人工神经网络HW1 MNIST Digit Classification with MLP

一、网络结构说明

1.1 基本参数

通过测试Google的neural network playground <https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.49537&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>)我发现了一些常见的学习率参数0.001，0.01，0.03，0.1，0.3，1，3，10等。经过短期试验，我发现学习率0.03在本机上运行收敛速度较快，同时能够得到比较满意的准确度，因此我主要选用0.03为学习率。其余参数的设置通过实验以及同学交流，参与交流的同学为陈博涵、张哲昕。另外关于参数设置我还借鉴了该博客<https://www.cnblogs.com/guoyaohua/p/8542554.html>

**Weight\_decay为0，batch\_size为100，momentum为0.9，如果无特殊说明，linear层的std为0.001。**

1.2 运行环境

本机运行MacBook Air，内存为4 GB 1600 MHz DDR3，处理器为1.4 GHz Intel Core i5，机型性能较次。（可以发现本机运行时间较长，经过排查，应是本机MacBook Air机型较老算力不足所致）

1.3 无隐藏层全连接网络

学习率为0.01，损失函数Euclidean。

|  |  |
| --- | --- |
| 层 | 1 |
| 类型 | Linear |
| 输入节点数 | 784 |
| 输出节点数 | 10 |

1.4 一个隐藏层激活函数为ReLU网络

学习率为0.03，损失函数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 |
| 类型 | Linear | ReLU | Linear | ReLU |
| 输入节点数 | 784 | 256 | 256 | 10 |
| 输出节点数 | 256 | 256 | 10 | 10 |

1.5 一个隐藏层激活函数为ReLU网络损失函数为SoftmaxCrossEntropy

学习率为0.03，损失函数SoftmaxCrossEntropy。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 |
| 类型 | Linear | ReLU | Linear | ReLU |
| 输入节点数 | 784 | 256 | 256 | 10 |
| 输出节点数 | 256 | 256 | 10 | 10 |

1.6 一个隐藏层激活函数为Sigmoid网络

学习率为0.03，损失函数Euclidean。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 |
| 类型 | Linear | Sigmoid | Linear | Sigmoid |
| 输入节点数 | 784 | 256 | 256 | 10 |
| 输出节点数 | 256 | 256 | 10 | 10 |

1.7 两个隐藏层激活函数为ReLU网络

学习率为0.03，损失函数Euclidean，linear层的std为0.01。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 | 5 | 6 |
| 类型 | Linear | ReLU | Linear | ReLU | Linear | ReLU |
| 输入节点数 | 784 | 256 | 256 | 128 | 128 | 10 |
| 输出节点数 | 256 | 256 | 128 | 128 | 10 | 10 |

1.8 两个隐藏层激活函数为Sigmoid网络

学习率为0.03，损失函数Euclidean，linear层的std为0.01。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 | 5 | 6 |
| 类型 | Linear | Sigmoid | Linear | Sigmoid | Linear | Sigmoid |
| 输入节点数 | 784 | 256 | 256 | 128 | 128 | 10 |
| 输出节点数 | 256 | 256 | 128 | 128 | 10 | 10 |

1.9 两个隐藏层激活函数为ReLU网络损失函数为SoftmaxCrossEntropy

学习率为0.08，损失函数SoftmaxCrossEntropy，linear层的std为0.01。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 | 5 | 6 |
| 类型 | Linear | ReLU | Linear | ReLU | Linear | ReLU |
| 输入节点数 | 784 | 256 | 256 | 128 | 128 | 10 |
| 输出节点数 | 256 | 256 | 128 | 128 | 10 | 10 |

二、模型运行结果

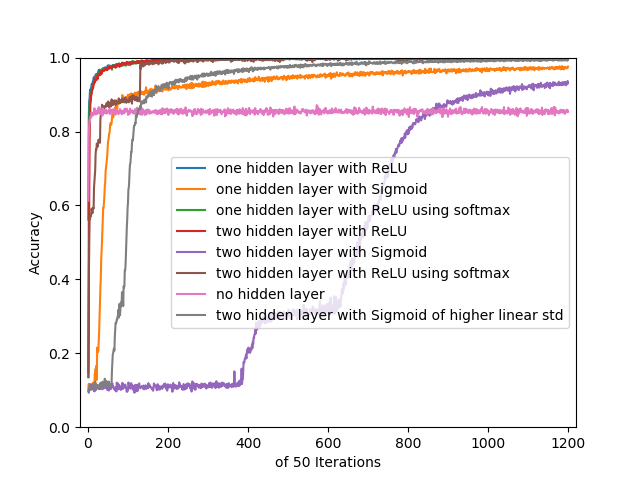
2.1 汇总实验结果

注：无特殊说明，本模块网络损失函数均使用Euclidean（绘制限制，同时ReLU在本任务中表现更好，Euclidean与ReLU的组合性能更好）

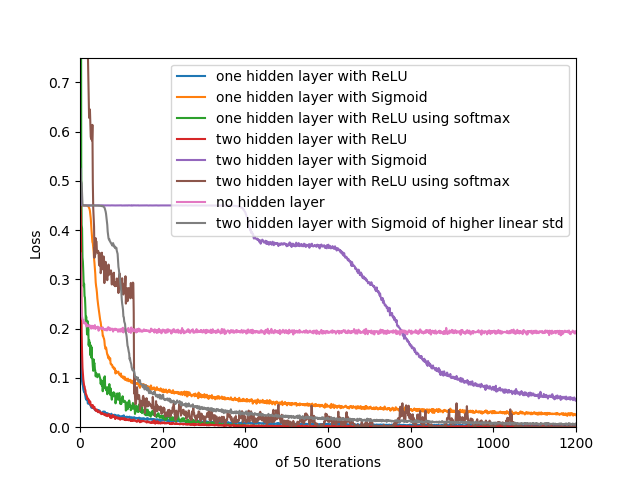
由于画图限制，在此图中所展示的曲线默认损失函数为Euclidean损失函数。关于两种函数的性能讨论详见后文，这里只给出了ReLU激活函数条件下两种损失函数的对比曲线。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 网络名称 | 无隐藏层 | 一个隐藏层ReLU | 一个隐藏层ReLU+softmax | 一个隐藏层Sigmoid | 两个隐藏层ReLU | 两个隐藏层ReLU+softmax | 两个隐藏层Sigmoid  (higher linear std) | 两个隐藏层Sigmoid  (normal linear std) |  |
| Training\_acc | 0.8532 | 0.9970 | **1.0000** | 0.9756 | 0.9980 | **1.0000** | 0.9946 | 0.9308 |  |
| Training\_loss | 0.1911 | 0.0036 | 0.0004 | 0.0244 | 0.0011 | **8.370e-06** | 0.0058 | 0.0564 |  |
| Testing\_acc | 0.8624 | 0.9825 | 0.9818 | 0.9665 | **0.9836** | 0.9834 | 0.9783 | 0.9274 |  |
| Testing\_loss | 0.1930 | 0.0203 | 0.0780 | 0.0301 | **0.0130** | 0.1533 | 0.0178 | 0.0597 |  |
| 运行时长 | **0:01:02.340** | 0:10:14.125 | **0:08:24.391** | 0:09:23.604 | 0:11:23.919 | 0:10:33.457 | 0:09:22.075 | 0:11:12.226 |  |

2.2 网络training\_acc图表



2.3 网络training\_loss图表



三、实验结论

3.1 Sigmoid激活函数与ReLU激活函数在MNIST Digit Classification任务上的区别

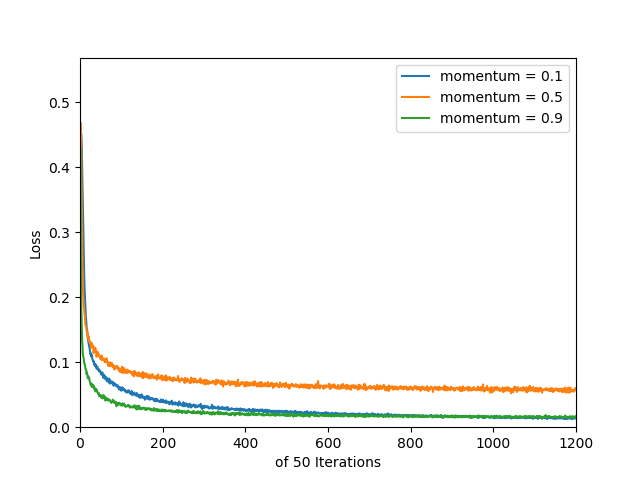
对比曲线one hidden layer with ReLU 与 one hidden layer with Sigmoid、two hidden layer with ReLU 与 two hidden layer with Sigmoid，与Sigmoid激活函数相比，我们发现使用ReLU激活函数的准确度高，运行时间短，loss更小。

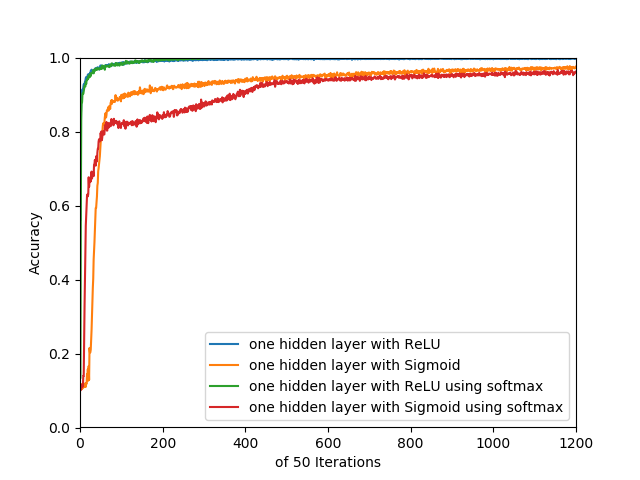
3.2 EuclideanLoss与SoftmaxCrossEntropyLoss的对比

对比曲线one hidden layer with ReLU 与 one hidden layer with ReLU using softmax、two hidden layer with ReLU 与 two hidden layer with ReLU using softmax，我们发现使用了softmax交叉熵loss的网络运行时间短，训练准确度高（甚至达到了100%的训练准确度）。值得注意的是，虽然softmax交叉熵loss的训练准确度、训练loss均优于Euclidean loss，但是在测试准确度和loss上softmax交叉熵的性能不及Euclidean loss，这可能是因为出现了过拟合的现象。

另外，Euclidean与softmax交叉熵针对不同的激活函数表现出不同的性能。在ReLU激活函数网络中，softmax交叉熵能够提升准确率、降低loss；但是在Sigmoid激活函数网络中，softmax交叉熵将loss提升至一个较大值（1.5左右）。虽然我不清楚这是什么原因，但是这个现象验证了loss与准确率并不是负相关，它们之间有一定的独立性。另外，在给定的学习率条件下，softmax交叉熵增加了Sigmoid网络的运行时间，略微降低了准确率，根据3.5的结论，准确率的升降或许会因学习率设定不同而有所不同。

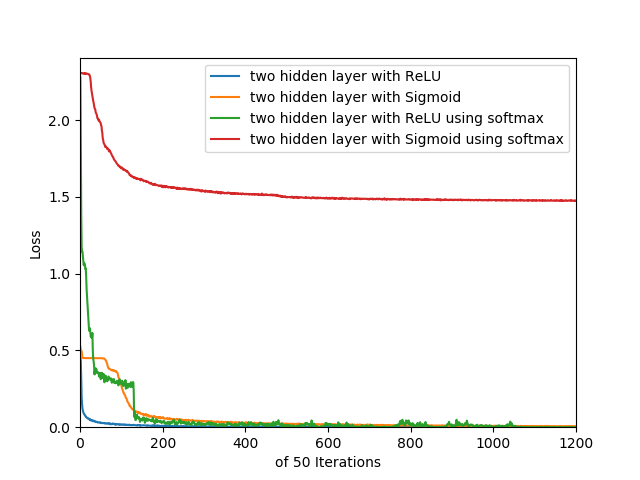
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 网络 | ReLU+Euclidean  (learning rate = 0. 03) | ReLU+softmax  (learning rate = 0. 03) | Sigmoid+Eucilidean  (learning rate = 0. 03) | Sigmoid+softmax  (learning rate = 0. 03) |
| Training\_acc | 0.9970 | **1.0000** | 0.9756 | 0.9608 |
| Training\_loss | 0.0036 | **0.0004** | 0.0244 | 1.4989 |
| Testing\_acc | **0.9825** | 0.9818 | 0.9665 | 0.9564 |
| Testing\_loss | **0.0203** | 0.0780 | 0.0301 | 1.5044 |
| 运行时长 | 0:10:14.125 | **0:08:24.391** | 0:09:23.604 | 0:10:16.582 |

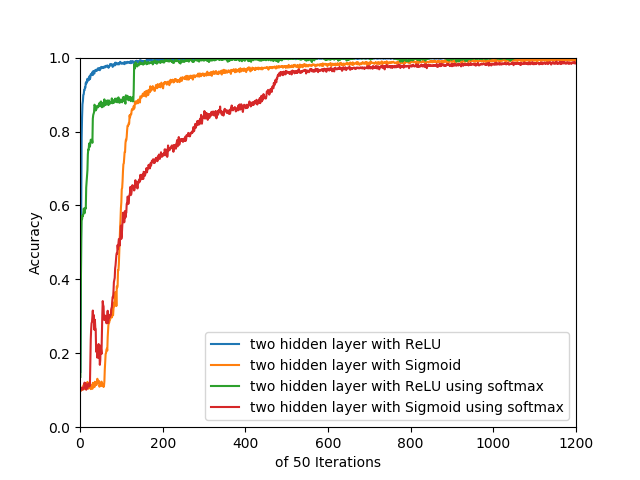




另外，这两个loss函数的差距在两个隐藏层的网络中也呈现出单个隐藏层的特征。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 网络 | ReLU+Euclidean  (learning rate = 0.03) | ReLU+softmax  (learning rate = 0.08) | Sigmoid+Eucilidean  (learning rate = 0. 03) | Sigmoid+softmax  (learning rate = 0.08) |
| Training\_acc | 0.9980 | **1.0000** | 0.9946 | **0.9854** |
| Training\_loss | 0.0011 | **8.370e-06** | 0.0058 | **1.4753** |
| Testing\_acc | **0.9836** | 0.9834 | 0.9783 | 0.9725 |
| Testing\_loss | **0.0130** | 0.1533 | 0.0178 | 1.4879 |
| 运行时长 | 0:11:23.919 | 0:10:33.457 | 0:09:22.075 | 0:11:45.847 |





3.3 隐藏层数量对MNIST Digit Classification任务的影响

通过对于一个隐藏层、两个隐藏层的ReLU激活函数的实验结果，结合基本的复杂度分析，两个隐藏层的计算量更大（多了数百个全连接节点），因而运行时间长。但是在其他性能上，由于ReLU激活函数本身已经很接近100%准确度了，因而没有明显差距，但是Sigmoid激活函数的运行结果来看，两个隐藏层的网络除时间外性能均优于单层网络。

隐藏层是必须的，在图中可以看出，单独加全连接网络的loss和准确度比其他任何网络都差。

另外，优于增加隐藏层带了的计算量和未知量数目的大幅增加，必要的调整学习率与网络初始std是必要的，在2.2的图中我们能看到三条曲线one hidden layer with Sigmoid、two hidden layer with Sigmoid、two hidden layer with Sigmoid，调整的原因是，如果不调整则到相同的训练epoch之后，多隐藏层的网络还没达到收敛，虽然表面上控制了变量，但是对比的基数不同，为了快速达到收敛，需要同时提高其学习率。他们的区别见下表

|  |  |  |  |
| --- | --- | --- | --- |
| 网络 | one hidden layer with Sigmoid | two hidden layer with Sigmoid | two hidden layer with Sigmoid |
| Linear层初始std | 0.001 | 0.001 | 0.01 |
| 学习率 | 0.03 | 0.03 | 0.03 |

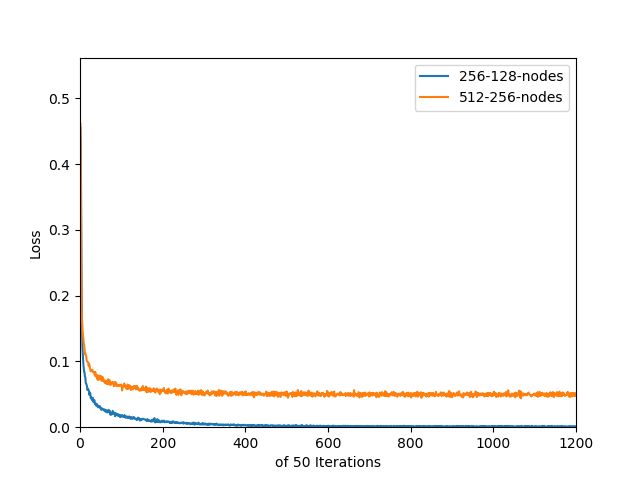
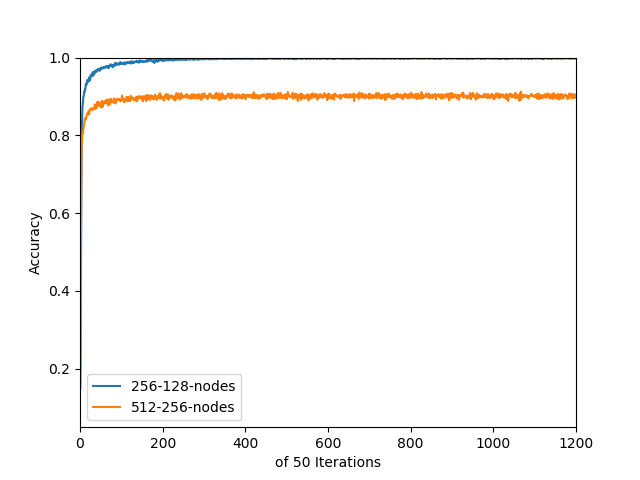
3.4 隐藏层新增节点的数量设置

我尝试了两种节点设置。AB两个网络的属性列于下表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 层 | 1 | 2 | 3 | 4 | 5 | 6 |
| 类型 | Linear | ReLU | Linear | ReLU | Linear | ReLU |
| A输入节点数 | 784 | 256 | 256 | 128 | 128 | 10 |
| A输出节点数 | 256 | 256 | 128 | 128 | 10 | 10 |
| B输入节点数 | 784 | 512 | 512 | 256 | 256 | 10 |
| B输出节点数 | 512 | 512 | 256 | 256 | 10 | 10 |

符合预期的是，隐藏节点越多的网络的收敛速度更慢，由3.3得出的结论，需要更高的学习率，因而…….结果如下

|  |  |  |
| --- | --- | --- |
| 网络 | A | B |
| Training\_acc | 0.9980 | 0.9010 |
| Training\_loss | 0.0011 | 0.0497 |
| Testing\_acc | **0.9836** | 0.8887 |
| Testing\_loss | **0.0130** | 0.0605 |
| 运行时长 | 0:11:23.919 | 0:23:35.060 |

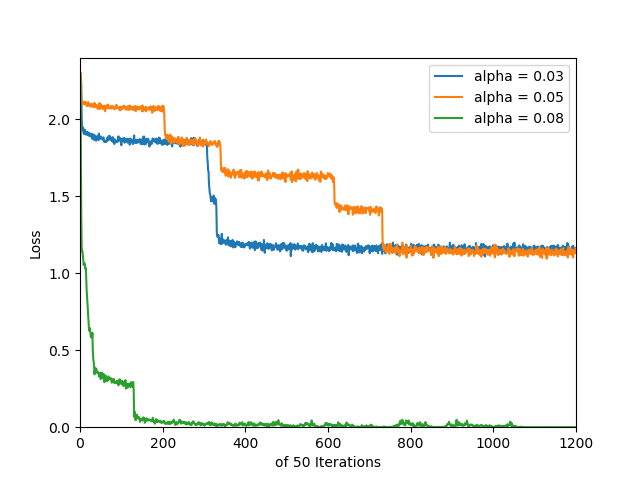


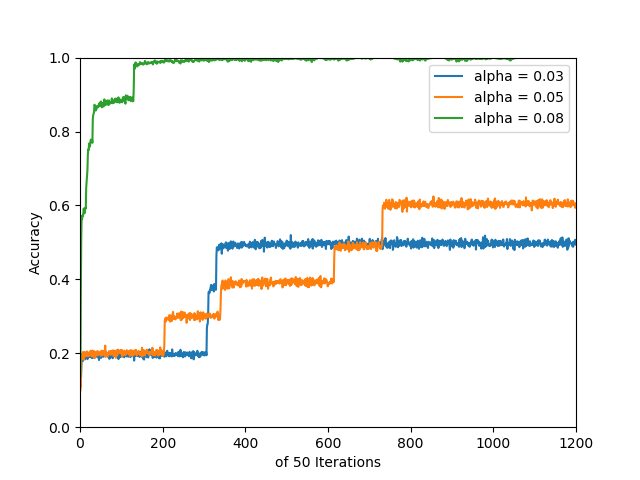
3.5 学习率对MNIST Digit Classification任务的影响

针对1.9中（两层ReLU+softmax交叉熵损失

函数）的网络，我做了三组实验，列于下表

|  |  |  |  |
| --- | --- | --- | --- |
| 学习率 | 0.03 | 0.05 | 0.08 |
| Training\_acc | 0.4996 | 0.6056 | **1.0000** |
| Training\_loss | 1.1536 | 1.1412 | **8.370e-06** |
| Testing\_acc | 0.4967 | 0.5957 | 0.9834 |
| Testing\_loss | 1.2115 | 1.2001 | 0.1533 |
| 运行时长 | 0:11:23.997 | 0:11:29.718 | 0:10:33.457 |





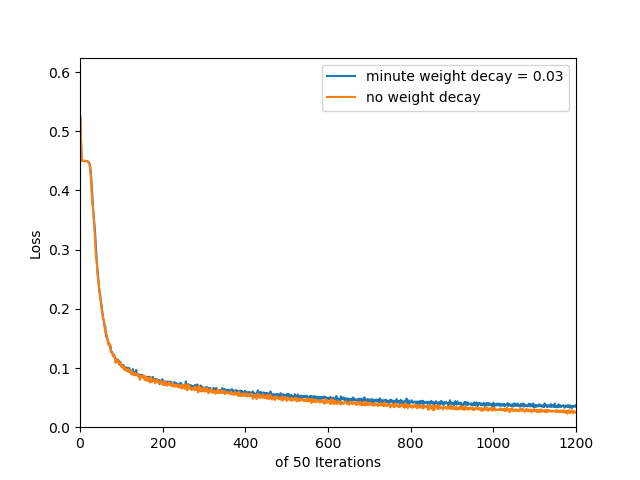
从结果可以看出，对于计算量相对较大的网络，如果学习率不足，可能导致梯度下降时陷入局部最小值，无法迈出足够的步长，导致有限的训练epoch内收敛在了很低的acc内。

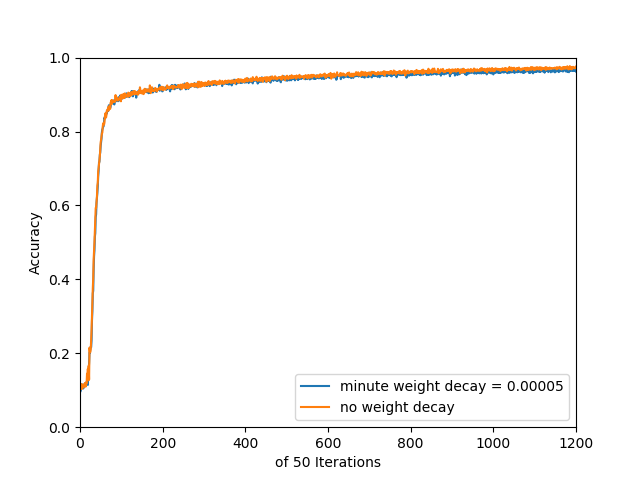
但是，学习率过大容易导致过拟合现象。表现为训练高准确度，测试准确度比训练准确度低一定量。

3.6 weight\_decay对MNIST Digit Classification任务的影响

虽然有许多参考资料支持痕量的weight\_decay能够提升性能，但是在实验中，我使用weight\_decay = 0.00005与0在单层Sigmoid网络中的性能对比如下表

|  |  |  |
| --- | --- | --- |
| Weight\_decay | 0.00005 | 0 |
| Training\_acc | 0.9624 | 0.9756 |
| Training\_loss | 0.0373 | 0.0244 |
| Testing\_acc | 0.9614 | 0.9665 |
| Testing\_loss | 0.0372 | 0.0301 |
| 运行时长 | 0:09:43.084 | 0:09:23.604 |





3.7 batch\_size、max\_epoch对MNIST Digit Classification任务的影响

max\_epoch主要影响的是训练迭代的次数，对于一个网络，如果在一定的epoch数量内能够达到收敛，那么此时max\_epoch最高取比该epoch稍大的值，既能使之收敛，又能防止可能出现的过拟合现象、

batch\_size主要影响每一次迭代的“视野”。Batch\_size过小，单次迭代的稳定性下降，会受到偏差较大的样本的影响，在最终收敛阶段准确率仍然会有明显波动；batch\_size过大，单次迭代会忽略单个样本的特性，使输入样本模糊化，在classification任务中的体现为准确率下降。

3.8 momentum对MNIST Digit Classification任务的影响

针对1.4中（单层ReLU+Euclidean损失函数）的网络，我做了三组实验，列于下表

|  |  |  |  |
| --- | --- | --- | --- |
| momentum | 0.1 | 0.5 | 0.9 |
| Training\_acc | 0.9904 | 0.8986 | 0.9970 |
| Training\_loss | 0.0143 | 0.0570 | 0.0036 |
| Testing\_acc | 0.9797 | 0.8869 | 0.9825 |
| Testing\_loss | 0.0232 | 0.0678 | 0.0203 |
| 运行时长 | 0:08:32.675 | 0:08:44.582 | 0:10:14.125 |

