实验一：前馈神经网络

姓名：刘威

学号：PB18010469

实验目的

- 了解并熟悉前馈神经网络的原理及其学习算法
- 了解激活函数在神经网络中的作用
- 了解不同深度及宽度对前馈神经网络性能的影响
- 了解不同学习率对神经网络性能的影响

实验原理

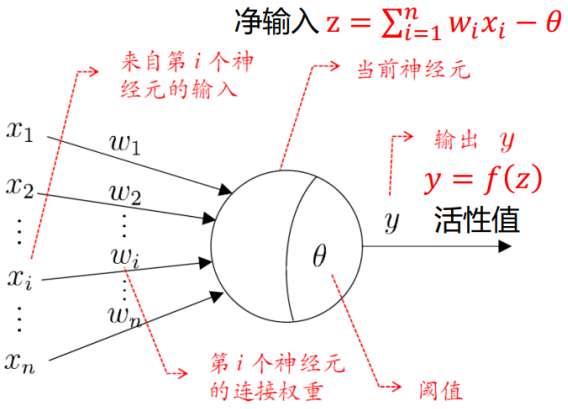
神经网络

- 神经网络的定义

  “神经网络是由具有适应性的简单单元组成的广泛并行互联的网络, 它的组织能够模拟生物神经系统对真实世界物体所作出的反应”

- 神经元模型

  - 输入：来自其他n个神经云传递过来的输入信号
  - 处理：输入信号通过带权重的连接进行传递, 神经元接受到总输入值将与神经元的阈值进行比较
  - 输出：通过激活函数的处理以得到输出



- 激活函数的性质
  - 连续并可导（允许少数点上不可导）的非线性函数，可导的激活函数可以直接利用数值优化的方法来学习网络参数
  - 连续并可导（允许少数点上不可导）的非线性函数，可导的激活函数可以直接利用数值优化的方法来学习网络参数
  - 激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性
- 常用激活函数

  Sigmoid, Tanh, ReLU, LeakyReLU, PReLU, softplus, ELU

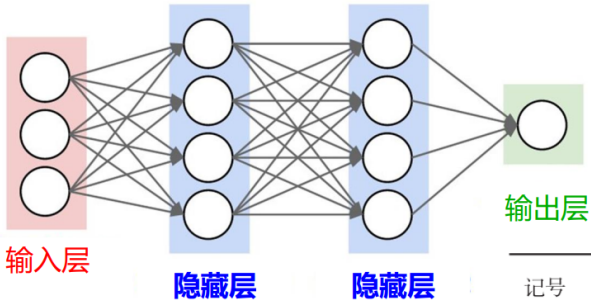
- 神经网络的主要三个特性
  - 信息表示是分布式的
  - 记忆和知识是存储在单元之间的连接上的
  - 通过逐渐改变单元之间的连接强度来学习新知识
- 网络结构

- 神经网络设计的另一个关键点是确定它的结构：具有多少单元，以及这些单元应该如何连接。
- 大多数神经网络被组织成层的单元组
- 大多数神经网络架构将这些层布置成链式结构，其中每一层都是前一层的函数
- 神经网络设计在于选择网络的深度和每一层的宽度，更深层网络通常能在每一层使用更少的单元数和更少的参数，并且有更强的泛化能力。但是通常也更难以优化。

前馈神经网络

前馈神经网络又称为多层感知机， 主要特点为：

- 各神经元分别属于不同的层，层内无连接
- 相邻两层之间的神经元全部两两连接
- 整个网络中无反馈，信号从输入层向输出层单向传播



给定一个前馈神经网络，用下面的记号来描述这样网络

一般层数L只考虑隐藏层和输出层

记号	含义
L	神经网络的层数
M_l	第 l 层神经元的个数
$f_l(\cdot)$	第 l 层神经元的激活函数
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	第 $l-1$ 层到第 l 层的权重矩阵
$\boldsymbol{b}^{(l)} \in \mathbb{R}^{M_l}$	第 $l-1$ 层到第 l 层的偏置
$\boldsymbol{z}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的净输入（净活性值）
$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的输出（活性值）

- 前馈神经网络通过下面公式进行信息传播

仿射变换 $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$　　$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)})$ 非线性变换

$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}) \qquad \boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)} f_l(\boldsymbol{z}^{(l-1)}) + \boldsymbol{b}^{(l)}$$

- 前馈计算：

$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \rightarrow \boldsymbol{z}^{(1)} \rightarrow \boldsymbol{a}^{(1)} \rightarrow \boldsymbol{z}^{(2)} \rightarrow \boldsymbol{a}^{(2)} \rightarrow \cdots \rightarrow \boldsymbol{a}^{(L-1)} \rightarrow \boldsymbol{z}^{(L)} \rightarrow \boldsymbol{a}^{(L)}$$

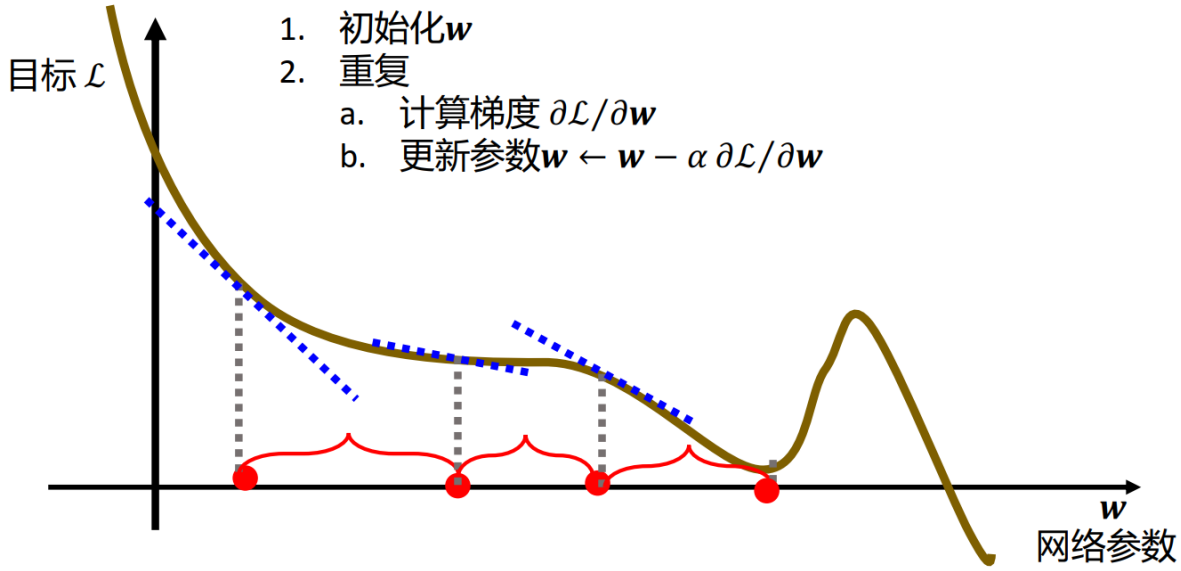
万能近似定理（Universal Approximation Theorem）

一个前馈神经网络如果具有线性输出层和至少一层具有任何一种“挤压” 性质的激活函数（例如logistic sigmoid激活函数）的隐藏层，只要给予网络足够数量的隐藏单元，它可以以任意的精度来近似任何从一个有限维空间到另一个有限维空间的Borel 可测函数。

万能近似定理只说明神经网络表达能力强大到可以近似任意一个的连续函数，却并没有给出如何找到这样的神经网络，以及是否是最优的。

网络参数学习：梯度下降

- 梯度下降图示：



- 反向传播算法：针对前馈神经网络而设计的高效方法

$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \rightarrow \boldsymbol{z}^{(1)} \rightarrow \boldsymbol{a}^{(1)} \rightarrow \boldsymbol{z}^{(2)} \rightarrow \boldsymbol{a}^{(2)} \rightarrow \cdots \rightarrow \boldsymbol{a}^{(L-1)} \rightarrow \boldsymbol{z}^{(L)} \rightarrow \boldsymbol{a}^{(L)}$$

仿射变换 $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$　　$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)})$　非线性变换

$\boldsymbol{W}^{(l)}$的第k行

$$z_k^{(l)} = \langle \boldsymbol{w}_k^{(l)}, \boldsymbol{a}^{(l-1)} \rangle + b_k^{(l)}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}} = \langle \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\boldsymbol{z}^{(l)}}, \frac{\partial \boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}} \rangle$$

$$\frac{\partial z_k^{(l)}}{\partial \boldsymbol{w}_i^{(l)}} = \delta(k=i)\boldsymbol{a}^{(l-1)}$$

误差项 $\boldsymbol{e}^{(l)} = \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\boldsymbol{z}^{(l)}}$

$$\frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{b}^{(l)}} = \langle \frac{\partial \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\boldsymbol{z}^{(l)}}, \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} \rangle$$

$$\frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \mathbf{1}_{m^{(l)}}$$　单位阵

$$\boldsymbol{x}=\boldsymbol{a}^{(0)}\rightarrow\boldsymbol{z}^{(1)}\rightarrow\boldsymbol{a}^{(1)}\rightarrow\boldsymbol{z}^{(2)}\rightarrow\boldsymbol{a}^{(2)}\rightarrow\cdots\rightarrow\boldsymbol{a}^{(L-1)}\rightarrow\boldsymbol{z}^{(L)}\rightarrow\boldsymbol{a}^{(L)}$$

仿射变换 $\boldsymbol{z}^{(l)}=\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)}+\boldsymbol{b}^{(l)}$ $\boldsymbol{a}^{(l)}=f_l(\boldsymbol{z}^{(l)})$ ➡ $a_k=f_l\left(z_k^{(l)}\right)$

误差项 $\boldsymbol{e}^{(l)}=\dfrac{\partial\mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\boldsymbol{z}^{(l)}}$

$$\frac{\partial\boldsymbol{a}^{(l)}}{\partial\boldsymbol{z}^{(l)}}=\frac{\partial f_l(\boldsymbol{z}^{(l)})}{\partial\boldsymbol{z}^{(l)}}=\operatorname{diag}\left(\nabla f_l(\boldsymbol{z}^{(l)})\right)$$

$$=\frac{\partial\mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\boldsymbol{z}^{(l+1)}}\frac{\partial\boldsymbol{z}^{(l+1)}}{\partial\boldsymbol{a}^{(l)}}\frac{\partial\boldsymbol{a}^{(l)}}{\partial\boldsymbol{z}^{(l)}}$$

$$=\boldsymbol{e}^{(l+1)}\boldsymbol{W}^{(l+1)}\operatorname{diag}\left(\nabla f_l(\boldsymbol{z}^{(l)})\right)\qquad\frac{\partial\boldsymbol{z}^{(l+1)}}{\partial\boldsymbol{a}^{(l)}}=\boldsymbol{W}^{(l+1)}$$

$$=\nabla f_l(\boldsymbol{z}^{(l)})\odot(\boldsymbol{e}^{(l+1)}\boldsymbol{W}^{(l+1)})$$

$$\frac{\partial z_k^{(l)}}{\partial\boldsymbol{w}_i^{(l)}}=\delta(k=i)\boldsymbol{a}^{(l-1)}$$

$$\frac{\partial\mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial w_{ij}^{(l)}}=\langle\boldsymbol{e}^{(l)},\frac{\partial\boldsymbol{z}^{(l)}}{\partial w_{ij}^{(l)}}\rangle=e_i^{(l)}a_j^{(l-1)}\quad\Rightarrow\quad\frac{\partial\mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial\boldsymbol{W}^{(l)}}=\boldsymbol{e}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^{\mathrm{T}}$$

$$\frac{\partial\mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial\boldsymbol{b}^{(l)}}=\langle\boldsymbol{e}^{(l)},\frac{\partial\boldsymbol{z}^{(l)}}{\partial\boldsymbol{b}^{(l)}}\rangle=\boldsymbol{e}^{(l)}$$

$$\boldsymbol{e}^{(l)}=\nabla f_l(\boldsymbol{z}^{(l)})\odot(\boldsymbol{e}^{(l+1)}\boldsymbol{W}^{(l+1)})$$

- 前馈神经网络的训练过程可以分为以下三步：
  - 前向计算：每一层的状态和激活值，直到最后一层
  - 反向计算：每一层的参数的偏导数
  - 更新参数

实验内容

使用`pytorch`或者`tensorflow`手写一个前馈神经网络，用于近似正弦函数$y=\sin(x),x\in[0,4\pi)$。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。

即回归任务，输入和输出均为一维。

实验结果

实验使用`pytorch`进行。

源码结构及说明

模型结构

实现了可自由调整深度及各隐藏层的宽度的前馈神经网络，激活函数可以通过参数设置。

具体来说构建模型需要以下两个参数：

- `neurons`：列表类型，表示各网络层的神经元个数，包含输入层和输出层。在本实验中输入层和输出层神经元个数固定为 1。例如 `neurons=[1, 20, 10, 1]` 表示模型包含两个神经元个数分别为 20 和 10 的隐藏层。
- `activations`：字符串类型，指定要使用的激活函数，可供选择的有 `'sigmoid'`,`'tanh'`,`'relu'`,`'leakyrelu'`,`'prelu'`,`'elu'`,`'softplus'`。例如 `activation='relu'` 表示输入层及各隐藏层都将使用 `relu` 作为激活函数。因为是回归任务，输出层没有激活函数。

优化器及损失函数

优化器固定使用 `torch.nn.optim.Adam`，但学习率 `lr` 是可调节的参数。

训练的损失函数及测试的性能评估均固定使用 `torch.nn.MSELoss`

数据生成

使用 `numpy` 生成区间$[0, 4\pi)$上的均匀样本点作为训练及测试数据，可以控制总样本点数目及训练样本的比例。具体来说有如下三个参数：

- `data_size`：整型，表示生成的样本点个数。包含训练及测试的样本。
- `train_ratio`：浮点数，表示训练集的比例，训练集的大小将为 `round(data_size*train_ratio)`。训练集从生成的样本中随机选择。
- `random_state`：整型，随机种子，决定训练集测试集的划分。

训练及测试

固定批大小为10，训练10轮。即 `batch_size=10,epoch=10`.

结果及分析

本实验固定的参数如下：

```
default_params = {
    'data_size': 10000,
    'train_ratio': 0.8,
    'random_state': 7,
    'epochs': 10,
    'batch_size': 10
}
```

值得注意的是，这里的固定参数也就是说**以下实验的训练数据量及训练轮数都是一样的，而不是各自充分训练的结果。**

调节的参数有：

- `neurons`：研究不同网络深度及宽度对模型性能的影响。
- `activation`：研究不同激活函数对模型性能的影响。
- `lr`：研究不同学习率对模型性能的影响。

比较一：网络深度与宽度

固定参数 `activation=tanh`，`lr=0.001`。

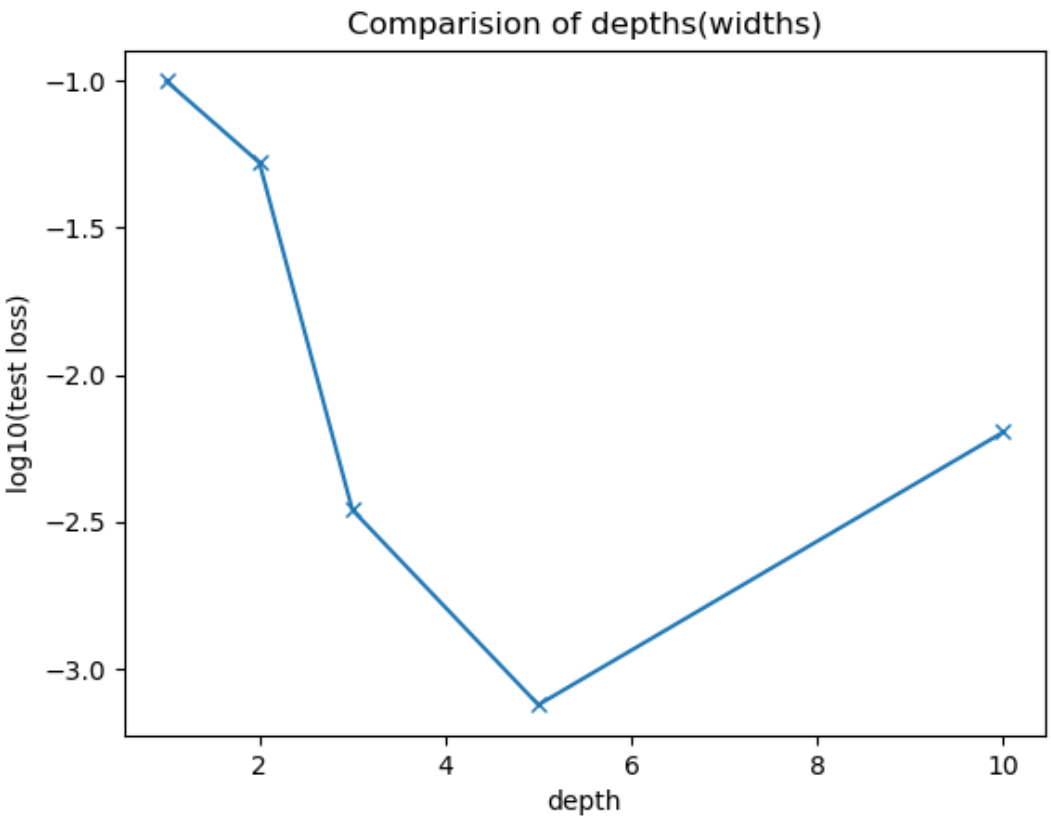
改变参数 `neurons` 设置五组对照，在固定网络参数保持在1000左右的情况下设置5个不同的深度，并保持各层宽度一致（相应地也是五个不同的宽度）。注意：这里深度为隐藏层的个数，不考虑输入输出层。具体设置及各设置下模型的参数个数如下：

```
# setup1: neurons=[1, 333, 1] (depths=1)       #(params)=1000
# setup2: neurons=[1, *[30]*2, 1] (depths=2)   #(params)=1021
# setup3: neurons=[1, *[21]*3, 1] (depths=3)   #(params)=988
# setup4: neurons=[1, *[15]*5, 1] (depths=5)   #(params)=1006
# setup5: neurons=[1, *[10]*10, 1] (depths=10) #(params)=1021
```

对上述五种设置，独立地训练并测试10次得到10次测试的loss，为避免极端值的影响，去掉最高值和最低值后取平均，作为评价不同设置下模型性能的标准。结果为：

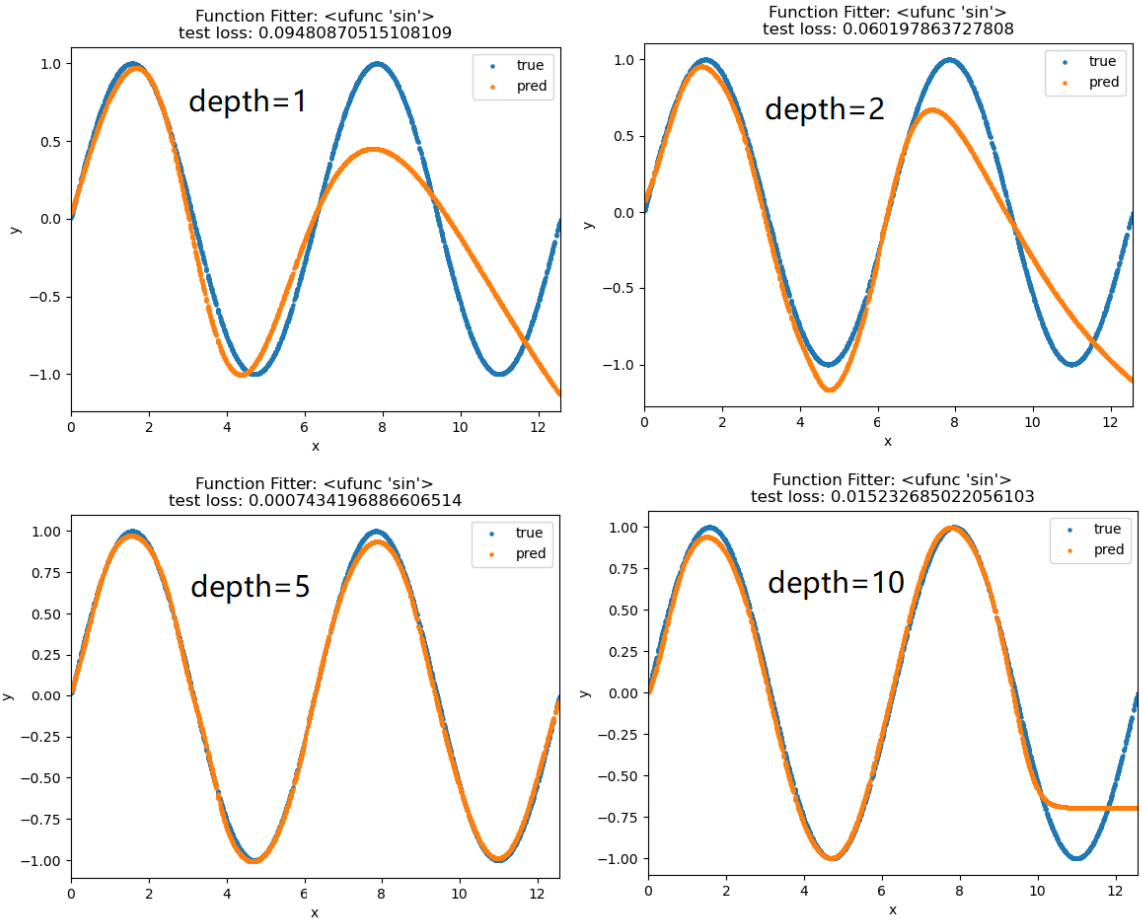
```
{1: 0.09975287499999999, 2: 0.052726499999999996, 3: 0.0034785, 5: 0.00075625,
10: 0.006392500000000001} # 键为深度，值为loss
```

对loss取对数并作图如下：



可以看到，在保持网络参数不变的情况下，随着网络深度的加深（对应地，网络宽度减小），同样的训练条件下，模型的性能可能会慢慢变好，但当网络过深时，模型的性能也会下降，这可能是由于梯度消失，优化过程比浅层的网络更难。

细节的拟合效果可以观察拟合图：



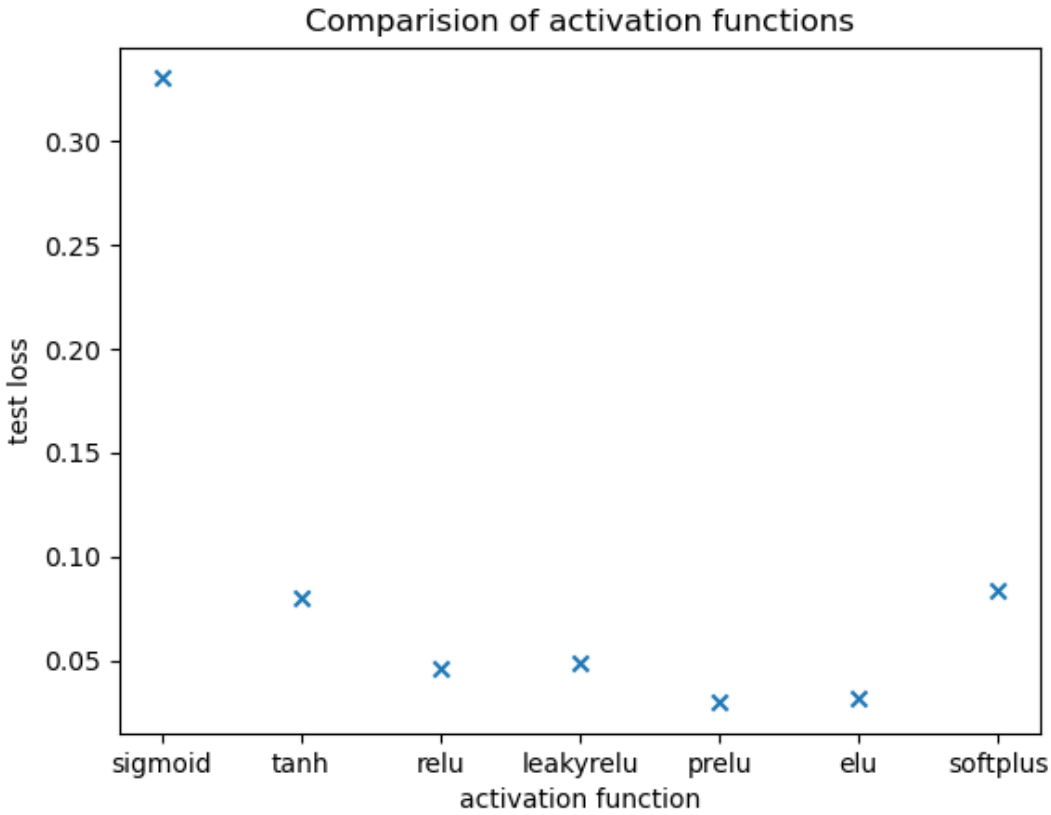
可以注意到当深度较大时（depth=10），x较小的部分的拟合效果还不错，但当数值较大靠近尾部时，突然出现很大偏差，可能是数值太大时，激活函数的梯度很小，而深层网络会进一步扩大这个影响（也就是所谓的梯度消失问题），导致参数几乎无法更新。

比较二：激活函数

固定参数`neurons=[1, 20, 20, 1]`,`lr=0.001`

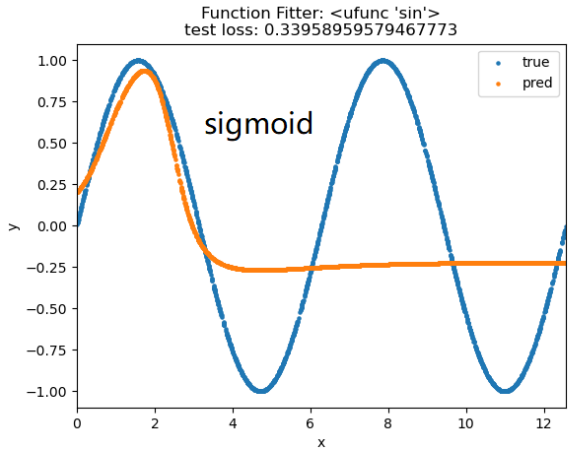
在 7 种激活函数的选择下，分别独立地训练并测试10次得到10次的测试loss，为避免极端值的影响，去掉最高值和最低值后取平均，作为评价不同设置下模型性能的标准。结果如下：

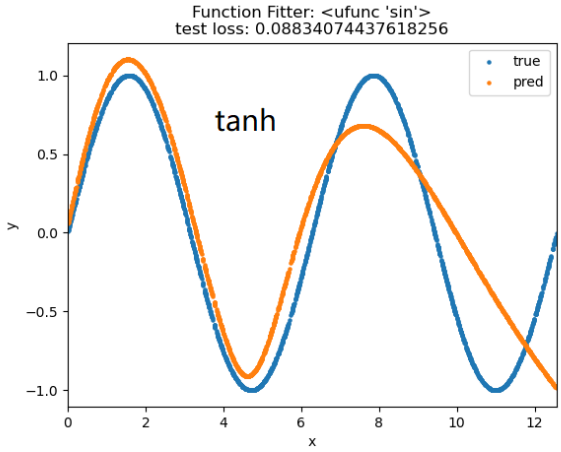
```
{'sigmoid': 0.33004449999999996, 'tanh': 0.07989337499999999, 'relu':
0.046223875, 'leakyrelu': 0.049304, 'prelu': 0.029966624999999997, 'elu':
0.032395125, 'softplus': 0.08388562499999999}
```

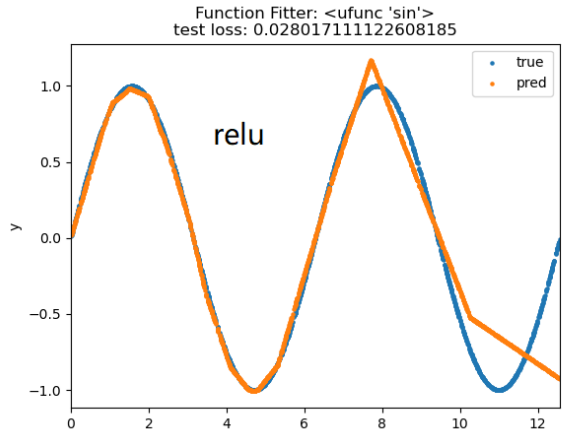


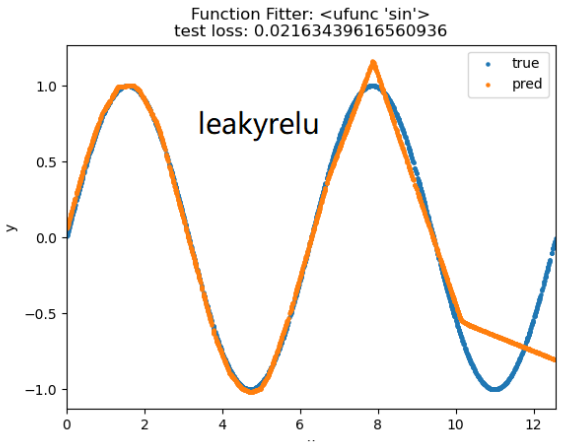
可见`sigmoid`激活函数效果非常差，`tanh`和`softplus`效果较好但不如`relu`家族。而`relu`家族中`prelu`和`elu`效果最好。这可能是因为`prelu,elu`均包含额外的可学习参数。

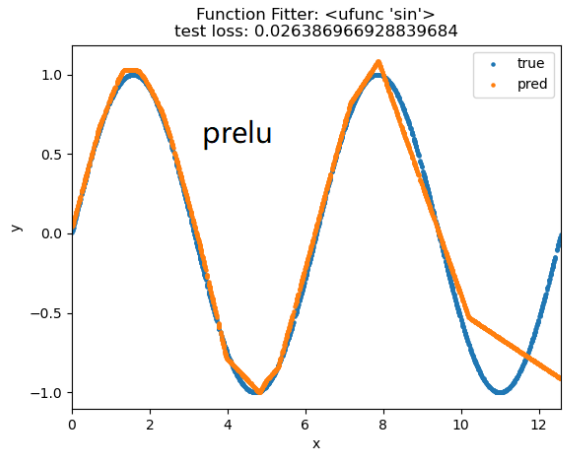
除了比较测试loss外，我们还可以观察拟合图，分析各激活函数的特点。

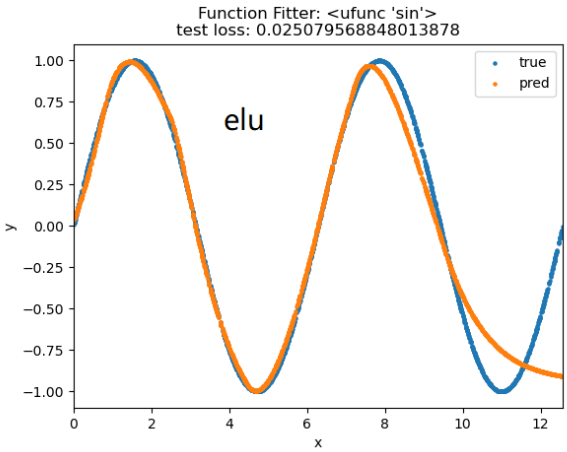


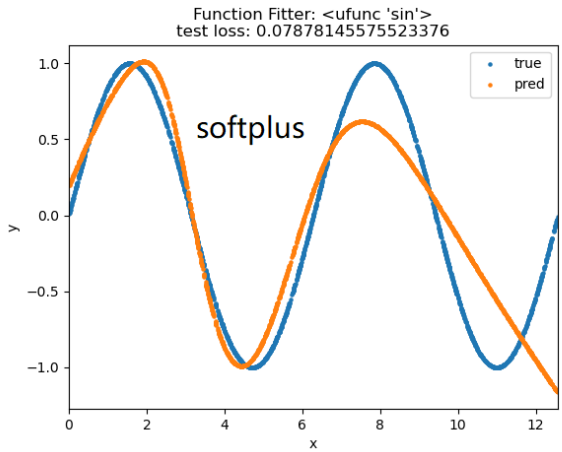












分析拟合图可以看出：

- `sigmoid`出现了类似于深层网络所有的梯度消失的问题，即当x较大时，模型性能急剧变差。一方面该激活函数确实存在饱和的问题，另一方面该激活函数是非零中心化的，会使得后一层

的神经元的输入发生偏置偏移，使得梯度下降的收敛速度变慢。

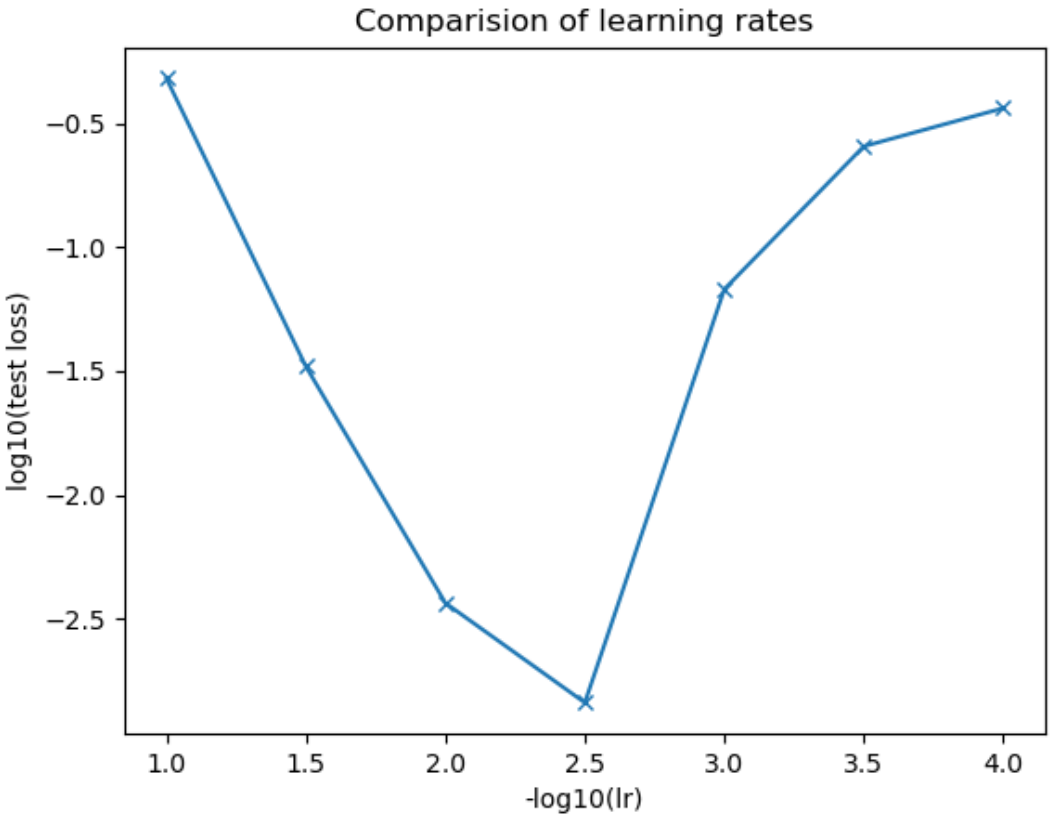
- `relu,leakyrelu,prelu` 的拟合图均出现**不光滑的现象**，这是由于激活函数本身不光滑导致的。而 `elu` 本身是光滑的，没有出现类似的现象。它们都不会像 `sigmoid,tanh` 那样有梯度饱和的问题，因而学习效率更高。

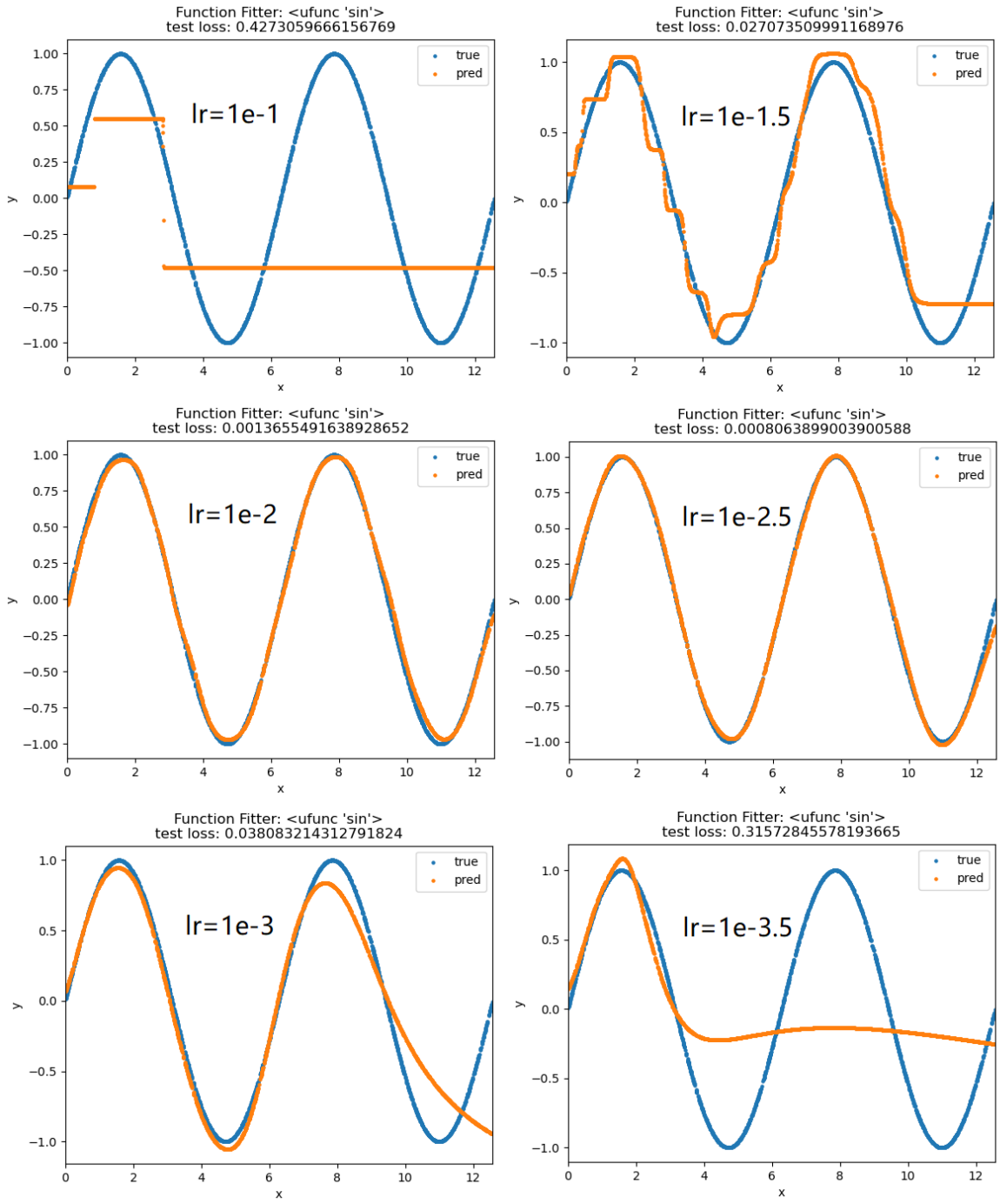
比较三：学习率

固定参数 `neurons=[1, 20, 20, 1]`,`activation='tanh'`

设置七个不同的学习率：$10^{-k}, k = 1, 1.5, 2, 2.5, 3, 3.5, 4$，分别独立地训练并测试10次得到10次的测试loss，为避免极端值的影响，去掉最高值和最低值后取平均，作为评价不同设置下模型性能的标准。结果如下：

```
{0.1: 0.17341449586998198, 0.03162277660168379: 0.0337293358056455, 0.01:
0.01761509341899724, 0.0031622776601683794: 0.0014473155011952299, 0.001:
0.04821309684815943, 0.00031622776601683794: 0.1052767455255433, 0.0001:
0.011405165033439895}
```





可见随着学习率慢慢变大，模型性能先变好后变差。

具体来说，可以从拟合图上得到一些结论：

- 当学习率太大时，参数更新步长较大，导致目标函数波动较大，会使得收敛速度较慢，并且在结果上出现类似阶跃函数输出值突变的现象。
- 当学习率太小时，参数更新步长较小，导致目标函数减小地很慢，收敛速度较慢，需要更长的训练时间才能达到较好地收敛。

实验总结

本实验中，实现了前馈神经网络并用它来拟合初等函数。

通过三个比较实验，初步了解了网络深度和宽度，激活函数，学习率对前馈神经网络性能的影响。

对于网络深度和宽度而言，深层的网络收敛速度更快，但会引发梯度消失的问题；

对于激活函数而言，`relu`及其变种是很好的选择，但因为不光滑的特点会导致输出不光滑（`elu`不存在这样的问题）。

对于学习率而言，学习率太大和太小都不利于学习，针对特定的问题应该选择大小适中的学习率。