人工神经网络 第一次作业 实验报告

计72 李岱轩 2017012289

1 神经网络设计

本次实验神经网络设计中，我根据要求对代码进行了补全，同时改动了部分接口以首先更简单的神经网络搭建。

我实现了relu, gelu, sigmoid激活函数， **并把这些函数封装在了Linear类中**，使得一个线性层可以直接选择自己的激活函数。同时，我也实现了欧几里得，交叉熵和hinge损失函数。

细节设计上，我**对每一个损失函数之前先进行了softmax处理来合理化梯度**；我**对交叉熵和gelu中的指数计算，进行了截断**以防止溢出；最后实验结果表明，这样的设计是合理的。

2 实验设计

在搭建神经网络进行测试的过程中，我尝试了如下实验：

- 一个隐藏层（256个节点），relu激活函数，CrossEntropy损失函数；
- 两个隐藏层（500，256个节点），relu激活函数，CrossEntropy损失函数；
- 两个隐藏层（500，256个节点），gelu激活函数，CrossEntropy损失函数；
- 两个隐藏层（500，256个节点），sigmoid激活函数，CrossEntropy损失函数；
- 两个隐藏层（500，256个节点），relu激活函数，Eucledian损失函数；
- 两个隐藏层（500，256个节点），relu激活函数，Hinge损失函数（dalta取0.5，先过softmax）；
- 三个隐藏层（500，400，256个节点），relu激活函数，CrossEntropy损失函数；

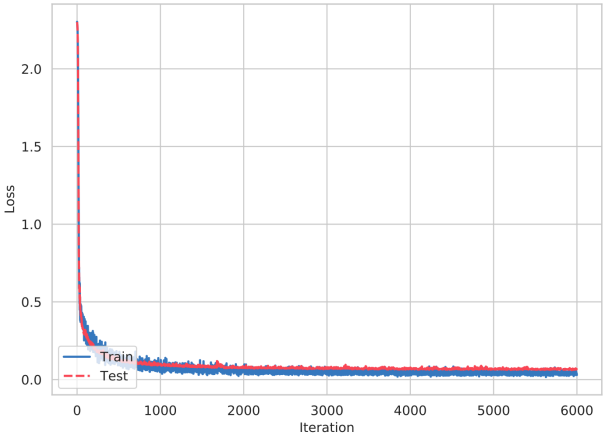
参数设置为：

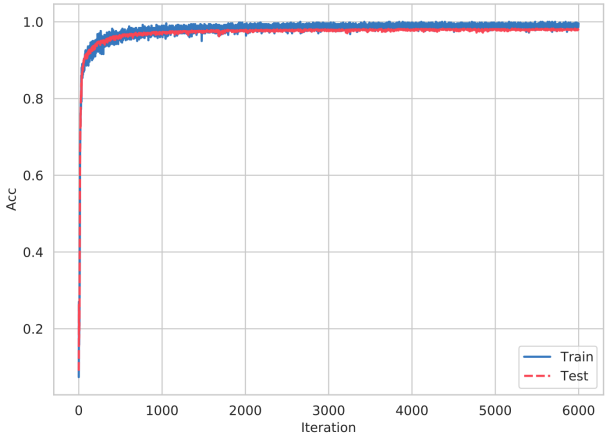
```
config = {
    'learning_rate': 0.1,
    'weight_decay': 0.0005,
    'momentum': 0.9,
    'batch_size': 500,
    'max_epoch': 50,
    'disp_freq': 50,
    'test_epoch': 5
}
```

经过实验，所有网络都在上述设置下收敛，部分网络收敛较快，但为了使画图的标尺一致，也保持设置不变。

3 实验结果

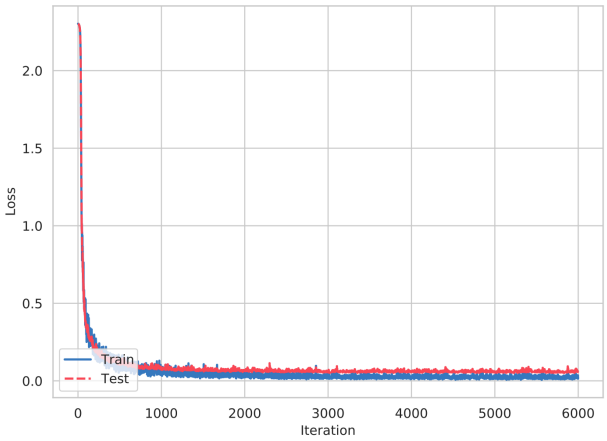
（1）一个隐藏层（256个节点），relu激活函数，CrossEntropy损失函数

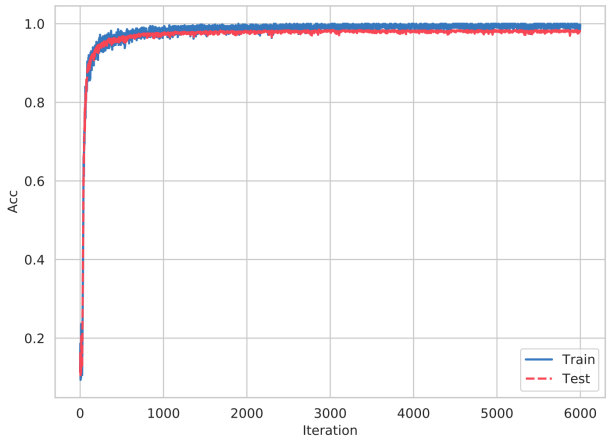




运行时间1066s，最高准确率0.9826。

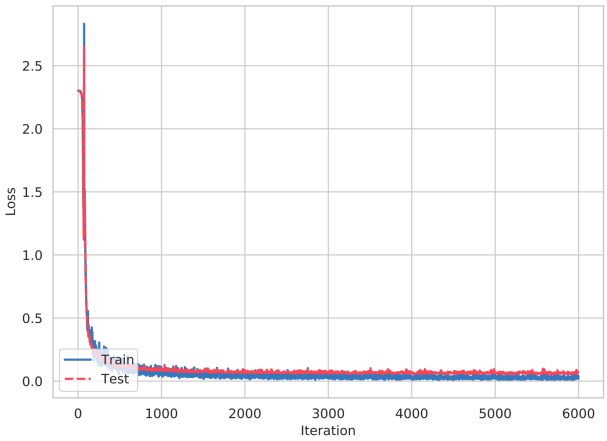
（2）两个隐藏层（500，256个节点），relu激活函数，CrossEntropy损失函数

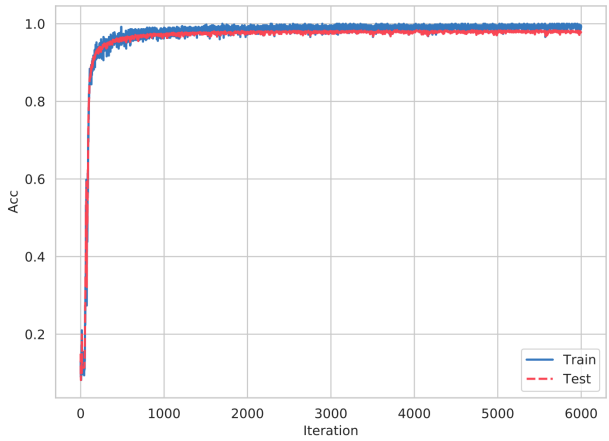




运行时间2053s，最高准确率0.9853。

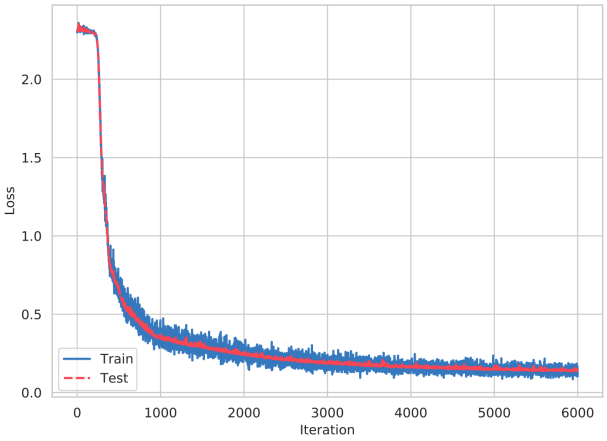
（3）两个隐藏层（500，256个节点），gelu激活函数，CrossEntropy损失函数

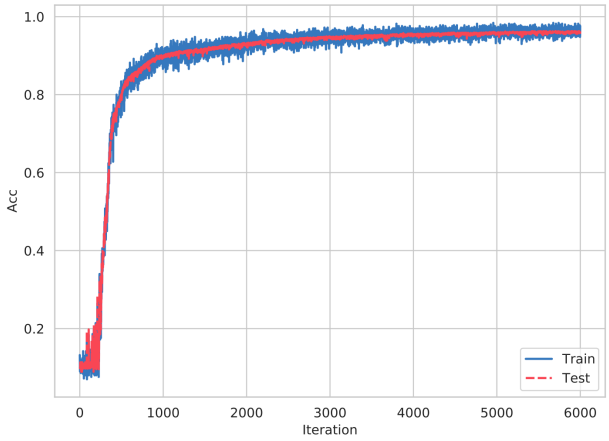




运行时间6808s，最高准确率0.9838。

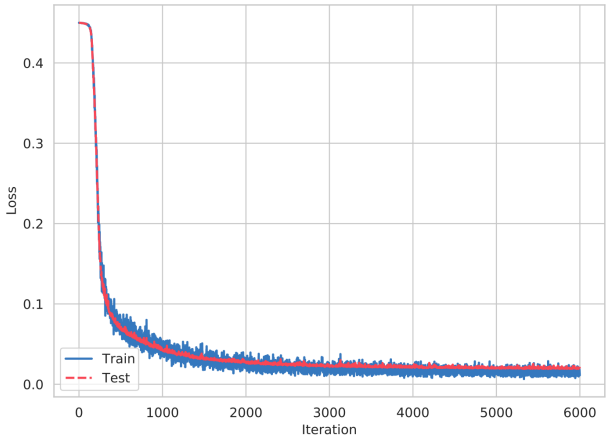
（4）两个隐藏层（500，256个节点），sigmoid激活函数，CrossEntropy损失函数

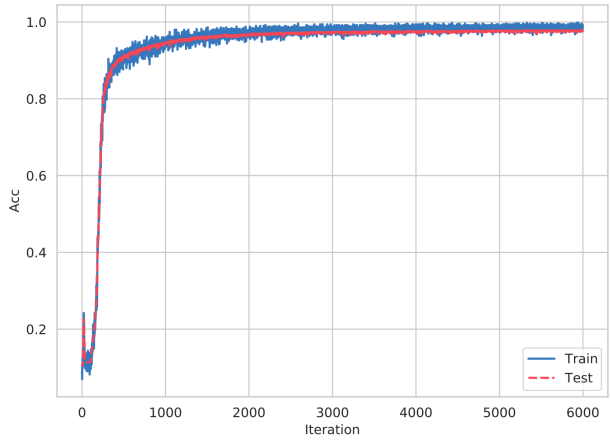




运行时间2425s，最高准确率0.9641。

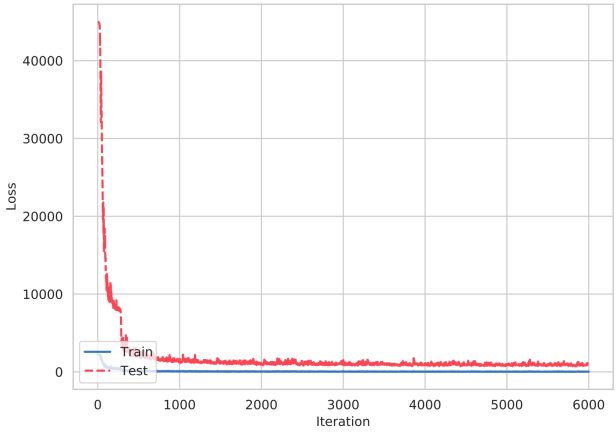
（5）两个隐藏层（500，256个节点），relu激活函数，Eucledian损失函数

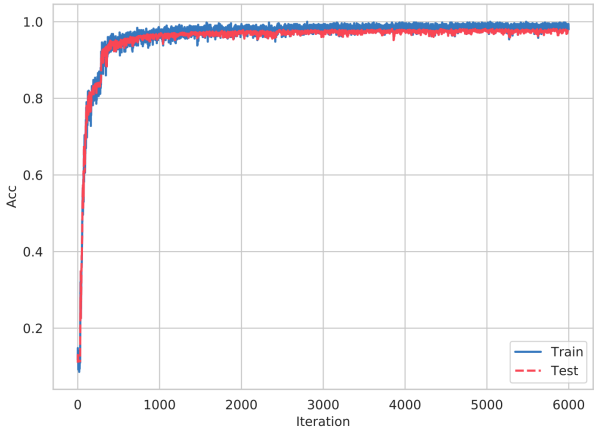




运行时间1943s，最高准确率0.9793。

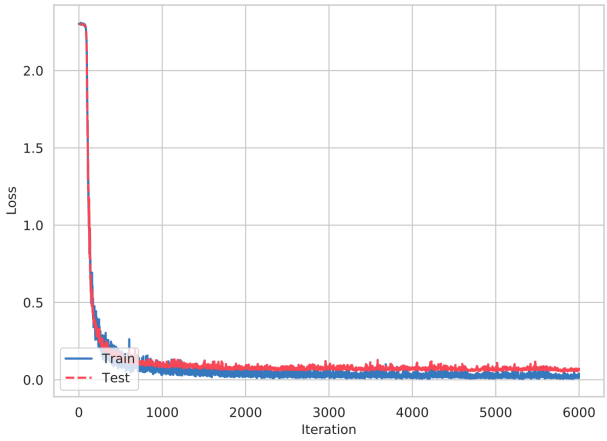
（6）两个隐藏层（500，256个节点），relu激活函数，Hinge损失函数

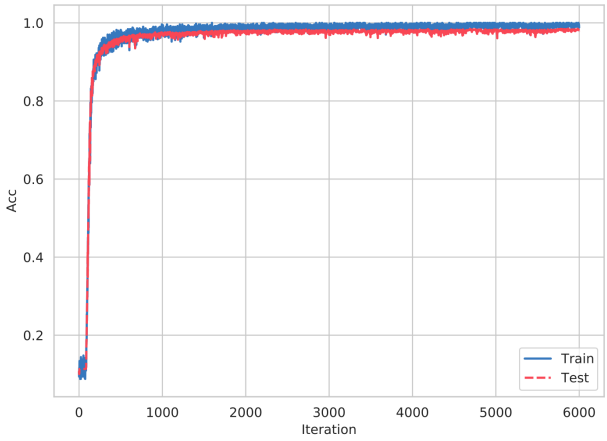




运行时间1902s，最高准确率0.9816。

（7）三个隐藏层（500，400，256个节点），relu激活函数，CrossEntropy损失函数





运行时间2768s，最高准确率0.9853。

4 实验结果与讨论

- 从（1） （2） （7）可以看出， 多层网络训练效果比单层网络好，但是训练时间更长，收敛速度相对较慢；
- 从（2） （5） （6）可以看出，CrossEntrophy损失函数比Hinge和Euclidian函数效果都更好，收敛速度也更快，准确率也更好；
- 从（2） （3） （4）可以看出， gelu函数相比relu函数，再相比sigmoid激活函数，准确率也更好，收敛更快（epoch上来看）。

从理论上分析可以得到和实验结果相同的结论。

多层网络由于具有更多的参数，参数变化的自由度更大，因此最终的性能也会更好。但是由于参数多导师计算开销更大，因此效率不是很高。

CrossEntrophy对错误标签的惩罚较大。一方面对数函数在0-1区域的导数远大于多项式函数，因此收敛速度较快；另一方面由于欧几里得损失函数对少量的正样本产生的惩罚远小于交叉熵，因此后者训练效果会更好；另一方面， Hinge函数在负样本达到delta时就会停止对其进行进一步梯度更新，因此delta的选择也很大的影响了训练的结果，本实验经过softmax之后选择了0.5， 导致了在局部最大值处陷入的现象，训练结果因此变得不好。

gelu和relu函数相比sigmoid更好，是因为后者容易饱和，即陷入导数较小的正半轴区域。经过查阅资料，gelu相比relu增加了随机考量，对输出乘以了正态分布的概率函数，因此得到了更好的实验效果，但是由于计算更复杂，因此时间花费更长。

5 总结

本次实验遇到的最大困难就是计算资源的困难。跑全部上述实验用了约3-4个CPU天，如果没有大规模计算资源很难实现。如果可能，也希望未来能提供一定的计算资源，或者用更轻量级的任务来进行。

总体来讲，实验收获很大。感谢老师和助教的付出。