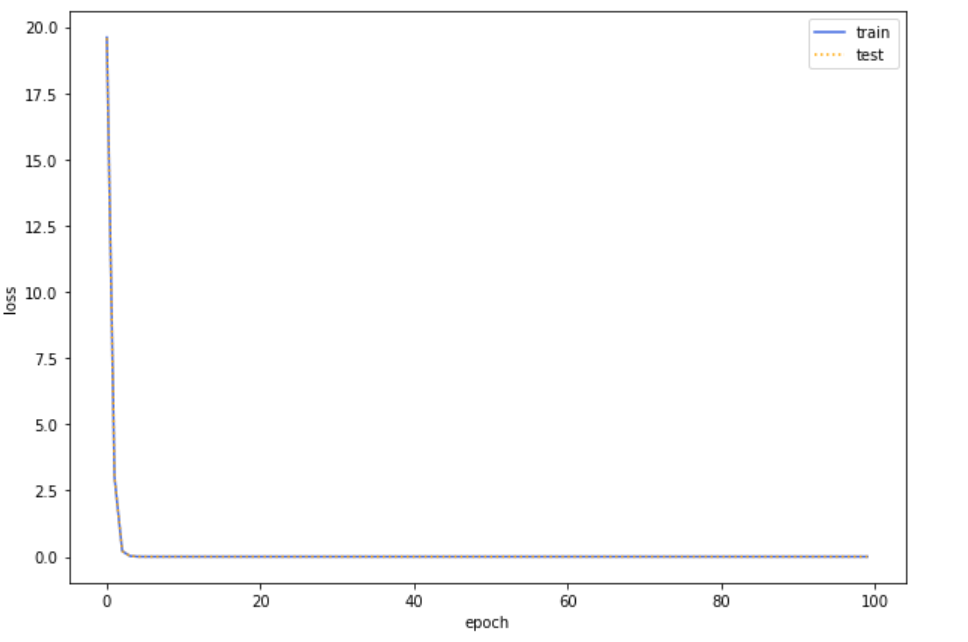


图5.1.1 回归任务结果图

2、二分类任务结果图：最开始test\_loss和train\_loss都在0.31左右，经过两个epoch迅速降到0.04，然后趋于稳定

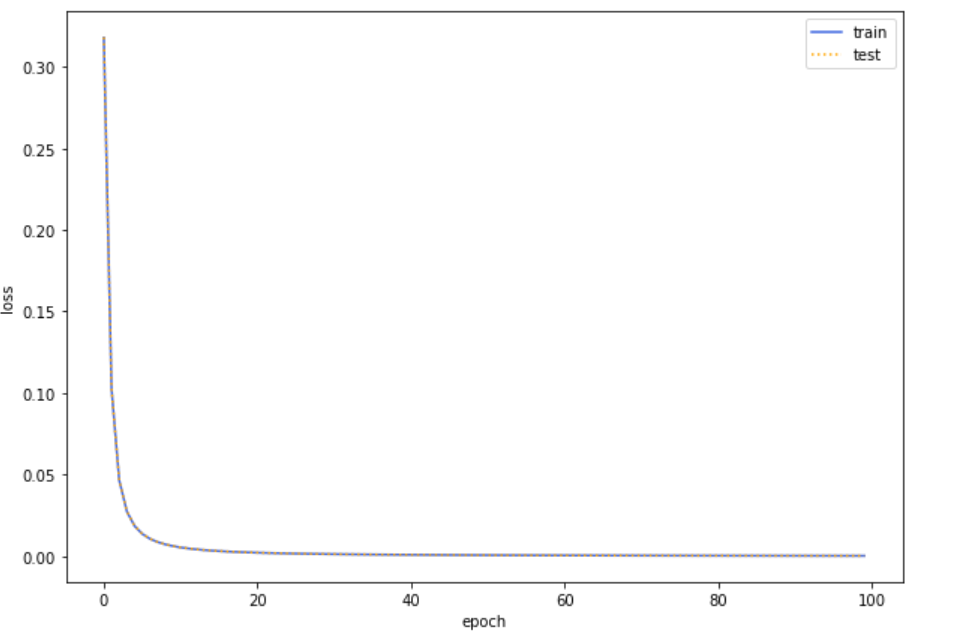


图5.1.2 二分类任务结果图

3、多分类任务结果图：训练集和测试集loss都是逐步下降的，最开始train\_loss高于test\_loss，在第7个epoch之后train\_loss低于test\_loss

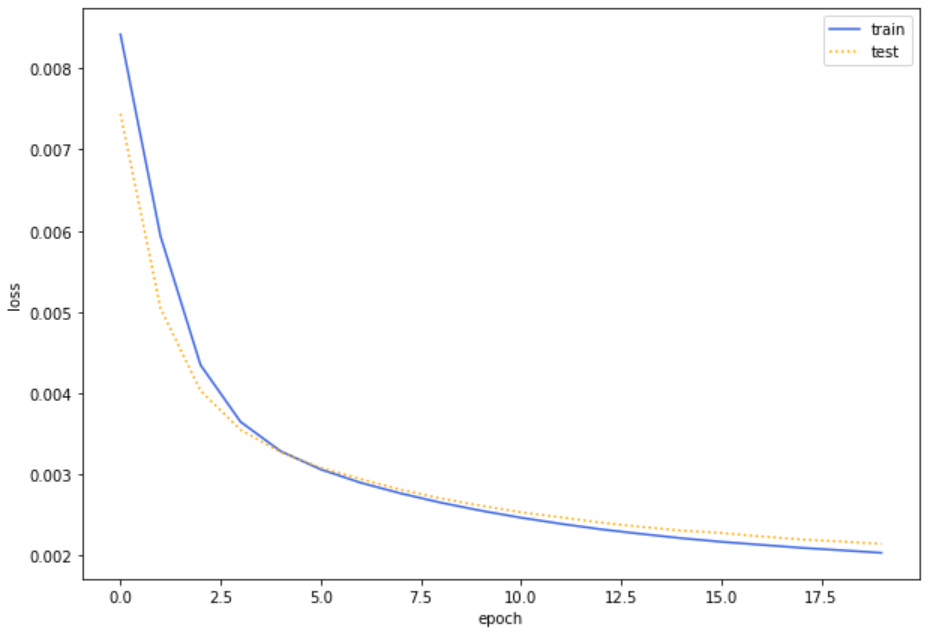


图5.1.3 多分类任务结果图

**5.2 利用torch.nn实现前馈神经网络**

1、回归任务结果图：最开始test\_loss和train\_loss都在0.017左右，train\_loss一直低于test\_loss，但是二者始终保持同样的趋势，在第6个epoch之后趋于稳定

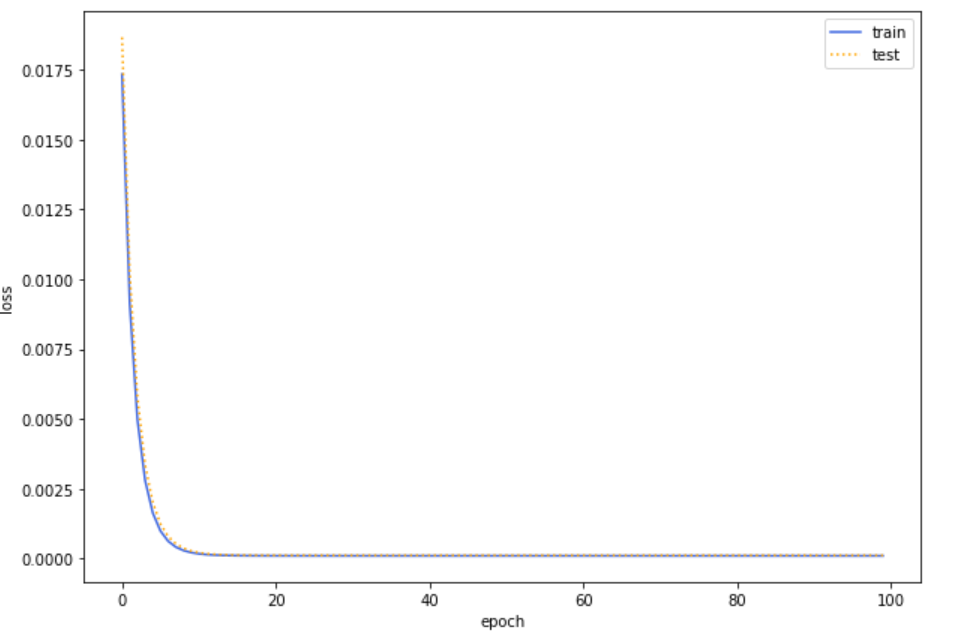


图5.2.1 回归任务结果图

2、二分类任务结果图：最开始test\_loss和train\_loss都在0.09左右，经过4个epoch迅速降到0.008，然后趋于稳定，但是总体上看test\_loss和train\_loss都是逐渐在下降的，只是下降得不明显

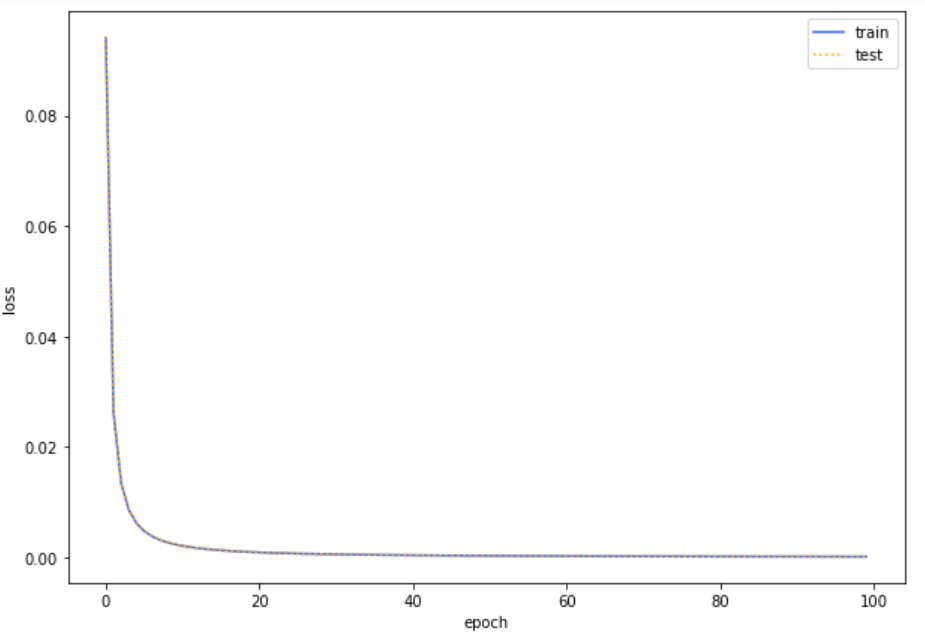


图5.2.2 二分类任务结果图

3、多分类任务结果图：训练集和测试集loss都是逐步下降的，最开始train\_loss高于test\_loss，在第7个epoch之后train\_loss低于test\_loss

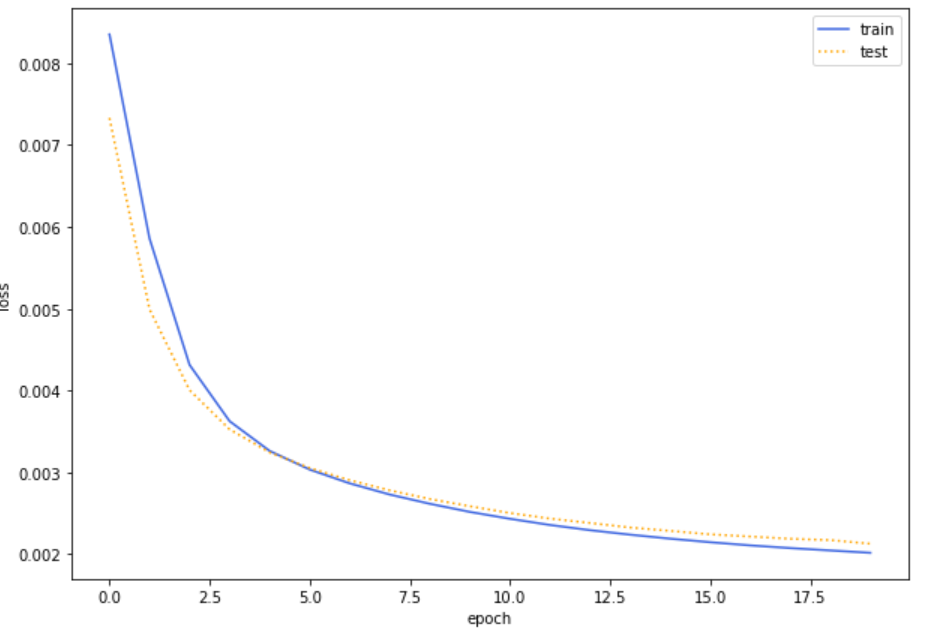


图5.2.3 多分类任务结果图

**5.3 在多分类基础上使用不同激活函数**

1、不同激活函数Train\_Loss结果对比图：Tanh激活函数的Train\_Loss比ReLU和LeakyReLU都要低，ReLU和LeakyReLU的Train\_Loss几乎相同，可以得到对于Train\_Loss来说使用Tanh激活函数效果比较好

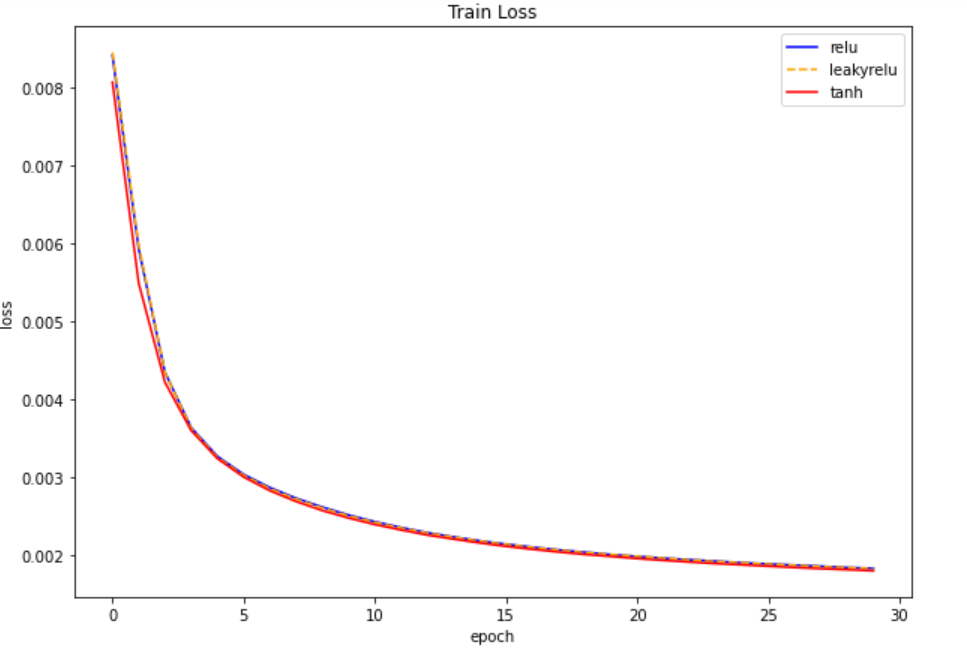


图5.3.1 不同激活函数Train\_Loss结果对比图

2、不同激活函数Test\_Loss结果对比图：Tanh激活函数的Test\_Loss比ReLU和LeakyReLU都要低，ReLU和LeakyReLU的Test\_Loss几乎相同，可以得到对于Test\_Loss来说使用Tanh激活函数效果比较好

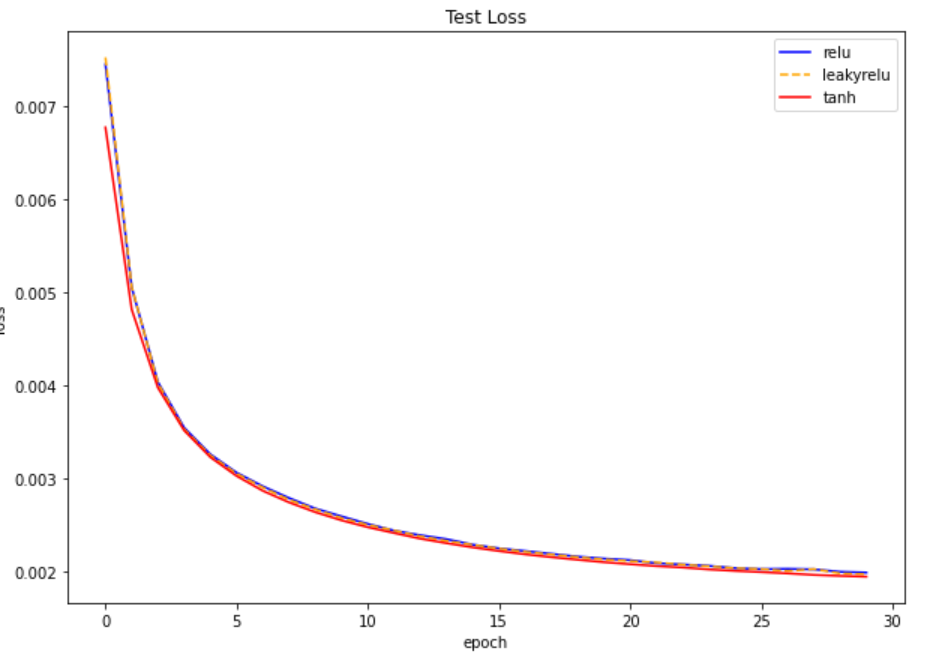


图5.3.2 不同激活函数Test\_Loss结果对比图

3、不同激活函数Train\_Acc结果对比图：Tanh激活函数的Train\_Acc是最高，其次是LeakyReLU，ReLU的Train\_Acc是最低的，可以得到对于Train\_Acc来说使用Tanh激活函数效果比较好，经过30个epoch的Train\_Acc在0.83左右

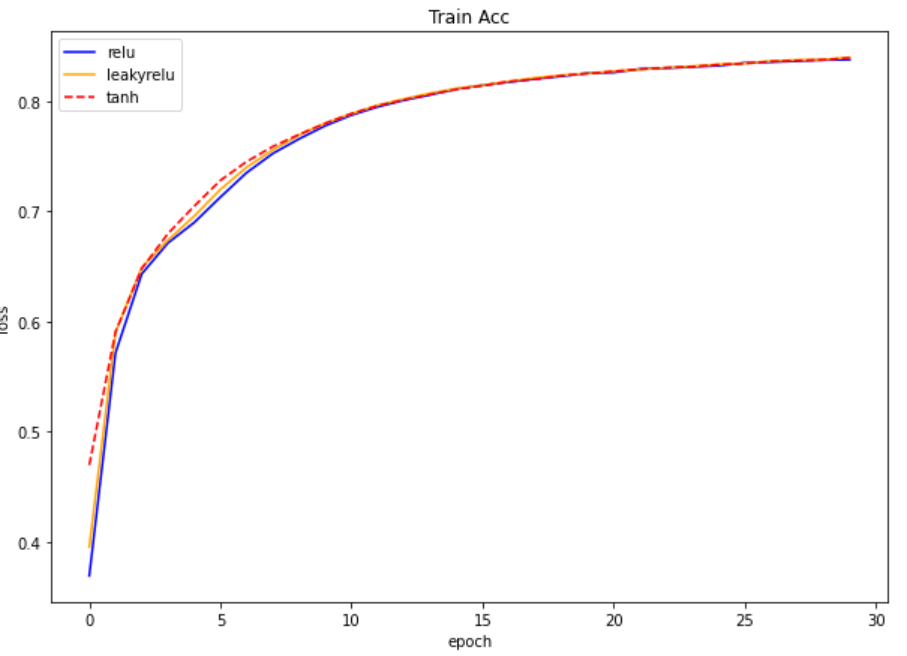


图5.3.3 不同激活函数Train\_Acc结果对比图

4、不同激活函数Test\_Acc结果对比图：Tanh激活函数和LeakyReLU的Test\_Acc差不多都挺高，ReLU的Test\_Acc是最低的，可以得到对于Test\_Acc来说使用Tanh或者LeakyReLU激活函数效果比较好，经过30个epoch的Train\_Acc在0.82左右

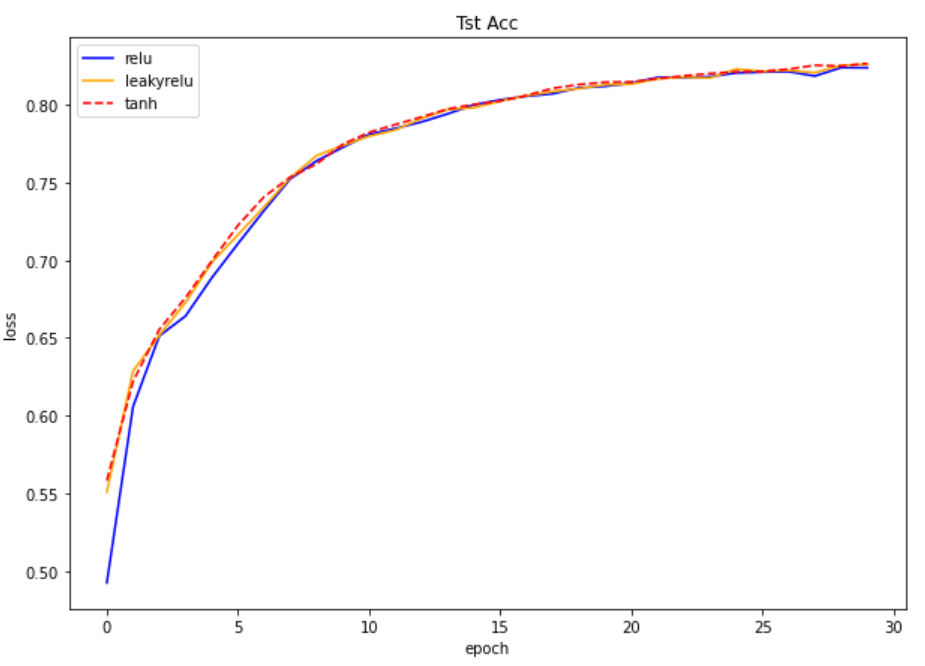


图5.3.4 不同激活函数Test\_Acc结果对比图

**5.4 在多分类基础上使用不同隐藏层层数和隐藏单元个数**

1、不同隐藏层数Train\_Loss结果对比图：一层隐藏层的loss最低，其次是两层隐藏层，三层隐藏层的loss最高，效果最差。一层和两层隐藏层的loss一直在下降，三层隐藏层的loss在第17个epoch之后才开始下降。

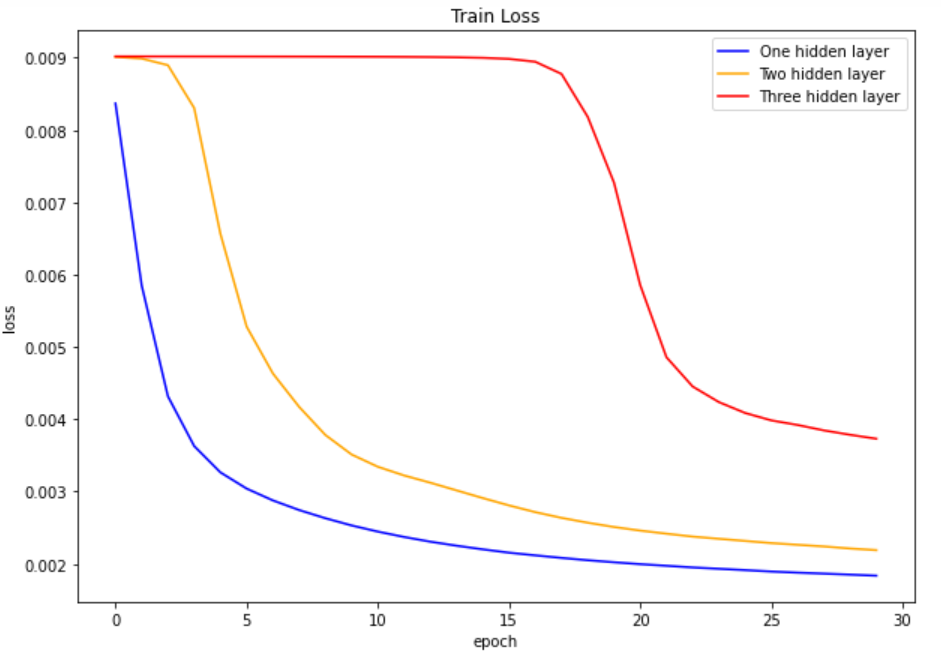


图5.4.1 不同隐藏层数Train\_Loss结果对比图

2、不同隐藏层数Test\_Loss结果对比图：一层隐藏层的loss最低，其次是两层隐藏层，三层隐藏层的loss最高，效果最差。一层和两层隐藏层的loss一直在下降，三层隐藏层的loss在第16个epoch之后才开始下降。

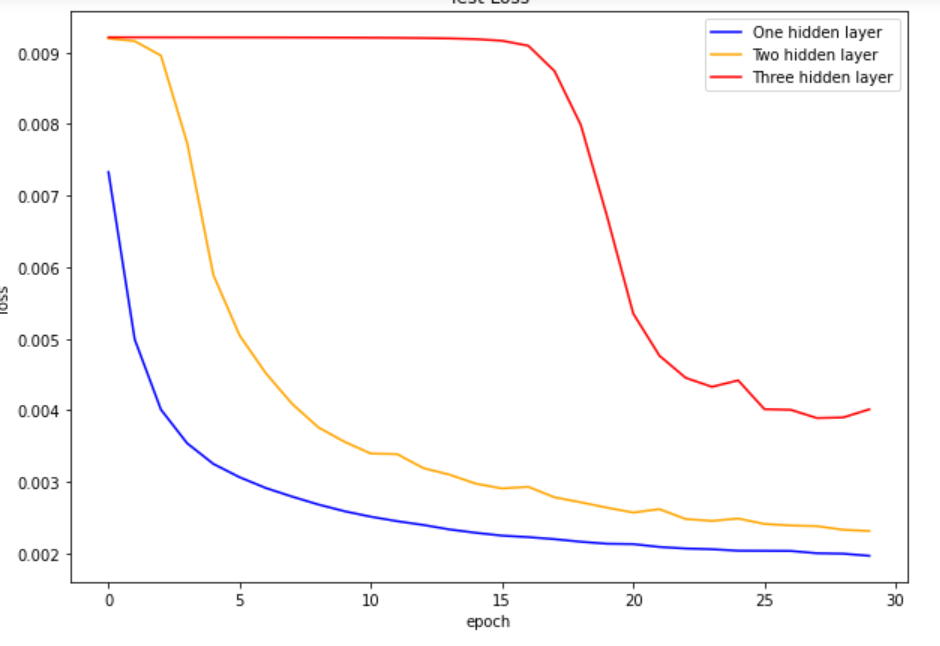


图5.4.2 不同隐藏层数Test\_Loss结果对比图

3、不同隐藏层数Train\_Acc结果对比图：一层隐藏层的acc最高，一开始就有0.46，30epoch后升到0.83；其次是两层隐藏层，一开始只有0.18，30epoch后升到0.80，三层隐藏层的acc效果最差，从0.1上升到0.61，并且不稳定

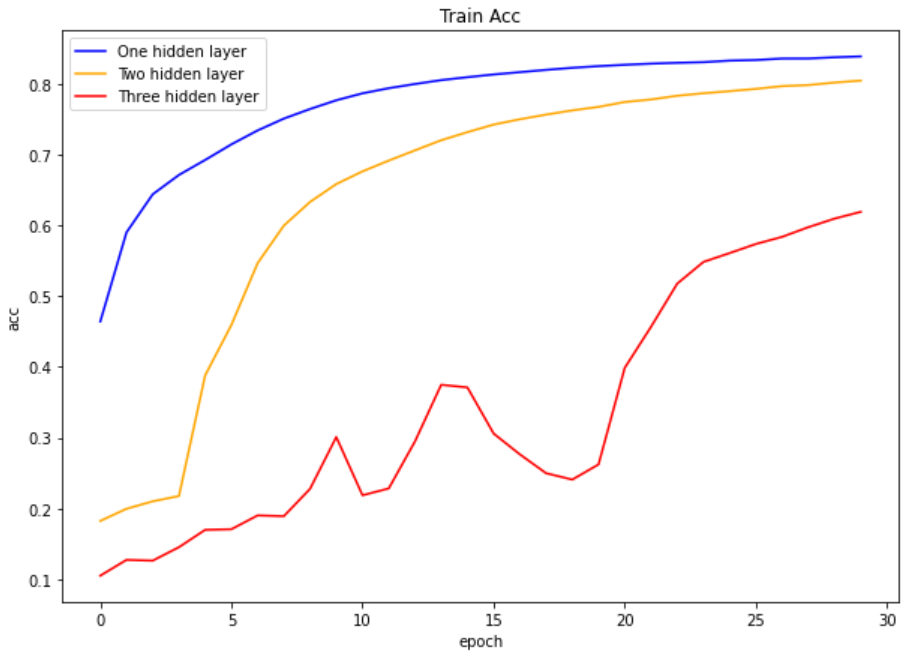


图5.4.3 不同隐藏层数Train\_Acc结果对比图

4、不同隐藏层数Test\_Acc结果对比图：一层隐藏层的acc最高，一开始就有0.49，30epoch后升到0.82；；两三个隐藏层最开始acc都很低，两个隐藏层的acc最开始最低，在第二个epoch之后才比三个隐藏层的高，三个隐藏层的acc不稳定，最高也只在0.61

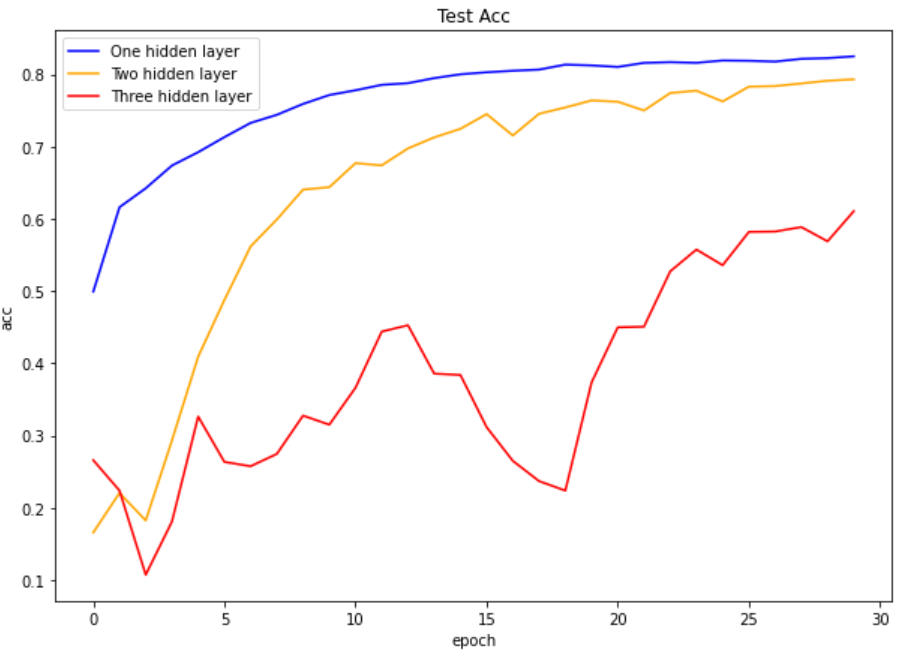


图5.4.4 不同隐藏层数Test\_Acc结果对比图

5、不同隐藏单元个数Train\_Loss结果对比图：256隐藏单元个数的loss最低，其次是128隐藏单元个数，64隐藏单元个数的loss最高。但是以大概数值来看，三个隐藏单元个数的loss都差不多，且都在逐步下降并趋于稳定

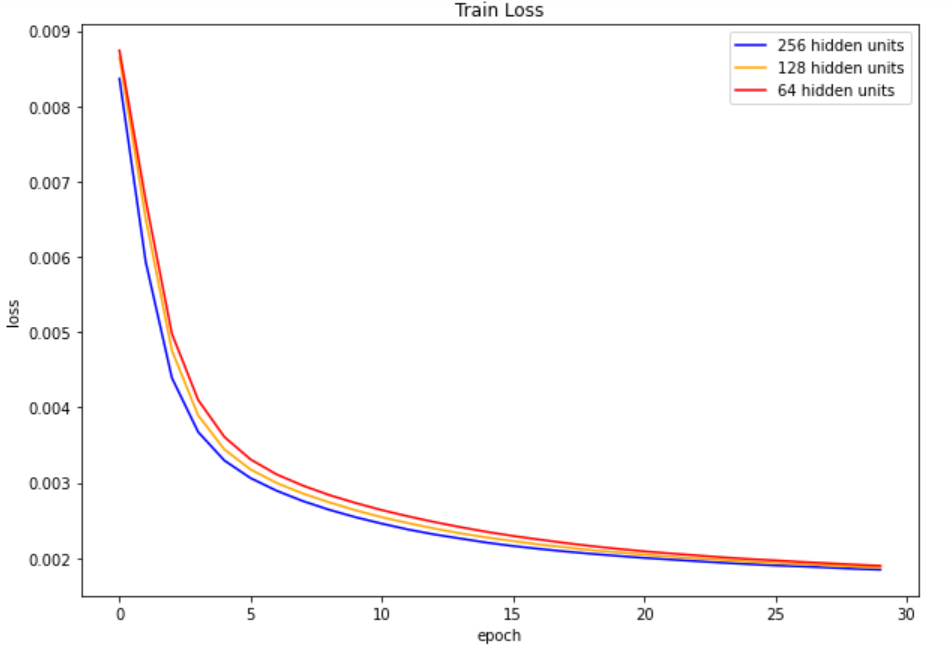


图5.4.5 不同隐藏单元个数Train\_Loss结果对比图

6、不同隐藏单元个数Test\_Loss结果对比图：256隐藏单元个数的loss最低，其次是128隐藏单元个数，64隐藏单元个数的loss最高。但是以大概数值来看，三个隐藏单元个数的loss都差不多，且都在逐步下降并趋于稳定

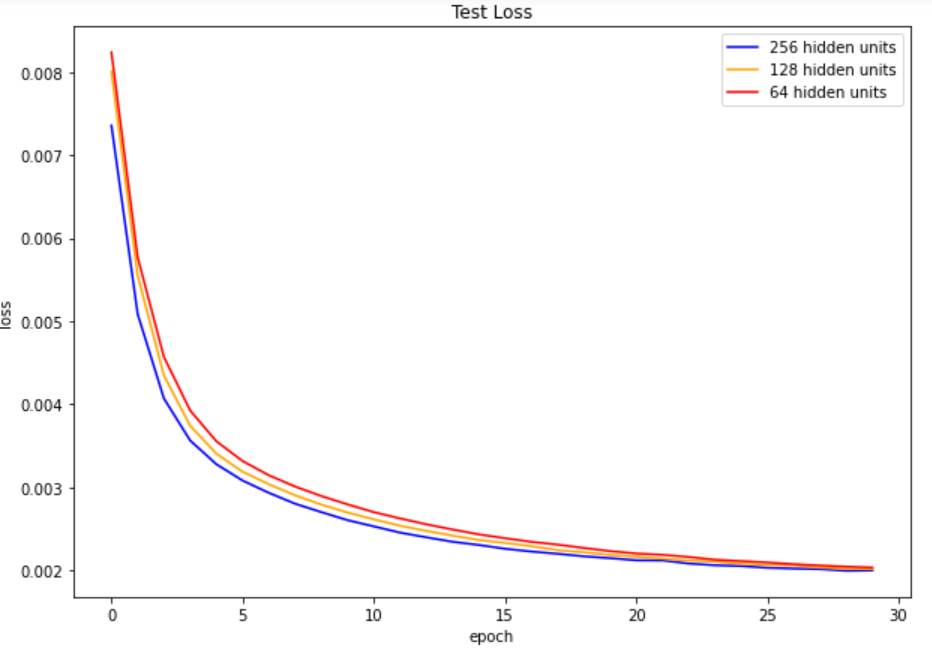


图5.4.6不同隐藏单元个数Test\_Loss结果对比图

7、不同隐藏单元个数Train\_Acc结果对比图：256隐藏单元个数的acc最高，一开始128隐藏单元个数的acc要低于64的，但是在第1个epoch之后就高于64隐藏单元个数了。在30个epoch之后，三个隐藏单元个数的acc相差无几，都在0.83左右

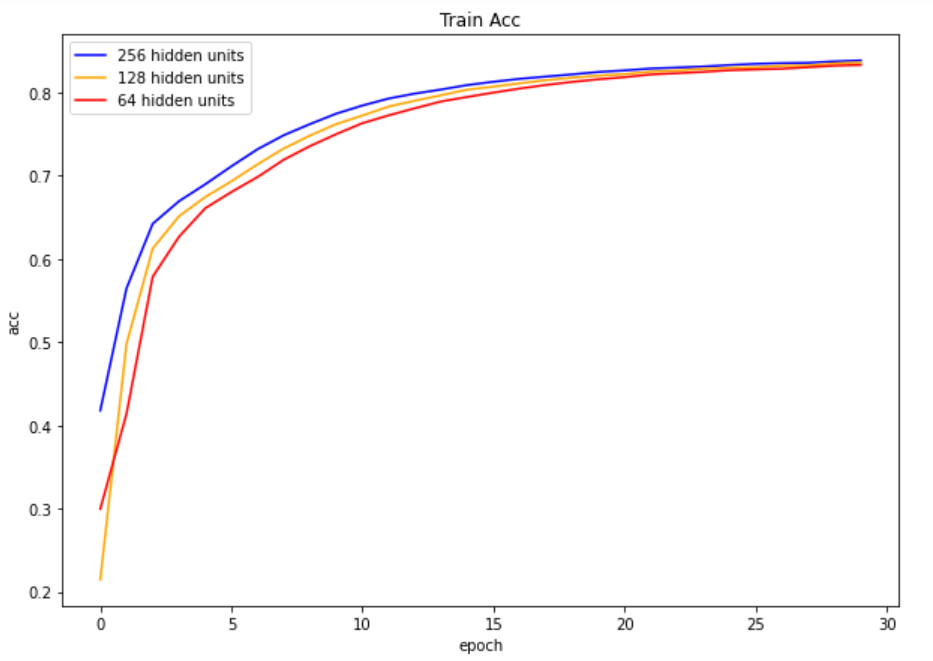


图5.4.7 不同隐藏单元个数Train\_Acc结果对比图

8、不同隐藏单元个数Test\_Acc结果对比图：256隐藏单元个数的acc最高，一开始128隐藏单元个数的acc要低于64的，但是在第1个epoch之后就高于64隐藏单元个数了。在30个epoch之后，三个隐藏单元个数的acc相差无几，都在0.82左右

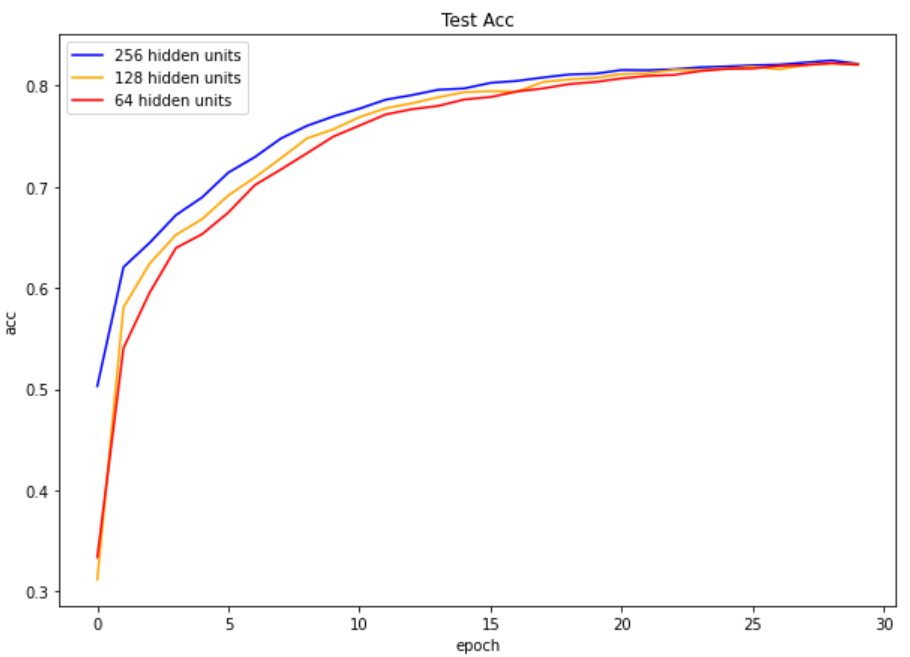


图5.4.8 不同隐藏单元个数Test\_Acc结果对比图

**5.5 在多分类基础上实现Dropout**

1、手动实现Dropout的Train\_Loss结果图：一开始的loss都差不多，不使用Dropout和Dropout=0.2的loss在100个epoch之后是最低的，Dropout=0.5的loss是其次，Dropout=0.7的loss最高。可以得出适当的Dropout可以防止过拟合，Dropout太高也不利于拟合模型

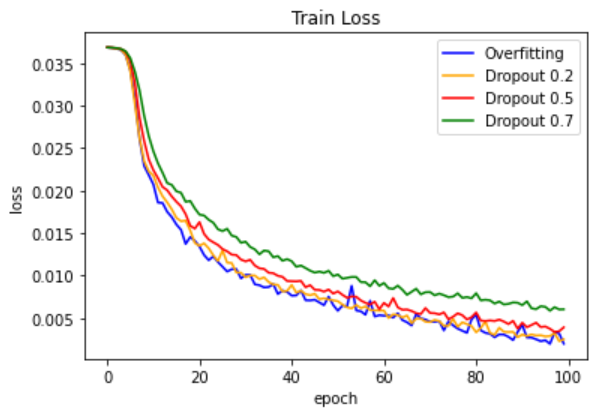


图5.5.1 手动实现Dropout的Train\_Loss结果图

2、手动实现Dropout的Test\_Loss结果图：一开始的loss都差不多，不使用Dropout和Dropout=0.2的loss不太稳定。从最后40个epoch来看，Dropout=0.5和Dropout=0.7的loss比较低也比较稳定，效果较好。

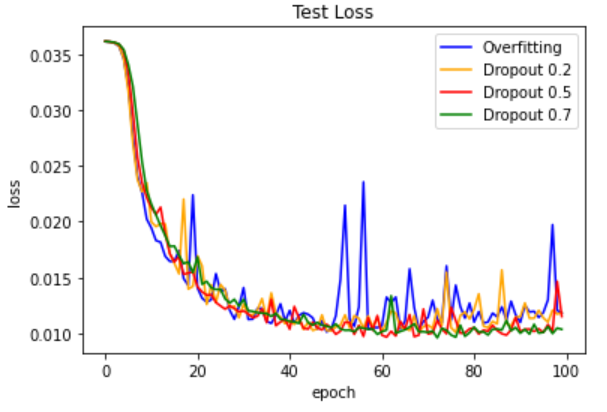


图5.5.2 手动实现Dropout的Test\_Loss结果图

3、手动实现Dropout的Train\_Acc结果图：一开始的acc都差不多，不使用Dropout和Dropout=0.2的acc在100个epoch之后是最高的，Dropout=0.5的loss是其次，Dropout=0.7的acc最低。Dropout太高准确率反而会下降，适当的Dropout可以增加准确率

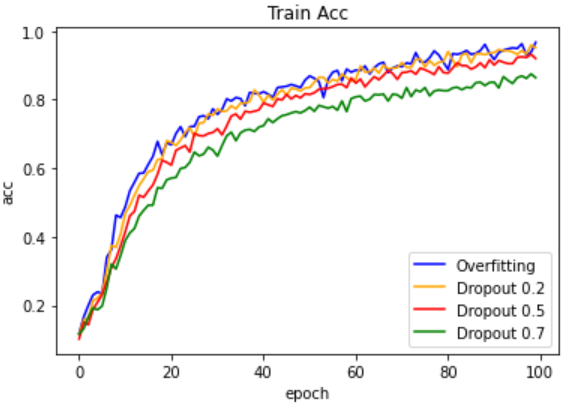


图5.5.3 手动实现Dropout的Train\_Acc结果图

4、手动实现Dropout的Test\_Acc结果图：一开始的acc都差不多，Dropout=0.5和Dropout=0.7的acc在100个epoch之后是最高的，不使用Dropout的acc不太稳定，Dropout=0.2的acc有一点不稳定，但是最终四种Dropout的acc都在0.8左右

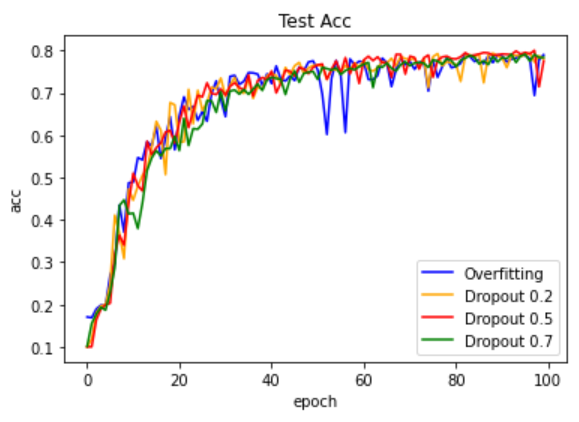


图5.5.4 手动实现Dropout的Test\_Acc结果图

5、利用torch.nn实现Dropout的Train\_Loss结果图：一开始的loss都差不多，不使用Dropout和Dropout=0.2的loss在100个epoch之后是最低的，Dropout=0.5的loss是其次，Dropout=0.7的loss最高。可以得出适当的Dropout可以防止过拟合，Dropout太高也不利于拟合模型

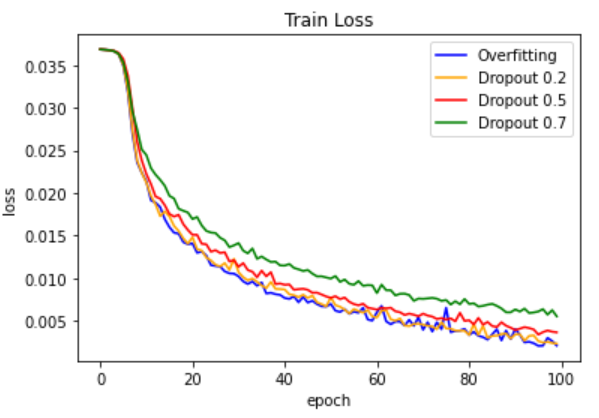


图5.5.5 利用torch.nn实现Dropout的Train\_Loss结果图

6、利用torch.nn实现Dropout的Test\_Loss结果图：一开始的loss都差不多，不使用Dropout和Dropout=0.2的loss一直不太稳定，涨幅比较大。从最后40个epoch来看，Dropout=0.5和Dropout=0.7的loss比较低也比较稳定，效果较好。

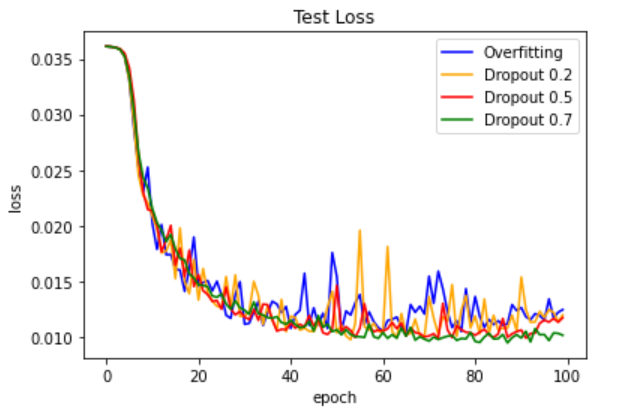


图5.5.6 利用torch.nn实现Dropout的Test\_Loss结果图

7、利用torch.nn实现Dropout的Train\_Acc结果图：一开始的acc都差不多，不使用Dropout和Dropout=0.2的acc在100个epoch之后是最高的，Dropout=0.5的loss是其次，Dropout=0.7的acc最低。Dropout太高准确率反而会下降，适当的Dropout可以增加准确率

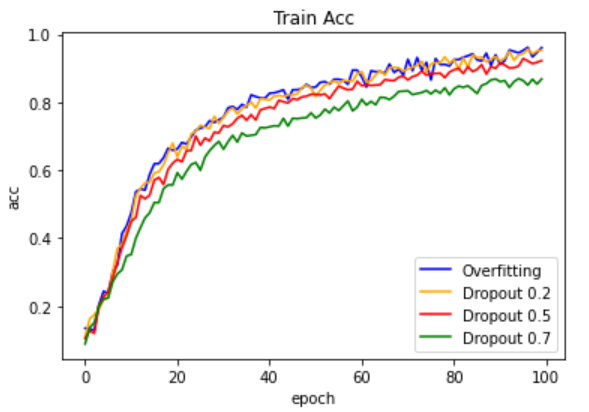


图5.5.7 利用torch.nn实现Dropout的Train\_Acc结果图

8、利用torch.nn实现Dropout的Test\_Acc结果图：一开始的acc都差不多，Dropout=0.5和Dropout=0.7的acc在100个epoch之后是最高的，不使用Dropout的acc不太稳定，Dropout=0.2的acc也有不稳定，但是最终四种Dropout的acc都在0.79左右

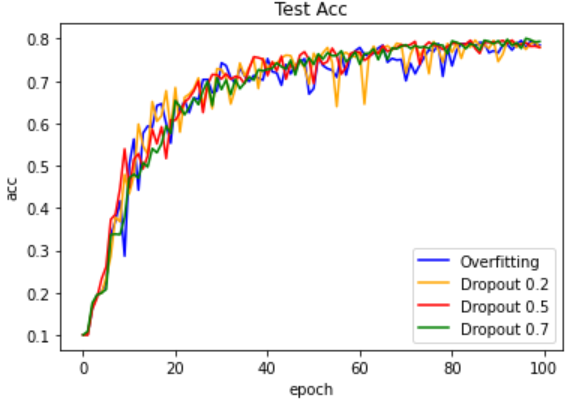


图5.5.8 利用torch.nn实现Dropout的Test\_Acc结果图

**5.6 在多分类基础上实现L2正则化**

1、手动实现L2正则化参数范式结果图：可以看到参数范式随着惩罚权重lambd的增加而变小，一定程度的缓解了过拟合，参数更接近0。

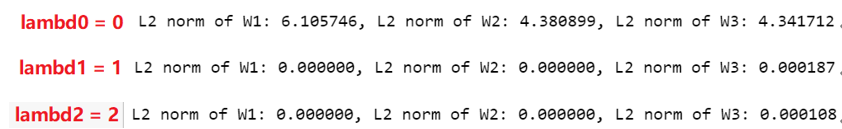


图5.6.1 手动实现L2正则化参数范式结果图

2、手动实现L2正则化惩罚权重等于0结果图：可以看到有明显过拟合的迹象，Train\_Loss一直在降低，但是Test\_Loss并不稳定

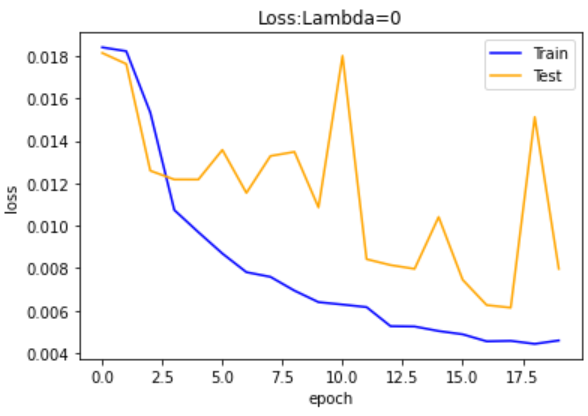


图5.6.2 手动实现L2正则化惩罚权重等于0结果图

3、手动实现L2正则化惩罚权重等于1结果图：一定程度上缓解了过拟合现象，Train\_Loss在第1个epoch后趋于稳定， Test\_Loss一直保持低且稳定

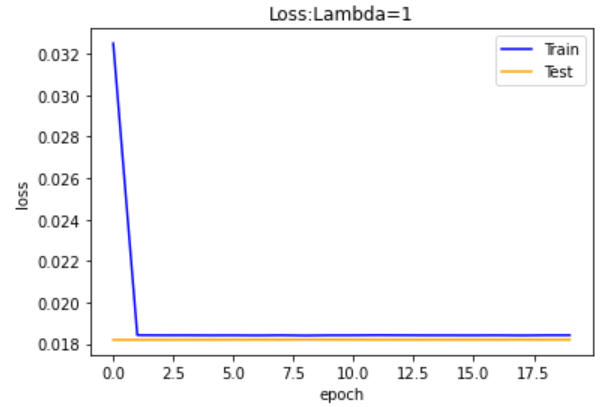


图5.6.3 手动实现L2正则化惩罚权重等于1结果图

4、手动实现L2正则化惩罚权重等于2结果图：一定程度上缓解了过拟合现象，Train\_Loss在第1个epoch后趋于稳定， Test\_Loss一直保持低且稳定

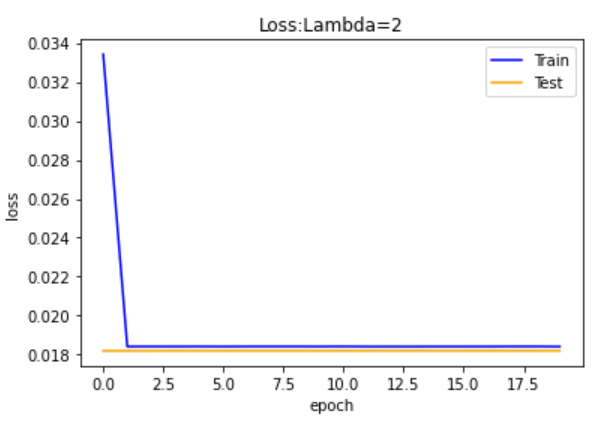


图5.6.4 手动实现L2正则化惩罚权重等于2结果图

5、利用torch.nn实现L2正则化参数范式结果图：可以看到参数范式随着惩罚权重lambd的增加而变小，一定程度的缓解了过拟合，参数更接近0。

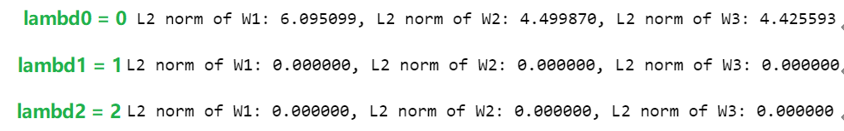


图5.6.5 利用torch.nn实现L2正则化参数范式结果图

2、利用torch.nn实现L2正则化惩罚权重等于0结果图：可以看到有明显过拟合的迹象，Train\_Loss一直在降低，但是Test\_Loss并不稳定

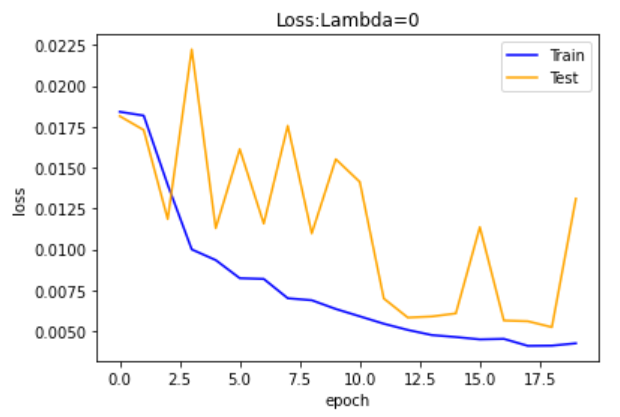


图5.6.2 利用torch.nn实现L2正则化惩罚权重等于0结果图

3、利用torch.nn实现L2正则化惩罚权重等于1结果图：过拟合现象十分严重，Train\_Loss和Test\_Loss整体趋于上升，且Test\_Loss不稳定

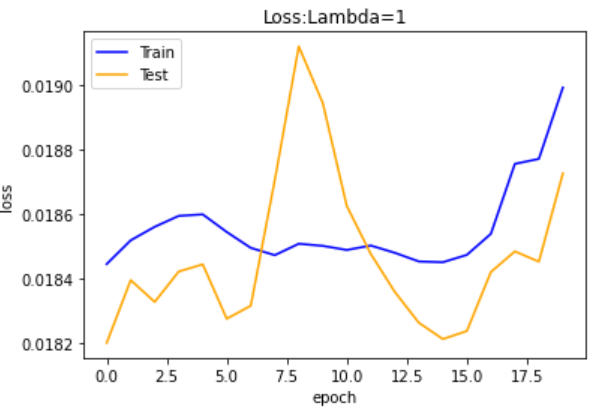


图5.6.3 利用torch.nn实现L2正则化惩罚权重等于1结果图

4、利用torch.nn实现L2正则化惩罚权重等于2结果图：一定程度上缓解了过拟合现象，Train\_Loss一直很稳定，Test\_Loss低于Train\_Loss，途中有几个epoch不稳定，然后最后趋于稳定

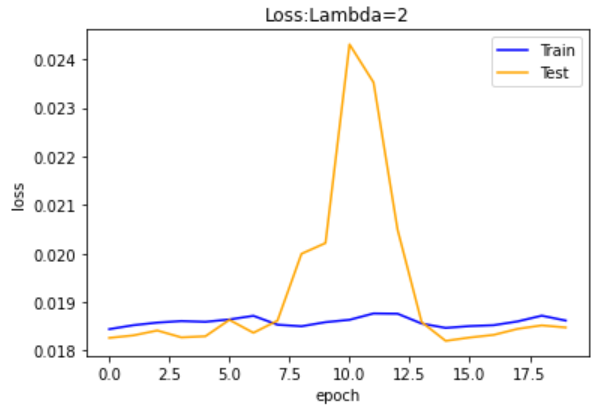


图5.6.4 利用torch.nn实现L2正则化惩罚权重等于2结果图

**5.7 实现十折交叉验证**

1、回归任务结果图：每一折的train\_loss和valid\_loss都差不多，average train loss是0.000688，average valid loss是0.000698

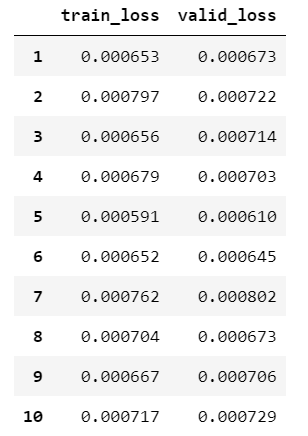


图5.7.1 回归任务结果图

2、二分类任务结果图：二分类效果比较好，train\_acc和valid\_acc在每一折上都是100%，average train loss是0.000900，average valid loss是0.001042

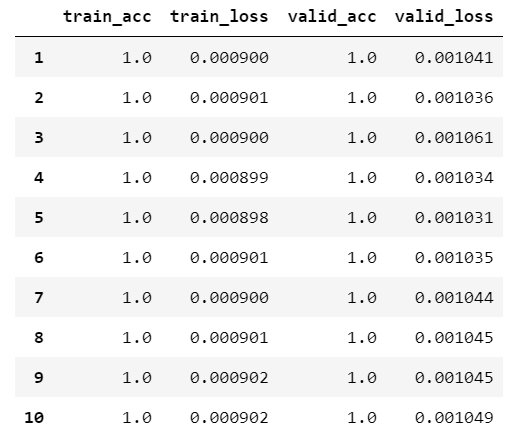


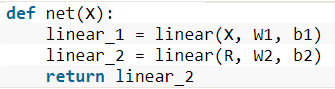
图5.7.2 二分类任务结果图

3、多分类任务结果图：多分类的最终效果不太好，average train loss为0.997489, average train accuracy为0.656200，average valid loss为1.001231, average valid accuracy为0.653500。因为迭代的epoch比较低，只有10，所以loss没有降得很多，acc也不是很高。



图5.7.3 多分类任务结果图

## 六、实验心得体会

1. 实现回归问题时由于是需要实现线性函数，所以不需要加入激活函数（解决非线性问题），实验中也没有加入隐藏层。如果需要加入隐藏层，则可以写成，由线性函数可知，求得的w=w2w1，b=w2b1+b2
2. 对于二分类问题的激活函数使用Sigmoid，损失函数使用二分类交叉熵损失函数。Sigmoid函数用作二分类问题的意义：输出结果只有一个，如果值在0.5-1之间，表示分为正类；如果值在0-0.5之间，表示分为负例。Softmax函数用作n分类问题的意义：输出结果有n个，这n个标签的概率加起来为1，哪个标签概率高就分为哪一类。当n=2时，Sigmoid函数与Softmax函数相同。
3. 如果使用BCEWithLogitsLoss（用于二分类问题）作为损失函数，那就不需要Sigmoid层了，因为BCEWithLogitsLoss = BCELoss + Sigmoid
4. 多分类问题激活函数使用Softmax，损失函数使用多分类交叉熵损失函数。如果使用CrossEntropyLoss（用于多分类问题）作为损失函数，那就不需要Softmax层了，因为CrossEntropyLoss= NLLLoss+ Softmax
5. 隐藏层层数问题
6. 没有隐藏层：只能够表示线性可分的函数
7. 隐藏层数=1：可以拟合任何“包含从一个有限空间到另一个有限空间的连续映射”的函数
8. 隐藏层数=2：可以表示任意精度的任意决策边界，并且可以拟合任意精度的任何平滑映射
9. 隐藏层数>2：可以学习复杂描述
10. 隐藏层单元个数问题
11. 如果单元个数过少，网络不能具有必要的学习能力和信息处理能力
12. 如果单元个数过多，不仅会大大增加网络结构的复杂性，网络在学习过程中更易陷入局部极小点，而且会使网络的学习速度变得很慢
13. L2正则化的目的就是为了让权重衰减到更小的值，在一定程度上减少模型过拟合的问题，所以权重衰减也叫L2正则化。

## 七、参考文献

PyTorch版《动手学深度学习》

## 八、附录

略。

**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。