基于 Numpy 的手写数字分类 MLP

1. 数据预处理

对于原始灰度图像素 [0, 255]，将其零中心归一化为 [-0.5, 0.5]。实验表明多数情况下，归一化有助于提高模型精度。

```
PIXEL_DEPTH = 255
data = (data - (PIXEL_DEPTH / 2.0)) / PIXEL_DEPTH
```

2. 模型建立

以下实验中使用的模型为一个四层的 MLP，神经元数分别为128-64-64-10。在前三层神经元后使用了 sigmoid 激活函数，在最后一层神经元后使用 softmax 获取模型分类置信度向量。该模型架构仅在为证明MLP实现正确，在本作业中并未探究模型深度与宽度对最优准确率的影响。

此外，为研究正则化方法对模型的影响，本作业中也实现了 dropout 层的正向与反向传播，并把 dropout 层添加在前三层神经元和激活函数之间。

关于超参数：本作业设置初始学习率为 0.02，batchsize为 16，回合数为10。

3. 对比不同的参数初始化方法

本作业实现了三种参数初始化方法：

1. Zero：将线性层的所有参数（weight & bias）均初始化为0。
2. Normal：根据均值为 0，标准差为 0.02 的高斯分布随机初始化线性层所有参数。
3. Xavier：Xavier 随机初始化线性层所有参数。Xavier 初始化的基本思想是，若单层网络的输入和输出可以保持正态分布且方差相近，这样就可以避免输出趋向于 0，从而避免梯度弥散情况。 具体实现方法为：

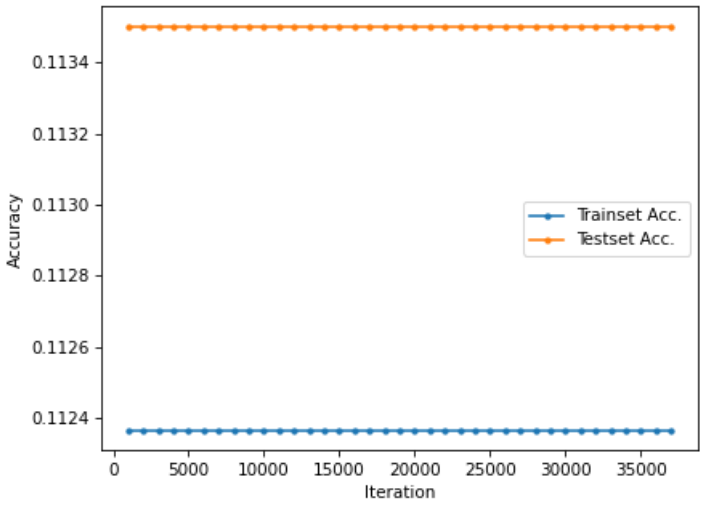
```
std = np.sqrt(2. / (self.in_channel + self.out_channel))
self.weight = np.random.normal(loc=0., scale=std, size=[self.in_channel,
self.out_channel])
self.bias = np.random.normal(loc=0., scale=std, size=[self.out_channel])
```

实验结果如下：

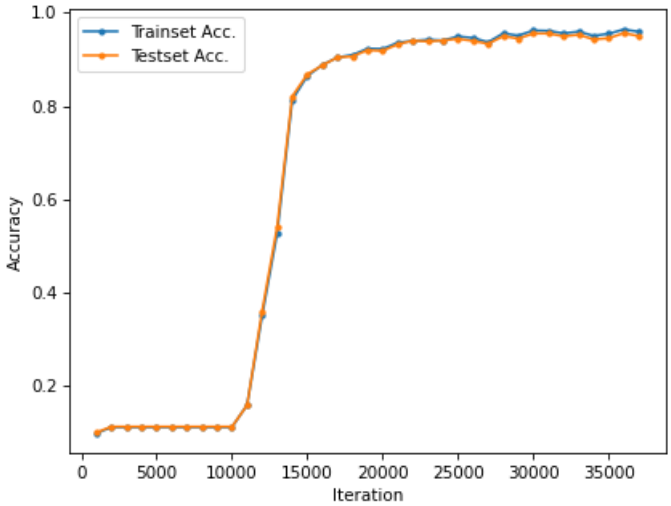
初始化	损失函数	学习率调整	正则化方法	训练集最终准Acc.	测试集最终Acc.	测试集最优Acc.
Zero	交叉熵	Const.	无	-	-	-
Normal	交叉熵	Const.	无	0.959	0.950	0.956
Xavier	交叉熵	Const.	无	0.980	0.969	0.974

训练集与测试集的准确率曲线如下：

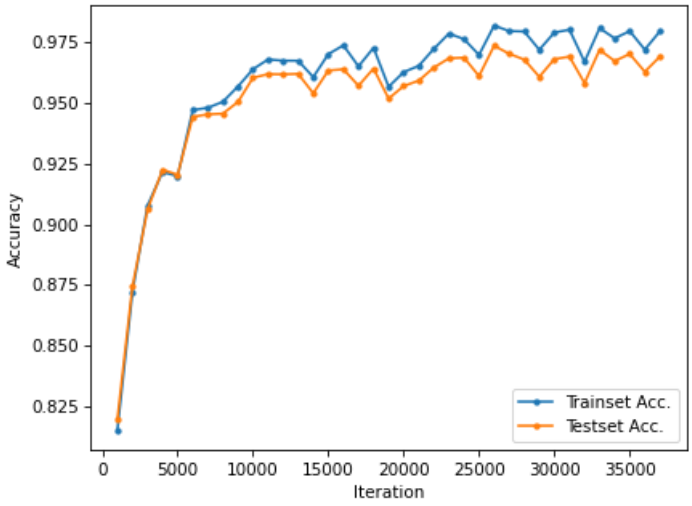
Zero初始化：



Normal初始化：



Xavier初始化：



实验发现：

1. Zero 初始化是失败的。如图所示，模型准确率随迭代次数增加保持不变。这归因于模型在 forward 和 backward 时，activation 和 gradient 都是相同的数值，所以模型对任意输入只会预测相同的结果。
2. Xavier 初始化可以加快模型收敛。如图所示，在第一次测试模型精度时就已经达到 > 0.8 的准确率。
3. Xavier 初始化可以得到更高的精度。

4. 对比不同的损失函数

本作业实现了交叉熵损失函数。在模型存在 softmax 的基础上，根据公式推导（参考：https://blog.csdn.net/qian99/article/details/78046329），梯度可以化简为 “激活值 - 真值”。

5. 对比不同的学习率调整方法

本作业实现了四种学习率调整方法：

1. Constant：学习率恒定为 0.02。
2. Step：计算出总迭代次数后，每迭代 1/3 就将学习率降低为原来的 0.1 倍。
3. Cosine：使得学习率以余弦函数型变化。具体实现如下：

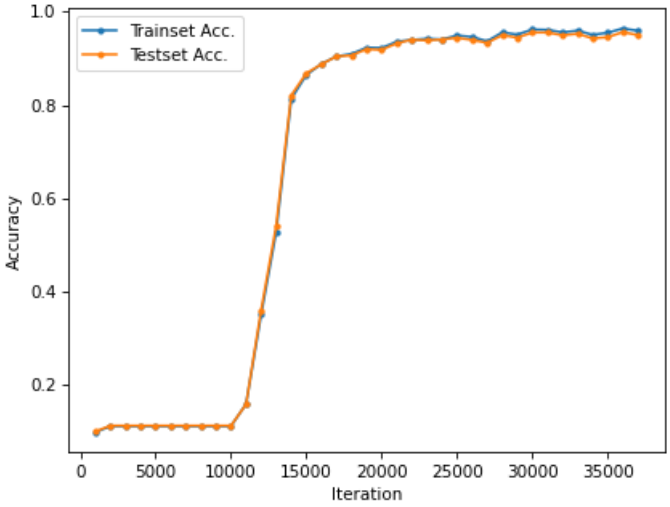
```
self.lr = 0.5 * (1 + math.cos(math.pi * iter / self.total_iter)) * self.init_lr
```

4. Cycle： 设定一个学习率上限和下限，迭代时学习率的值在上限和下限的区间里周期性地变化。具体实现如下：

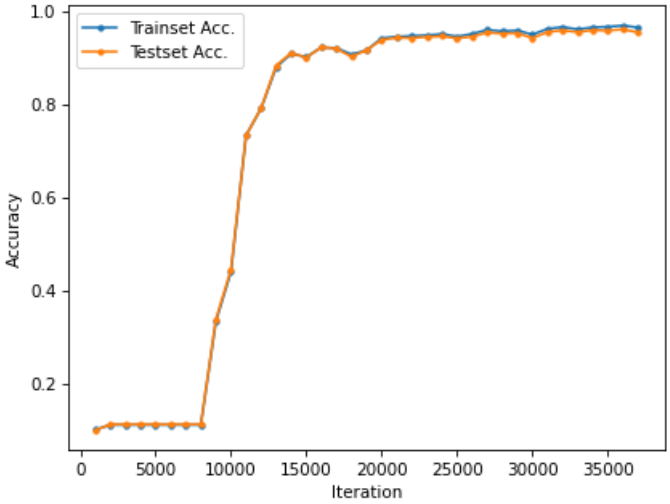
```
max_lr = self.init_lr
base_lr = self.init_lr * 0.01
cycle = np.floor(1 + iter / (2 * stepsize))
x = np.abs(iter / stepsize - 2 * cycle + 1)
self.lr = base_lr + (max_lr - base_lr) * np.maximum(0, (1-x))
```

初始化	损失函数	学习率调整	正则化方法	训练集最终准 Acc.	测试集最终 Acc.	测试集最优 Acc.
Normal	交叉熵	Const.	无	0.959	0.950	0.956
Normal	交叉熵	Step	无	0.970	0.955	0.962
Normal	交叉熵	Cosine	无	0.967	0.955	0.959
Normal	交叉熵	Cycle	无	0.941	0.937	0.939

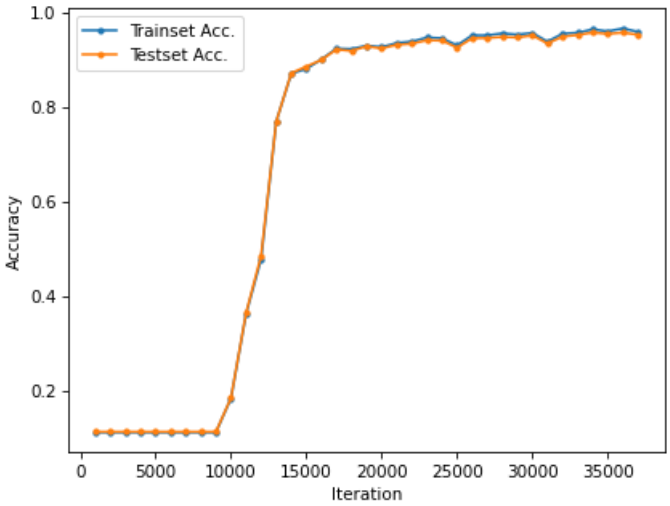
Const学习率策略：



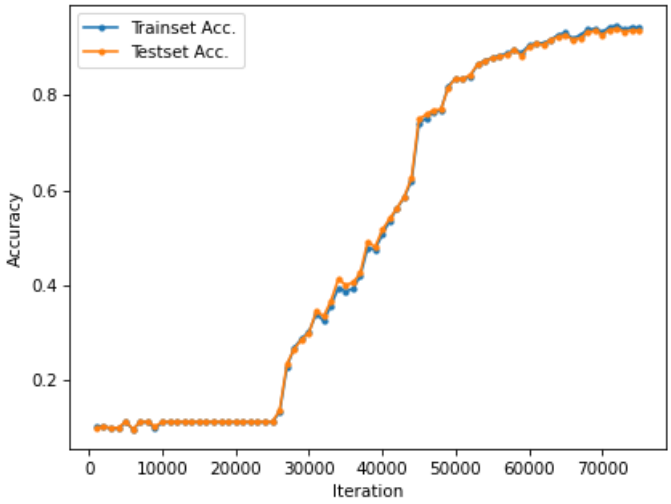
Step学习率策略：



Cosine学习率策略：



Cycle学习率策略：



实现发现：

1. Step 学习策略获得了最优的效果。这是因为在训练后期，更小的学习率可以让模型进一步趋近最优解。
2. Step 学习策略还可以加速模型收敛。
3. Cycle 学习策略由于存在学习率循环周期、最大最小学习率等超参数，调参较困难，在收敛速度和最终模型性能上都低于 baseline。

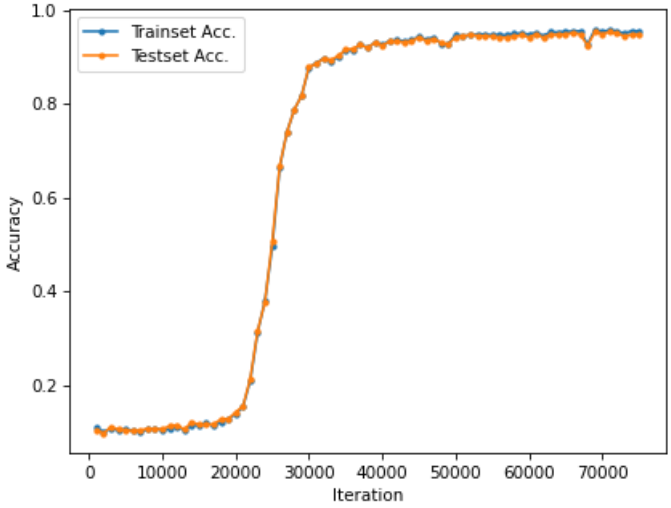
6. 对比不同的正则化方法

本作业实现了两种正则化方法：

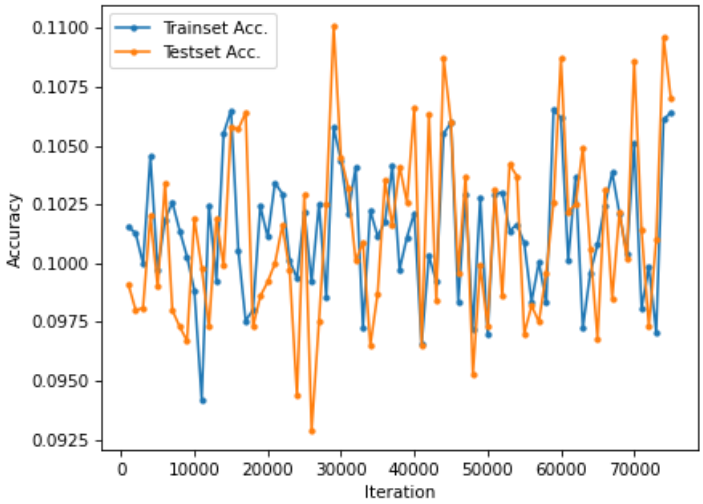
1. Dropout：随机置零一些神经元的 activation，用一个与 activation 相同大小的二值 mask 张量记录 dropout 的值用于正向与反向传播。
2. L2 正则化（权重衰减）

初始化	损失函数	学习率调整	正则化方法	训练集最终准Acc.	测试集最终Acc.	测试集最优Acc.
Normal	交叉熵	Cosine	无	0.967	0.955	0.959
Normal	交叉熵	Cosine	dropout(0.2)	0.958	0.949	0.954
Normal	交叉熵	Cosine	dropout(0.5)	-	-	-
Normal	交叉熵	Cosine	L2(1e-5)	0.984	0.964	0.971
Normal	交叉熵	Cosine	L2(1e-7)	0.983	0.966	0.967

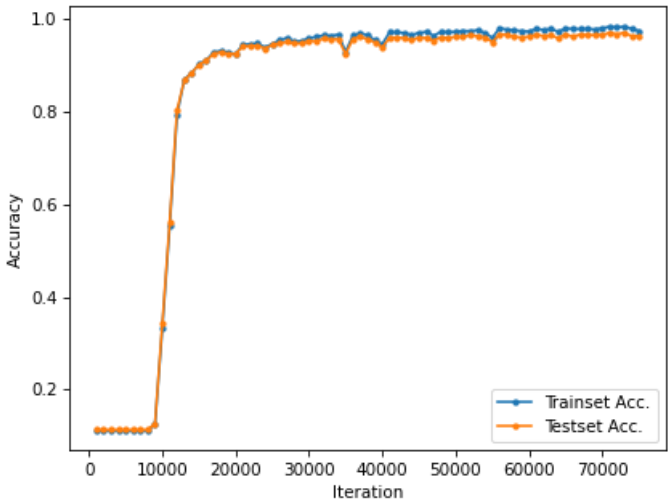
Dropout(rate=0.2)



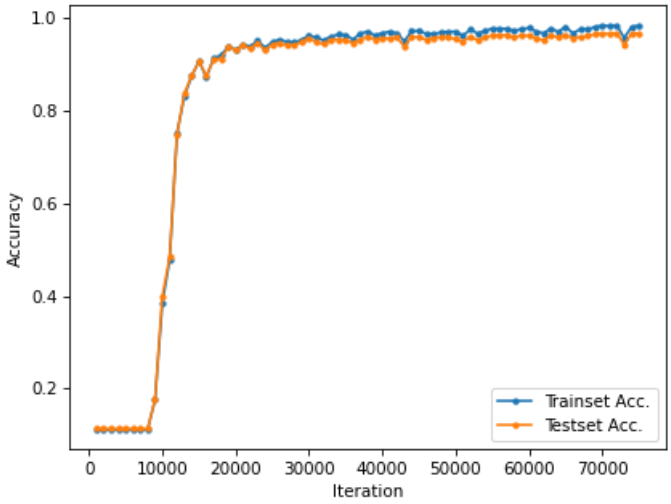
Dropout(rate=0.5)



L2 正则化 (weight decay = 1e-5)



L2 正则化 (weight decay = 1e-7)



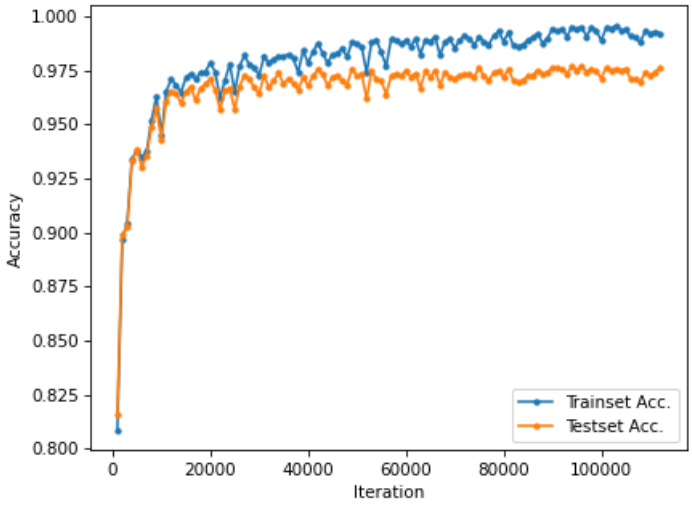
实验发现：

1. 添加正则化方法（dropout，L2）后模型收敛变慢，迭代相同次数后模型准确率低于 baseline。所以，本节涉及的模型都训练了 20 个回合。
2. 添加 L2 正则化方法后模型准确率增加。
3. 添加 dropout 的模型准确率低于 baseline。这说明当前模型的计算复杂度（模型capacity）与当前 MNIST 分类任务数据集的体量相当，添加 dropout 后会使模型出现不同程度的欠拟合。当 dropout rate 过大（比如 0.5）时，模型几乎无法学习到“知识”。

7. 最优识别结果

根据以上实验结果，组合初始化，损失函数，学习率调整，正则化方法的最优策略，得到最优的识别结果：

初始化	损失函数	学习率调整	正则化方法	训练集最终准Acc.	测试集最终Acc.	测试集最优Acc.
Xavier	交叉熵	Step	L2(1e-5)	0.992	0.976	0.977



本次作业全部用 Numpy 实现。

请把 MNIST 数据保存在 `raw_data` 文件夹中。

运行代码的命令是：

```
python main.py
```

如要调整模型的超参数请修改 `main.py` 的 `get_config` 函数。