手写数字识别实验报告

191300008 陈彦全 AI

实验内容以及阶段性实验成果展示

手动加载MNIST

- 下载MNIST训练集与测试集并针对.gz格式文件进行解压之后保存在./MNIST_data文件夹下，然后着手进行数据读取的相关工作。
- 在实验中，数据读取的代码实现文件为./load_data.py
- ./load_data.中实现了对MNIST数据集的解码函数decode(),加载数据的主函数load_data(),测试函数test(),方便返回值对应的函数generate_object()。具体实现见代码以及注释。
    - decode()函数针对http://yann.lecun.com/exdb/mnist/链接中MNIST官方给出的文件结构(示例如下)，利用python标准库中的struct进行解码。

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

    - test()函数使用的是torchvision库中的标准加载方式，用来确认MNIST数据集加载无误。导入torchvision库在test()函数中进行，避免不需要test()的情况下导入torchvision。
    - generate_object()用来对指定文件返回作用对象，例如对于t10k-images-idx3-ubyte文件，我们解析的对象为test_x。
    - load_data()利用上述函数，返回正确的train_x,train_y,test_x,test_y

Dataloader

- 在./dataloader.py中实现了一个简单的基于batch的dataloader函数Dataloader(x, y, batch_size, shuffle=True),利用yield生成器，每次生成x,y中batch_size大小的数据。

Layer

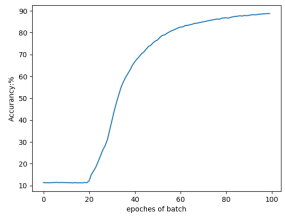
- 在./layer.py中实现了一个基本的linear layer以及ReLU激活层，并用can_updata参数来区别linear layer、ReLU激活层的参数更新过程，ReLU激活层不需要参数更新，linear layer需要根据计算的梯度进行参数更新。
- 在linear layer以及ReLU激活层中都实现了forward前向传播以及backward反向传播的过程，前向传播都是利用上一层的输出作为输入按照该层计算式进行计算，反向传播过程以后一层传来的梯度grad，累乘上该层的梯度作为结果返回，即grad*grad{linear layer/ReLU}。
- 特别的，linear layer中带有update方法，根据梯度计算的delta_w/delta_b更新权重w、偏置b,更新方式采用平凡的learning_rate * delta_w/delta_b的方案。

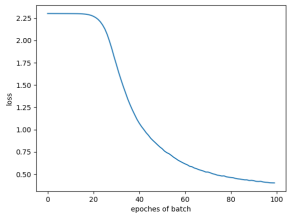
Loss Function

- 在./loss_function中实现了各种损失函数的forward（loss计算）、backward（梯度计算）。

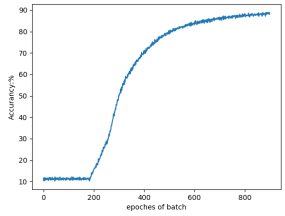
简单实现的双层layer的MLP以及baseline实验结果

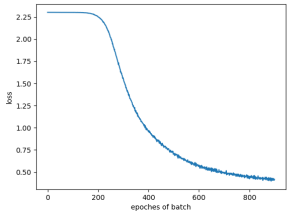
- 在./network.py中实现了一个网络的初始结构，记录当前网络中的各种layer以及layer数，可以通过append类方法增加一层网络结构（Linear Layer或者ReLu激活层）；forward类方法组织各层的forward过程，按照输入层到输出层的顺序执行各层的forward；backward类方法组织各层的backward过程，按照输出层到输入层的顺序执行各层的backward；update类方法按照输入层到输出层的顺序检索can_update为True的layer进行参数更新。
- 在./main.py中的Multiperception()函数实现了MLP网络model，./train.py实现了训练函数，./test.py实现了验证集的验证函数。
- 按照baseline的要求，固定学习率为1e-3,并将参数进行随机初始化，使用softmax处理模型输出并用交叉熵函数作为损失函数,不加入正则化layer。该参数设置的MLP模型，在128batch大小的训练集下，在以batch化训练100轮之后，得到的实验结果如下(实验绘图相关代码均在./plot.py中进行实现，实验结果会保存在./pics文件夹下):
    - 测试集结果: 最终正确率：0.8870648734177216





    - 训练集结果: 最终正确率：0.880625

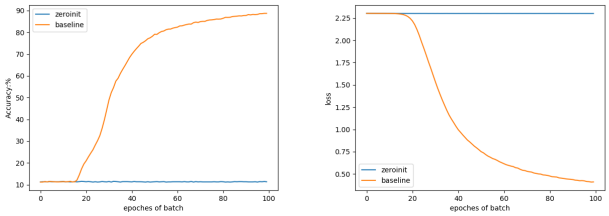




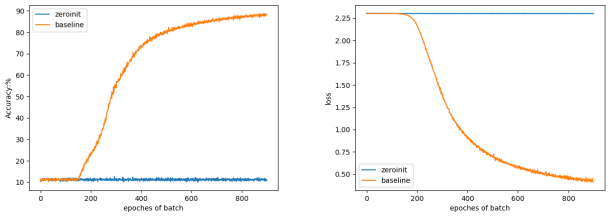
- 实现baseline中的小插曲:
    - 在训练的过程中，一开始频繁出现loss为Nan的情况，查阅相关资料，得知该情况出现在网络权重过大或者softmax没有进行数值稳定处理，因此，在参数随机初始化的过程中,需要传入一个init_weight参数0.001,调小权重的绝对值。
    - 另外，由于一开始设定的学习率为1e-6,batch大小为64，在解决loss为Nan的问题之后，loss仍然无法正确收敛，始终在2.3处徘徊，通过查阅资料，得知可能与学习率大小、batch大小之间的匹配程度有关，最终通过调整参数，发现之前说明的固定的学习率1e-3以及batch大小128的参数条件下，交叉熵loss能够正常收敛。
        - 实验后期发现学习率再增大一些，设定为1e-1左右，模型性能会巨幅提升，之前对于学习率的认识太保守了。

不同参数初始化方案的实验结果对比

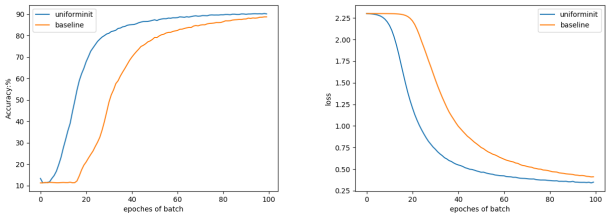
- 以下的实验结果除明说参数变更之外，均与baseline参数保持一致。初始化方法的具体实现参见./layer.py
- 全零初始化:
    - 实验结果如下:
        - 测试集结果：最终正确率:0.11362737341772151



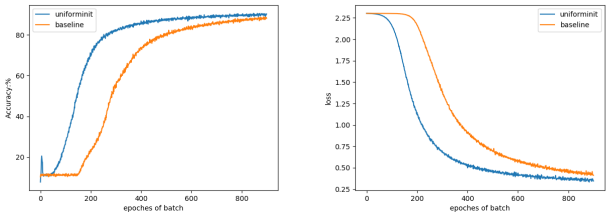
- 训练集结果：最终正确率:0.10875



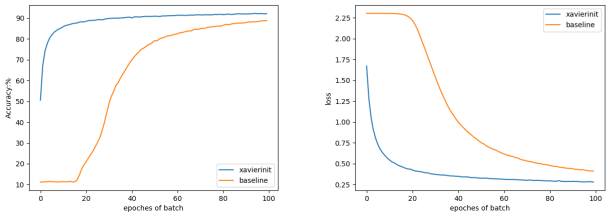
- 基于固定方差的参数初始化——采用均匀分布采样：
  - 实验结果如下：
    - 测试集结果：最终正确率:0.9010087025316456



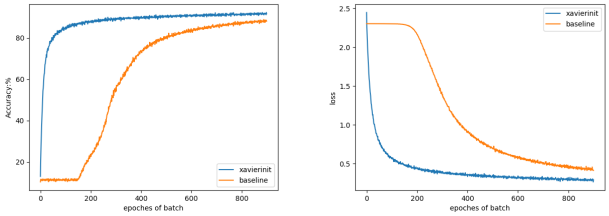
- 训练集结果：最终正确率:0.90203125



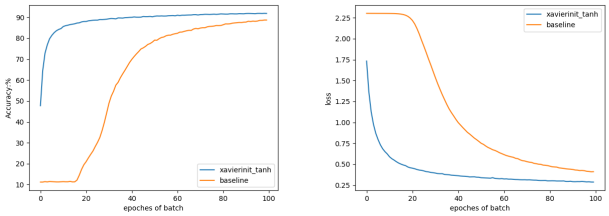
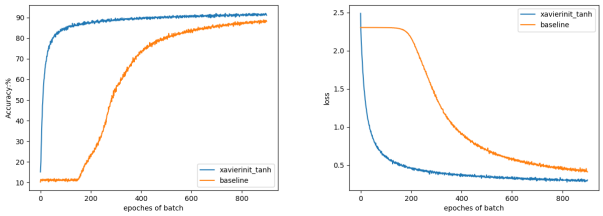
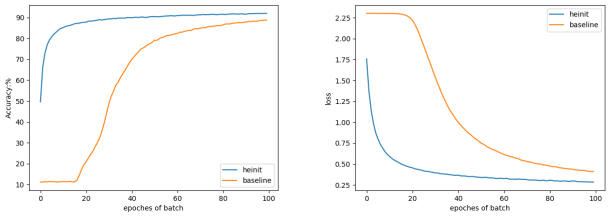
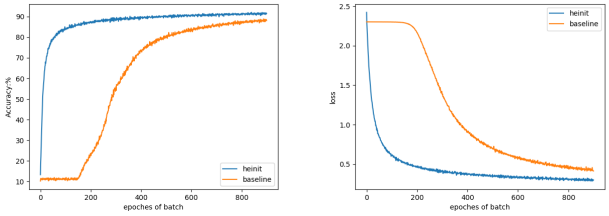
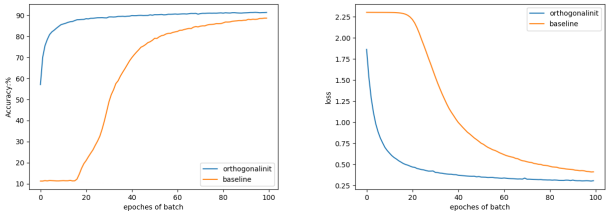
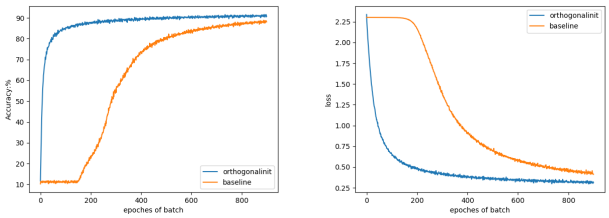
- Xavier 初始化：
  - 在使用与baseline一致的ReLU激活函数的情况下的实验结果如下：
    - 测试集结果：最终正确率:0.9208860759493671

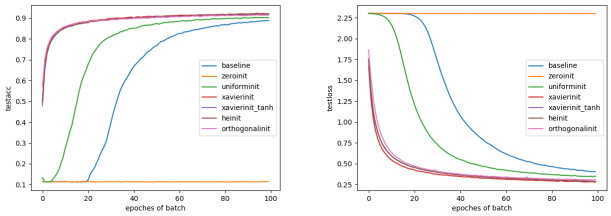


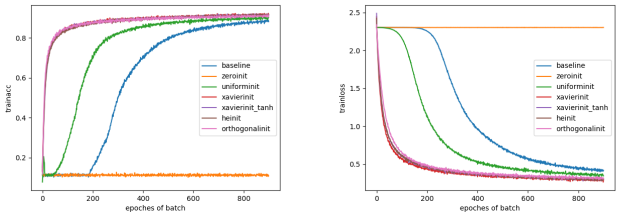
- 训练集结果：最终正确率:0.91703125



  - 查阅资料，得知Xavier 初始化与ReLU激活函数的相性不佳，因此考虑更换为tanh激活函数试一试。
  - 换用tanh激活函数的实验结果如下：

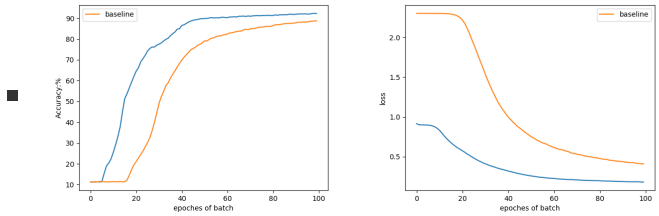
        - 测试集结果：最终正确率:0.9187104430379747
        - 
        - 训练集结果：最终正确率:0.9109375
        - 
    - 可能是因为实验中实现的MLP只有简单的一个隐藏层的原因，ReLu与tanh激活函数之间的劣势并没有体现出来。
- He初始化：
    - 实验结果如下：
        - 测试集结果：最终正确率:0.9192049050632911
        - 
        - 训练集结果：最终正确率:0.91546875
        - 
- 正交初始化：
    - 实验结果如下：
        - 测试集结果：最终正确率:0.9140625
        - 
        - 训练集结果：最终正确率:0.9078125
        - 

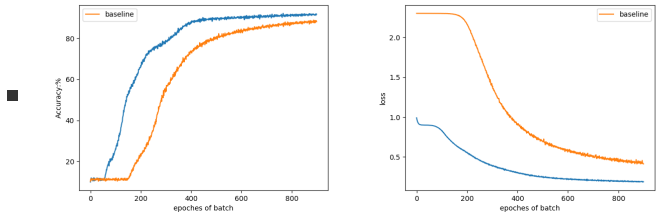
- 汇总对比结果：
  - 测试集结果：
    - 

  - 训练集结果：
    - 

  - 从上图对比可知，零初始化是无法进行正常训练的。而采取随机初始化的方法是这几种方法中除零初始化之外最为糟糕的一种，这也变相导致了在实验前期我需要调整学习率以及各种参数才能让随机初始化的MLP模型的loss函数收敛。而相比较于方差缩放的初始化方法，固定方差的初始化方案效果明显更差，这不仅反映在loss收敛速度，还反映在模型预测的正确率的增长上。正交初始化方案以及其余的基于方差缩放的初始化方案都没有明显的差别，但是都对baseline性能有巨大的提升。

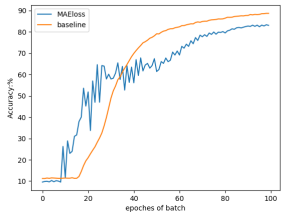
多种损失函数以及实验结果对比

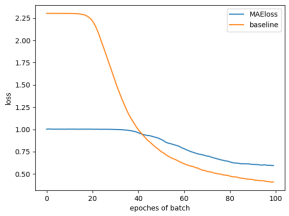
- 以下尝试的损失函数均只在baseline的参数设置基础上只改变损失函数。损失函数具体实现参见./loss_fuction.py.
- MSE loss :
  - 其error计算公式：
    - $loss = \frac{1}{2m}\sum_{i=1}^{m}(y' - y)^2$
  - 实验结果如下：
    - 测试集结果：最终正确率：0.9226661392405063
      - 

    - 训练集结果：最终正确率：0.91703125
      - 

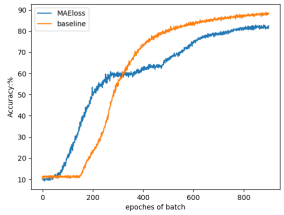
- MAE loss :
  - 其error计算公式：
    - $loss = \frac{1}{m}\sum_{i=1}^{m}|y' - y|$

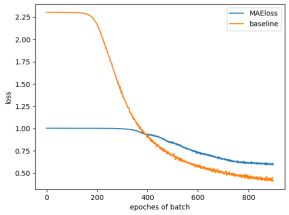
- 实验结果如下：
  - 测试集结果：最终正确率：0.8307950949367089





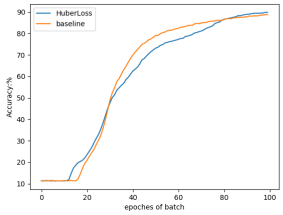
  - 训练集结果：最终正确率：0.8246875

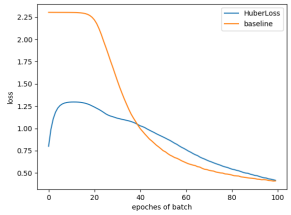




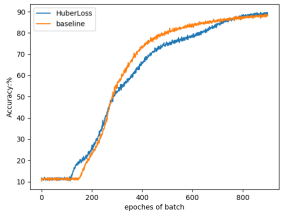
  - 可见MAEloss造成测试集的波动性还是挺明显的。

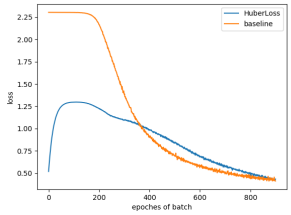
- Huber loss :
  - 参考HuberLoss — PyTorch 1.10.0 documentation
  - 其error计算公式：
    - $loss = \frac{1}{2}(y' - y)^2, if\ |y' - y| < delta$
    - $otherwise, loss = delta * (|y' - y| - 0.5 * delta)$
  - 在实验中，用pytoch中设定的默认值delta=1
  - 实验结果如下：
    - 测试集结果：最终正确率：0.8979430379746836



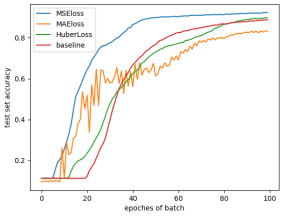


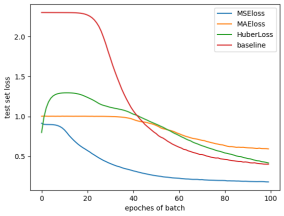
    - 训练集结果：最终正确率：0.884375

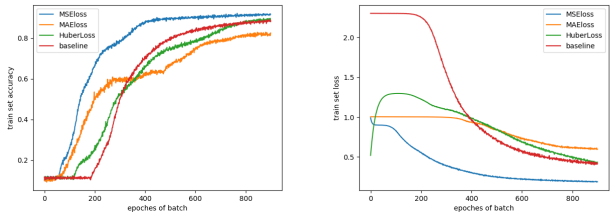




- 将实验中涉及的所有损失函数的结果合在一张图的对比图：
  - 测试集结果：
    -

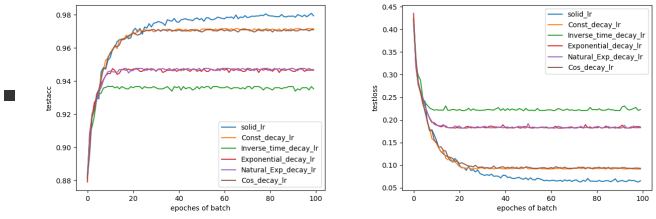


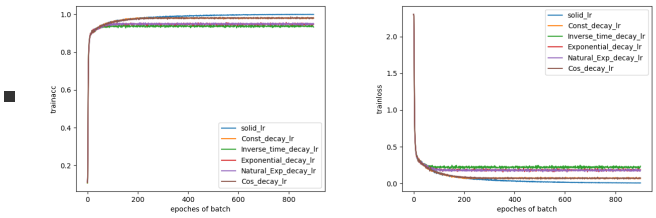


- 训练集结果：
  - 

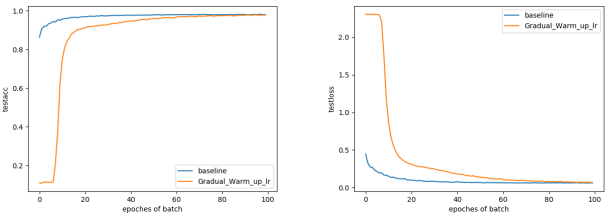
- 从上述结果可以看出，MAEloss的正确率波动性最大，在训练之后的正确率最低，在训练过程中，loss拟合速率也是最慢的；而MSEloss的正确率在训练过程中呈现一开始快速上升，随后缓慢增长的特点，这与作为baseline的交叉熵损失的特征比较类似，但是交叉熵损失在实验参数的情况下，开始收敛的速度很慢，寻找最优点的速率很慢；而作为MAEloss以及MSEloss改良版本的Huberloss，在正确率上，确实减少了MAELoss的波动性，又有MSEloss快速增长的特点，并且，在loss拟合的过程中也呈现了一条独特的收敛曲线，在训练限定的epoch大小之外，继续收敛的可能比MSEloss更大。

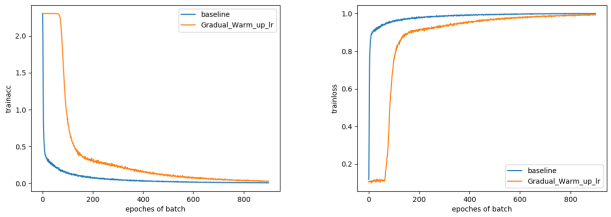
学习率调整方案以及实验结果对比

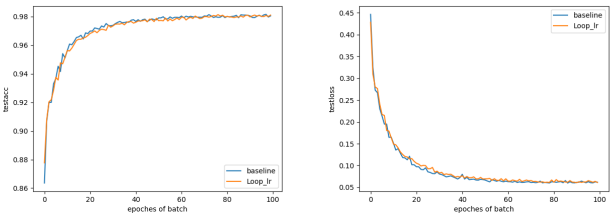
- 学习率调整方案的具体实现参见./lr_schedult.py。除了学习率方案调整之外，其余参数与baseline保持一致。
- 固定学习率衰减方案：
  - 对于固定学习率衰减，在./lr_schedult.py中简单实现了课程中提及的分段常数衰减、逆时衰减、指数衰减、自然指数衰减四种方法，这里对他们的结果直接进行对比展示，不再单独展示。
  - 由于一开始设定的学习率1e-3，在使用学习率衰减之后，实验效果不佳，有多个衰减方案在100轮内无法收敛，因此，这一部分设定的baseline的固定学习率为1e-1，并将重新进行baseline的实验。在下方的对比实验效果展示中，我们可以发现在设定学习率为1e-1之后的baseline的表现十分出色。
  - 实验结果如下：
    - 测试集结果：
      - 

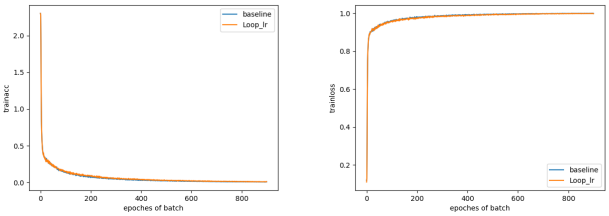
    - 训练集结果：
      - 

  - 实验结果发现，采取固定学习率为1e-1的baseline的效果是最好的，我认为大概是因为MLP模型在训练的后期仍然还处于继续拟合的阶段，导致学习率提前衰减并没有让模型更接近全局最优，反而让模型在走向全局最优的路上的速度减慢了。可能选择更大的学习率，其他学习率调整方案会有更大的用武之地。
- 逐渐预热的学习率方案：

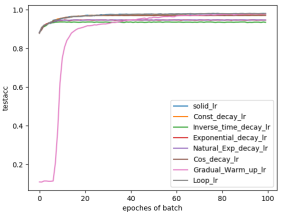
- 由于这个方案比较特殊，因此，在训练未开始之前先设定学习率为0，在训练的100轮过程中逐渐增长到固定学习率1e-1,且因为一般这个方案需要配合其他学习率衰减的方案混合使用，因此，单独使用该方案的效果可能不佳。
- 不过，在本实验中，因为调整之后的固定学习率1e-1还有再增大一点点会更好的可能性，因此逐渐预热的学习率方案表现也还可以。
- 实验结果如下：
  - 测试集结果：最终正确率：0.9778481012658228
  - 

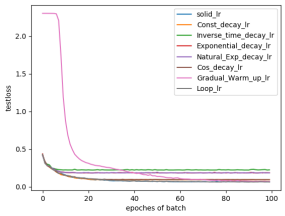
  - 训练集结果: 最终正确率：0.99265625
  - 

- 周期性的学习率调整方案：
  - 在实验中，设定了每次的学习率调整周期为20，每次调整的步幅为0.98.
  - 实验结果如下：
    - 测试集结果：最终正确率：0.9811115506329114
    - 

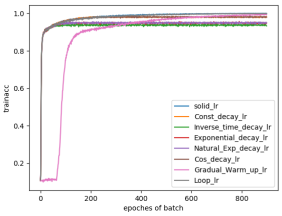
    - 训练集结果：最终正确率：0.998125
    - 

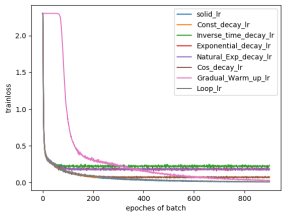
    - 可以看出，周期性的学习率调整方案减少了正确率曲线以及loss曲线的波动性。
- 综上，所有的学习率调整方案的对比实验结果如图所示：
  - 实验结果如下：
    - 测试集结果：





    - 训练集结果：



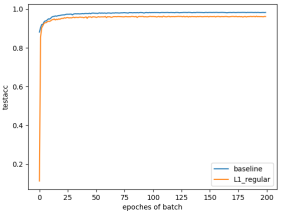


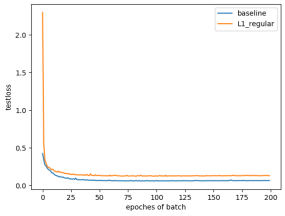
  - 可以发现，基本上在实验中的参数设置条件下，周期循环的学习率方案会更合适一些，泛化性能也是最好的。

不同正则化方案的实验结果对比

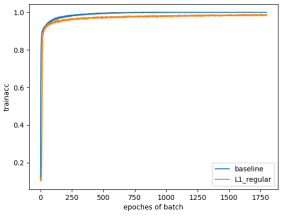
- 此处实验选择固定学习率为1e-1的baseline作为对比,为了模型容易过拟合，放大训练epoch为200轮次。

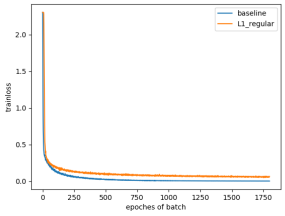
  最终得到的baseline的测试集正确率为0.98220541，训练集正确率为1.000000。
- 正则化方案实现在./reg.py中
- L1正则化：
  - 正则项为$\lambda \sum_{i=1}^{n} |\theta_i|$
  - 实验结果如下：
    - 测试集结果：最终正确率：0.96162975



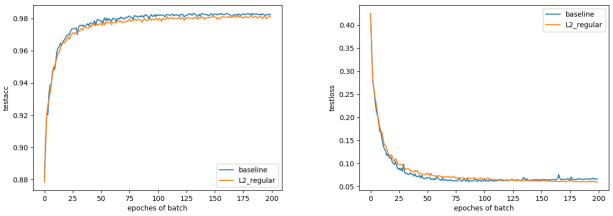


    - 训练集结果：最终正确率：0.98281250

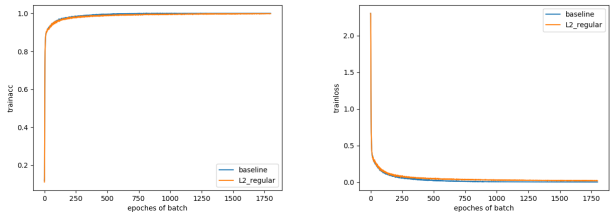




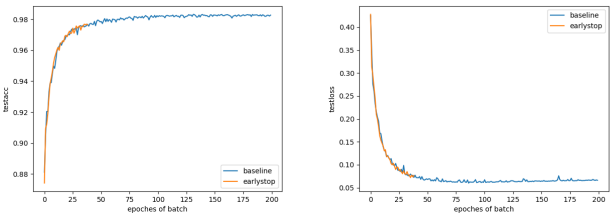
- L2正则化：
  - 正则项为$\lambda \sum_{i=1}^{n} \theta_i^2$
  - 实验结果如下：
    - 测试集结果：最终正确率：0.98121044



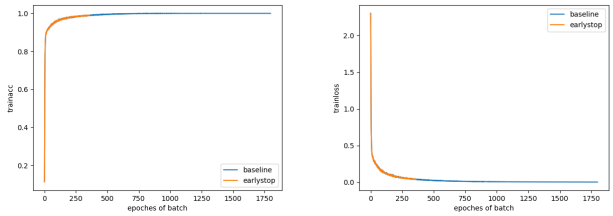
- 训练集结果：最终正确率：0.99812500



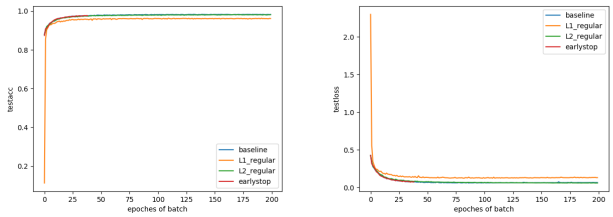
- earlystopping:
  - 设定连续四轮epoch，测试集正确率不增长就停止训练。
  - 实验结果如下：
    - 测试集结果：最终正确率：0.97596915



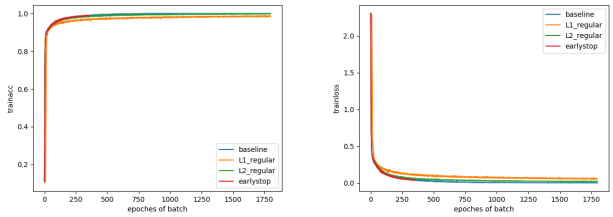
- 训练集结果：最终正确率：0.98859375



  - 从图像上我们就可以看出earlystopping方法的缺点在于提前停止，可能导致模型仍处于一个泛化性能继续提升的过程中就被叫停,为了取得不错的结果，earlystopping方法的参数设置需要斟酌再三；而其优点也很明显，节省了大量时间开销来换取一点点性能提升。
- 四种方案的综合对比图；
  - 测试集结果：

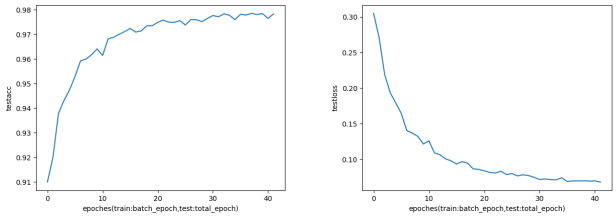


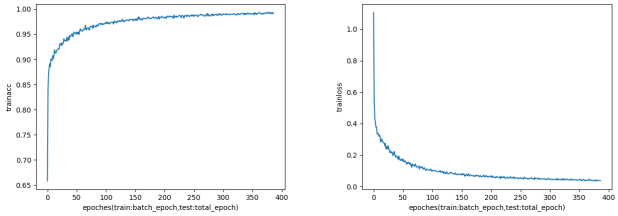
  - 训练集结果：



- 可以看出，earlystop在非常短epoch范围内就已经达到了很高的正确率，而L1正则化与L2正则化都防止了模型在训练集上正确率达到100%,在测试集上的泛化能力的表现来看，L1正则化与其他方法有所差别，L2正则化与baseline的泛化能力相差无几。

一个总体表现不错的参数方案

- 设置初始化方法为正交初始化，采取交叉熵作为损失函数，用L2正则化以及earlystop方法，以及参数调整后的阶段性常数衰减的学习率调整方案。
- 经过一轮训练，就可以达到90+的验证集正确率。在使用earlystop保证训练效率的情况下，模型的最终训练结果如下：
  - 最后一轮训练后的正确率情况：
    -
```
Iteration: 50, Batch Loss:0.03498203, Batch accuracy:0.99343750
Iteration:100, Batch Loss:0.04413701, Batch accuracy:0.99031250
Iteration:150, Batch Loss:0.03862097, Batch accuracy:0.99218750
Iteration:200, Batch Loss:0.03725046, Batch accuracy:0.99265625
Iteration:250, Batch Loss:0.03883618, Batch accuracy:0.99296875
Iteration:300, Batch Loss:0.03982929, Batch accuracy:0.99031250
Iteration:350, Batch Loss:0.03898584, Batch accuracy:0.99296875
Iteration:400, Batch Loss:0.03821162, Batch accuracy:0.99140625
Iteration:450, Batch Loss:0.03707996, Batch accuracy:0.99203125
Test : Batch Loss:0.06755674, Batch accuracy:0.97854035
```
  - 测试集结果：
    - 

  - 训练集结果：
    - 

  - 在设定earlystop的探测正确率降低周期为5的情况下，模型训练了40轮后停止了训练，测试集的最终正确率为0.97854035，训练集的每个batch正确率都在99%以上。尽管，从正确率上看略逊色于固定学习率为1e-1的baseline训练200轮的结果，但是在综合时间效率之后，我认为这种常数设置更为合理一些。