作业1：前馈神经网络拟合函数

作者：Zezhong 联系：zezhongding@mail.ustc.edu.cn

1 实验要求

使用$PyTorch$或者$TensorFlow$手写一个前馈神经网络，用于近似以下函数

$$y = sinx + e^{-x}, x \in [0, \pi] \tag{1}$$

并研究网络深度、学习率、网络宽度、激活函数对模型性能的影响

2 实验环境

实验是用$Python$基于$PyTorch$框架实现的，并在一台有20个40核$CPU\ E5-2698v4@2.20GHz$、4个显存16G的$NVIDIA\text{-}Tesla\ V100$和256$GB$内存的环境下运行。

3 实验过程

1. 生成数据集：在实验之中我首先在区间$[0, \pi]$之中按照均匀分布采样1×10^5个点。这个过程通过$np.random.uniform$函数实现。

```
#按照均匀分布进行采样
  datax = np.random.uniform(0,4 * PI,datasize)
```

2. 划分数据集：我将采样后的数据点经过函数$y = sinx + e^{-x}$生成每个数据集的标签。然后使用$train_test_split$函数进行划分。我按照公式(2)进行划分。

$$\text{训练集：验证集：测试集} = 6:2:2 \tag{2}$$

```
#将采样后的数据经过函数处理并划分，固定了random_state使得每次划分是一样的
  datay = np.array(list(map(func, datax)))
  train_X, tmp_X, train_Y, tmp_Y = train_test_split(datax, datay,
test_size=0.4, random_state=1)
  valid_X, test_X, valid_Y, test_Y = train_test_split(tmp_X, tmp_Y,
test_size=0.5, random_state=1)
```

3. 构建模型：我采用MLP作为前馈神经网络进行训练，优化器选择$Adam$。

```
optimizer = torch.optim.Adam(mlp.parameters(), lr=learning_rate)
```

在实验过程之中我对网络深度、学习率、网络宽度、激活函数进行调整。

3.1 网络深度：假定都使用$ReLU$激活函数且$epoch = 2 \times 10^3$的条件下对不同网络深度进行实验。$HiddenLayer$分别取1，3，5，7和9。设定每个$HiddenLayer$的节点数目为32。学习率为0.001。在上述限定条件下进行实验。

表1 网络深度变化对实验结果的影响

Number of Hidden Layer	1	3	5	7	9
时间 / $\times 10^1 ms$	1.97	2.59	3.62	4.54	5.45
验证集上的$Loss$	0.34208	0.00086	**0.00009**	0.00053	0.00105

显然通过分析表1的实验结果，我发现在5层的时候验证集上的$Loss$最小。分析图1可以发现，当层数较浅神经网络很难收敛。随着层数加深，网络收敛所需的迭代次数在不断减少。当层数过多时，神经网络出现过拟合情况，$Loss$升高。

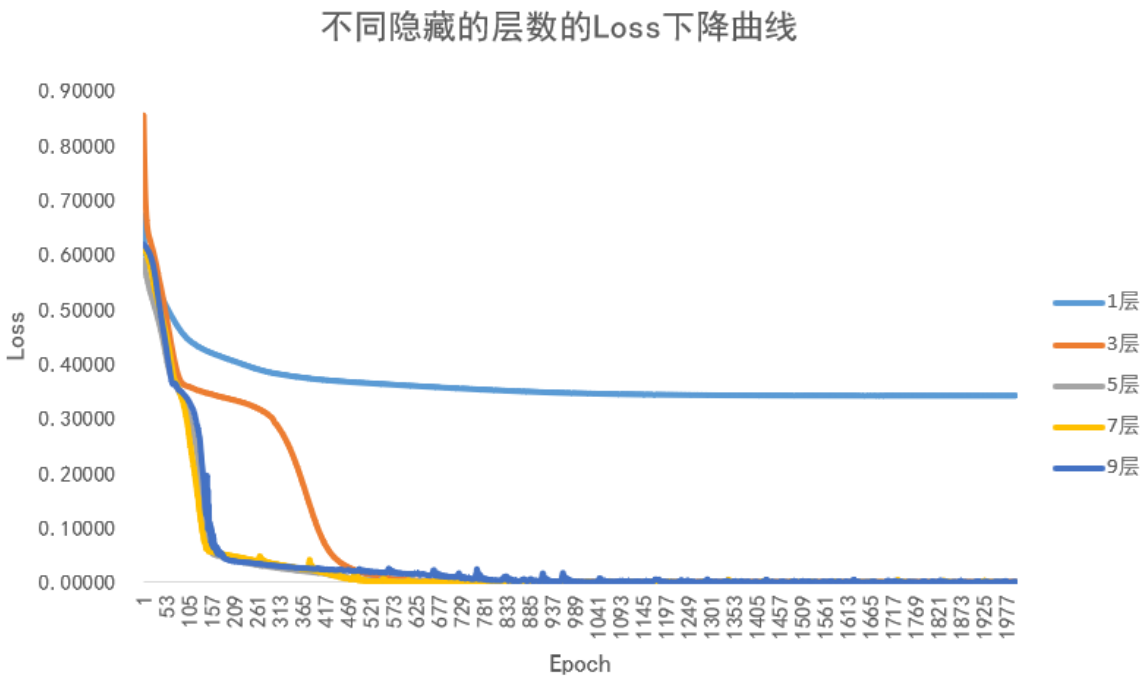

图 1 网络深度变化实验结果

3.2 学习率：假定都使用$ReLU$激活函数且$epoch = 2 \times 10^3$，网络隐藏层深度为5层的条件下对不同网络学习率进行实验。$LearningRate$分别取0.1，0.01，0.001和0.0001。设定每个$HiddenLayer$的节点数目为32。在上述限定条件下进行实验。

表2 学习率变化实验结果

Learning Rate	0.1	0.01	0.001	0.0001
时间 / $\times 10^1 ms$	3.56	3.59	3.60	3.68
验证集上的$Loss$	0.33541	0.00136	0.00012	**0.00008**

通过在验证集上的$Loss$进行分析我发现，学习率为0.0001时，验证集上的$Loss$可以被降低到最小为0.00008。但是，从图2中发现，学习越小它收敛会越慢。但是，如果学习率过大（例如，0.1）。那么模型的$Loss$会发生剧烈震荡，最大可达252.04且最终收敛的$Loss$为0.33541，训练出的模型质量很差。综上，为了在训练速度和模型质量上达到平衡，我们将学习率设置为0.0001。

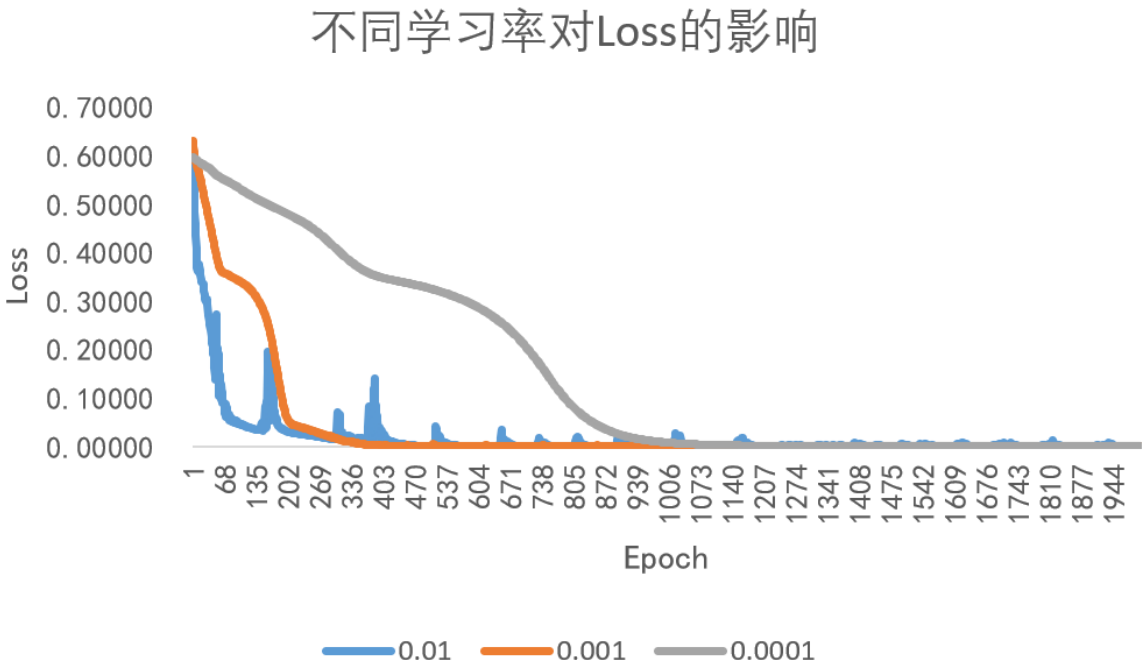

图 2 学习率变化对实验结果的影响

3.3 网络宽度：假定都使用$ReLU$激活函数且$epoch = 2 \times 10^3$，网络隐藏层深度为5层，学习率为0.001的条件下进行实验。设定所有$HiddenLayer$的节点数目分别为16，32，64和128。通过这种方法研究网络宽度对模型所产生的影响。在上述限定条件下进行实验。

表3 网络宽度变化对实验结果的影响

Number of Hidden Unit	16	32	64	128
时间 / $\times 10^1 ms$	4.83	3.60	6.47	19.24
验证集上的$Loss$	0.00054	0.00063	0.00049	**0.0002**

通过对表3和图3的实验数据进行分析，发现随着网络宽度的增加，训练时间会随之增加。在一定范围内，宽度越大，模型性能收敛的性能会越好。

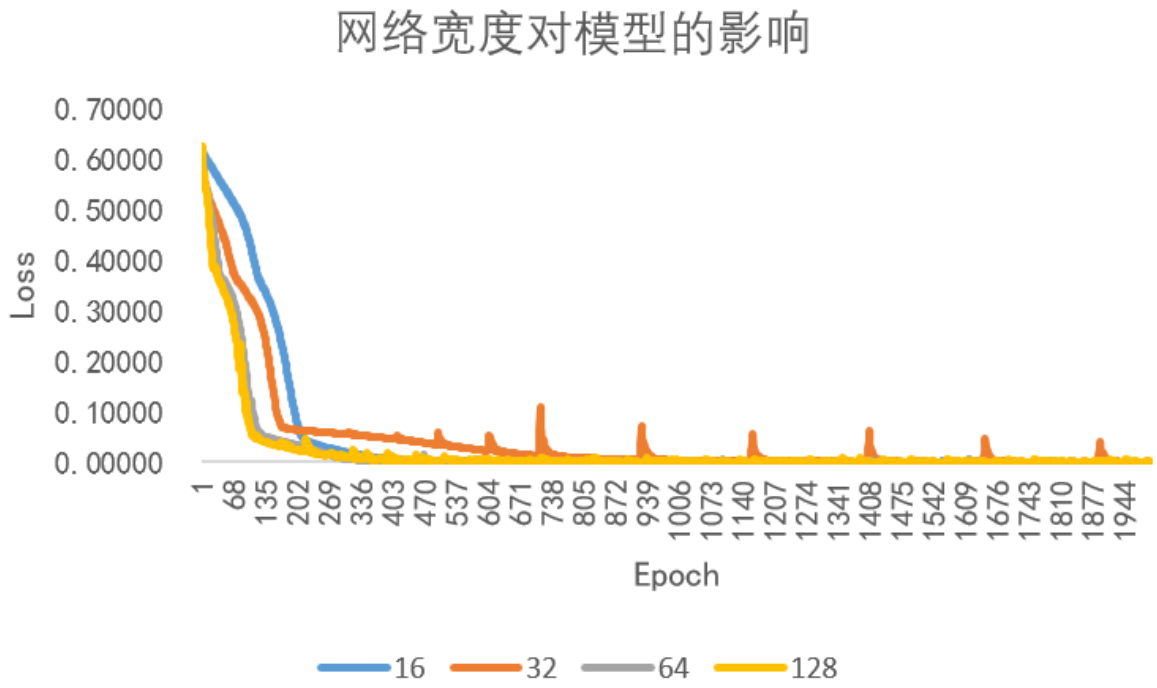

图 3 宽度变化对实验结果的影响

3.4 激活函数：假定$epoch = 2 \times 10^3$，网络隐藏层深度为5层，学习率为0.001的条件下进行实验。设定所有$HiddenLayer$的节点数目为64。实验比较三种类型的激活函函数$Sigmoid$，$Tanh$，$ReLU$和$LeakyReLU$，通过这种方法研究损失函数对模型所产生的影响。其中$LeakyReLU$中的$negative_slope$设置为0.01在上述限定条件下进行实验。

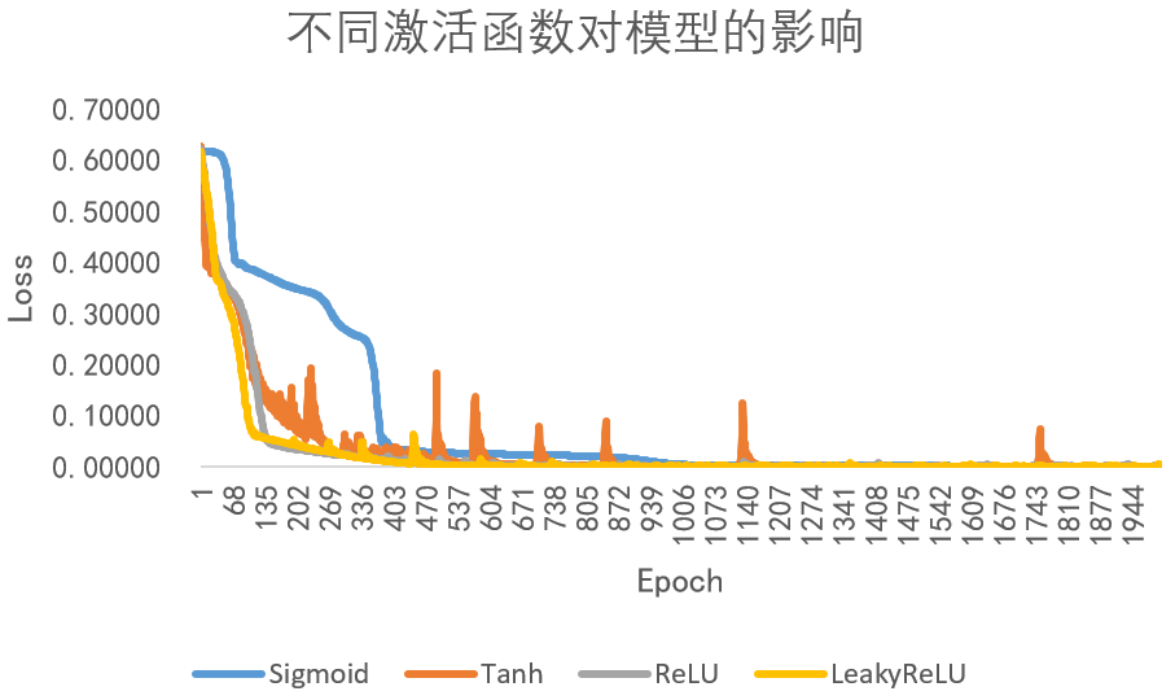

图 4 激活函数对实验结果的影响

表4 损失函数不同对实验结果的影响

Number of Hidden Unit	$Sigmoid$	$Tanh$	$ReLU$	$LeakyReLU$
时间 / $\times 10^1 ms$	19.56	10.57	5.82	30.57
验证集上的$Loss$	0.00003	0.00005	0.00017	0.00453

通过表4和图4的数据可以发现$ReLU$这类激活函数收敛相对更快但是$Loss$会比$Sigmoid$和$Tanh$更高。对于这类问题$LeakyReLU$虽然收敛快，但是$Loss$还是比$ReLU$高。

4. 最终实验

在经过上面的实验之后，经过分析，综合考虑之后选择激活函数为$ReLU$，学习率为0.0001，网络隐藏层深度为5层以及$epoch = 2 \times 10^3$。每个隐藏层的单元数为均为128。这是验证集中表现最好的参数最后的$Loss$可以下降到0.00005

```
#MLP构建代码
class MLP(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.layer1=torch.nn.Linear(1,128)
    self.layer2=torch.nn.Linear(128,128)
    self.layer3=torch.nn.Linear(128,128)
    self.layer4=torch.nn.Linear(128,128)
    self.layer5=torch.nn.Linear(128,128)
    self.layer_final=torch.nn.Linear(128,1)

  def forward(self,x):
    x=self.layer1(x)
    x=torch.nn.functional.relu(x)
    x=self.layer2(x)
    x=torch.nn.functional.relu(x)
    x=self.layer3(x)
    x=torch.nn.functional.relu(x)
    x=self.layer4(x)
    x=torch.nn.functional.relu(x)
    x=self.layer5(x)
    x=torch.nn.functional.relu(x)
    x=self.layer_final(x)
    return x
```

最后的测试	$Valid$	$Test$
时间 / $\times 10^1 ms$	29.37	29.37
MSE	0.00014	0.00013

对于最后结果进行分析发现，每次训练的优化过程之中具有不确定这导致了训练结果会存在波动，最终的MSE为0.00013。

4 实验总结

在这个实验中，我通过对不同参数进行调整分析了不同参数以及激活函数对于模型的影响。总的来说，一定范围内的深度和宽度提升能够使得模型变强。学习率对于模型的训练速度和质量都会产生较大影响，对于不同方法要考虑合适的学习率。$ReLU$激活函数比$Sigmoid$和$Tanh$可以使得训练更快，也可以在缓解梯度消失。