第一次实验报告

PB21020651 武宇星

关键代码

先将除了激活函数外的超参数单独定义出来，方便在后续调试参数过程中，只需更改这几个值即可，其中 epoch 一直固定为 100

```
N = 10000
width = 200
depth = 9
learning_rate = 0.00006
epochs = 100
```

这部分为根据样本数量要求均匀采样 x，计算 y 构成数据集并随机划分成互不相交的训练集，验证集和测试集

```
# 数据集生成
x = torch.linspace(1, 16, steps = N).unsqueeze(1)  # 生成输入数据
y = torch.log2(x) + torch.cos(torch.pi * x / 2)  # 计算目标输出
dataset = TensorDataset(x, y)
# 划分数据集
train_size = int(N * 0.8)
val_size = int(N * 0.1)
test_size = N - train_size - val_size
train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

这部分为前馈神经网络的模型定义部分，宽度 width 传入 hidden_size,深度 depth 传给 hidden_layers，在 self.activate 处更改使用的激活函数

```
class FeedforwardNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, hidden_layers):
        super(FeedforwardNN, self).__init__()
        layers = [nn.Linear(input_size, hidden_size)]
        for _ in range(hidden_layers - 1):
            layers.append(nn.Linear(hidden_size, hidden_size))
        layers.append(nn.Linear(hidden_size, output_size))
        self.layers = nn.ModuleList(layers)
        self.activate = nn.ReLU()

    def forward(self, x):
        for layer in self.layers[:-1]:
            x = self.activate(layer(x))
        x = self.layers[-1](x)
        return x
```

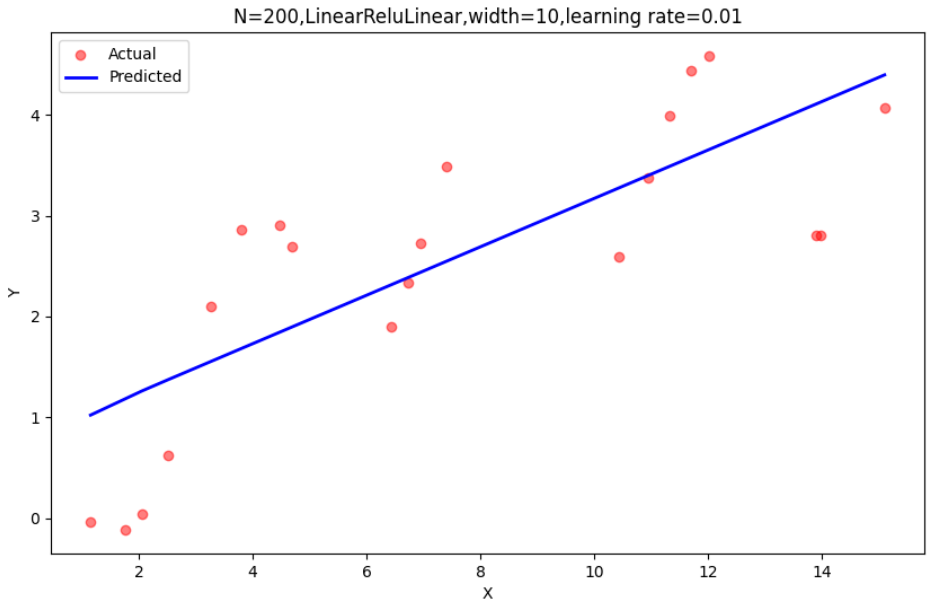
这部分定义优化器类型和学习率 learning_rate 传给 lr，并把损失函数定义为要求的 MSE，因

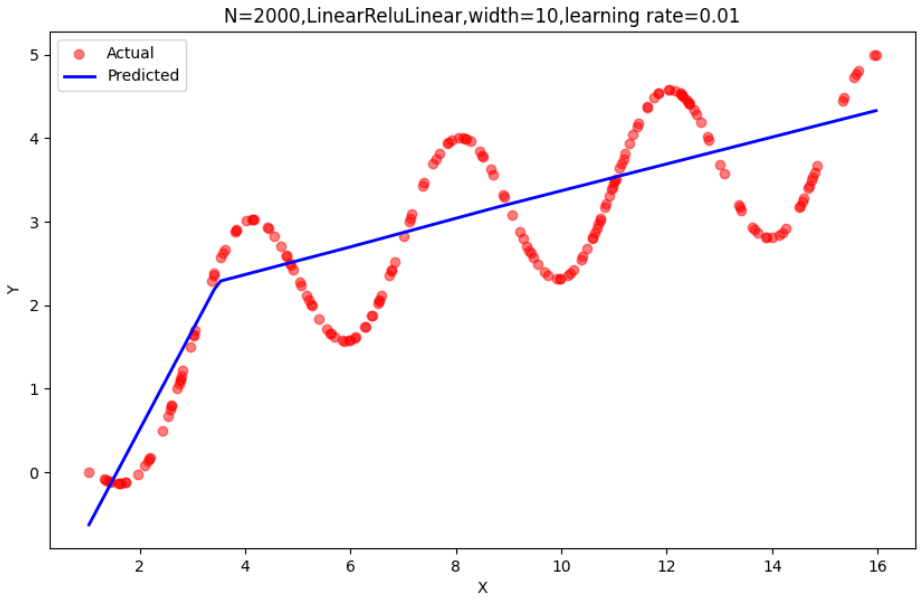
为使用服务器跑的代码所以也指定了运行的 GPUid

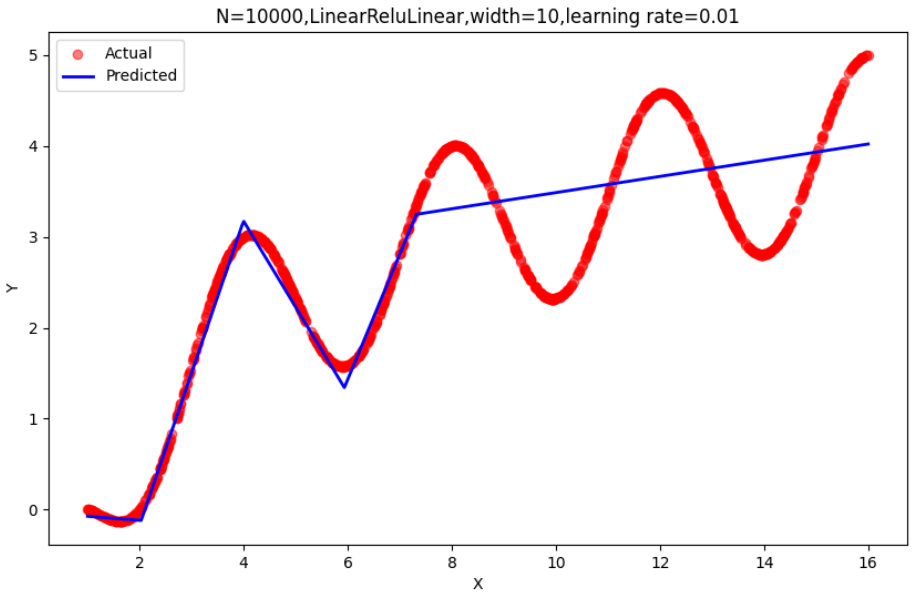
```
model = FeedforwardNN(input_size=1, hidden_size=width, output_size=1, hidden_layers=depth)
# print(model)
device = "cuda:1" if torch.cuda.is_available() else "cpu"
model.to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

实验过程

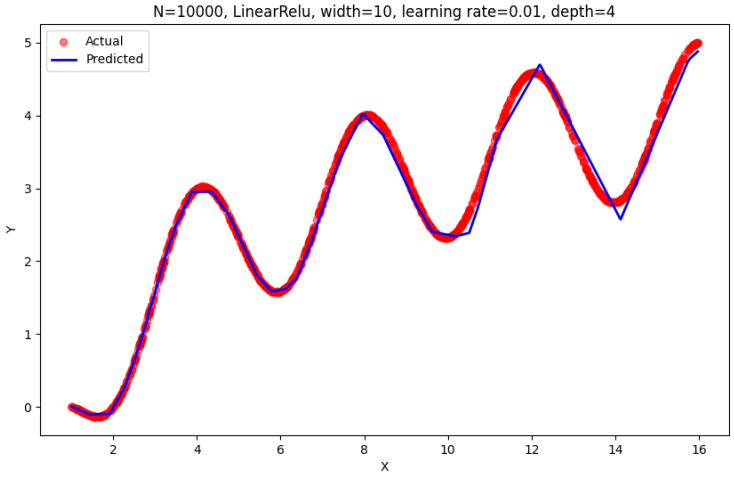
首先从样本数调起，显然样本更多训练效果更好，故后续固定 N=10000

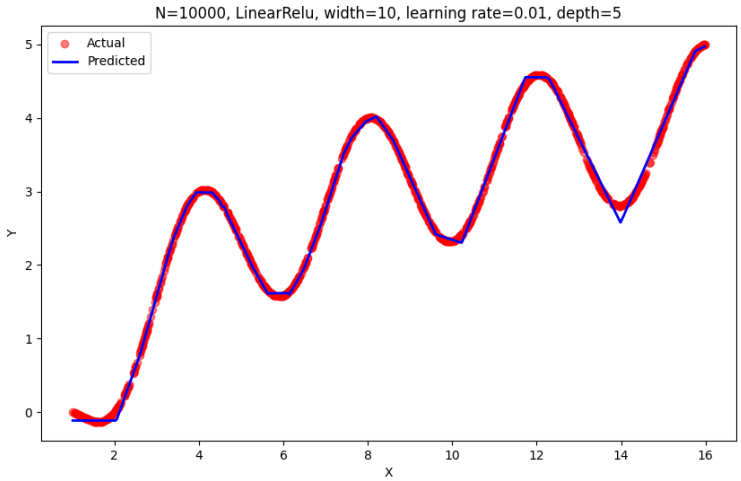


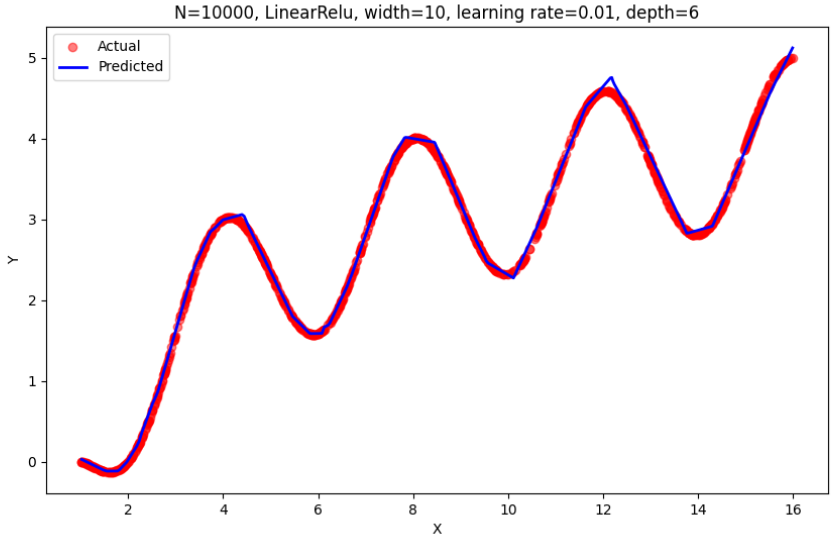


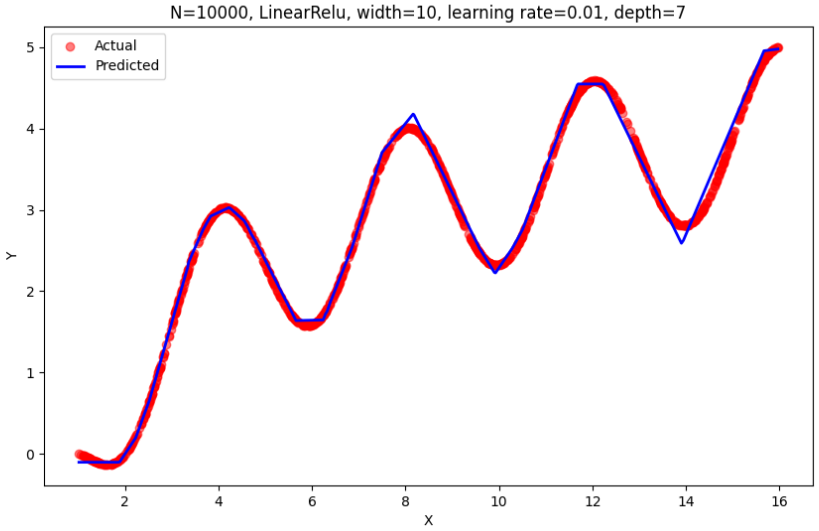


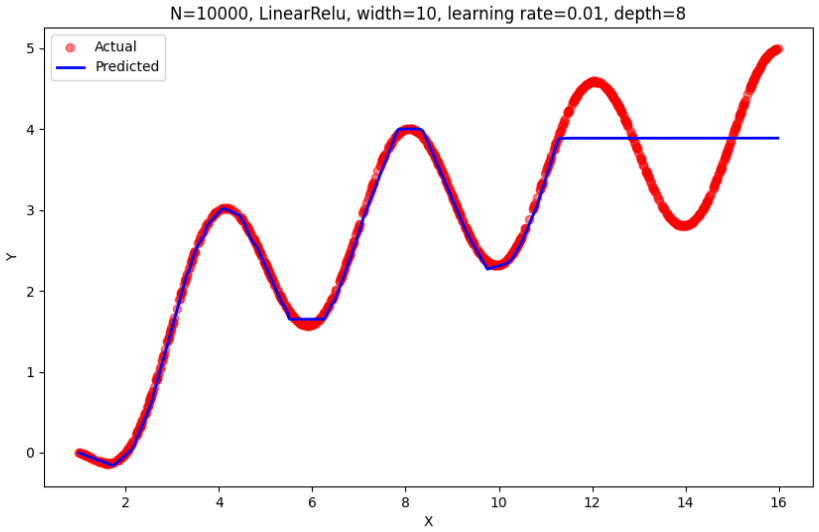
然后调整网络深度，深度今在 4-9 之间尝试了，发现较小时并无太大差别，但 8，9 明显拟合的很差，固定参数为 6

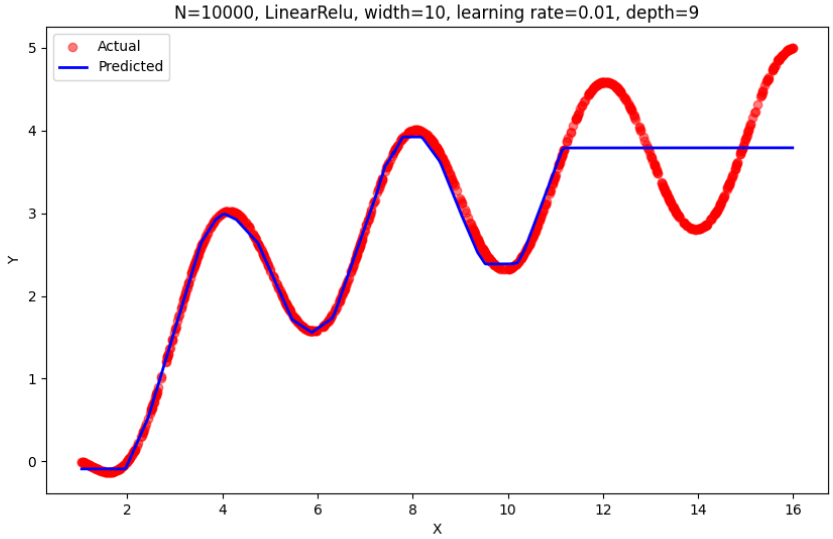




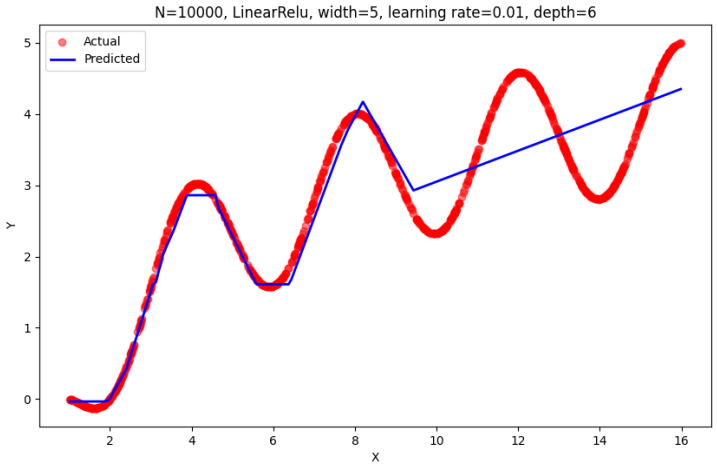

N=10000, LinearRelu, width=10, learning rate=0.01, depth=6
Actual
Predicted
X
Y

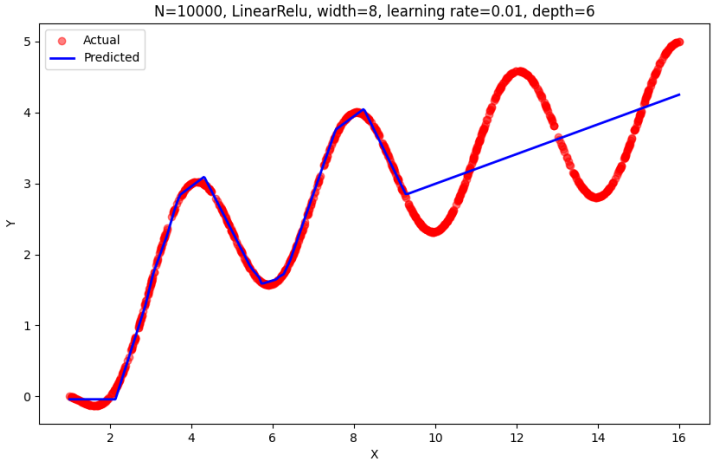

N=10000, LinearRelu, width=10, learning rate=0.01, depth=7
Actual
Predicted
X
Y

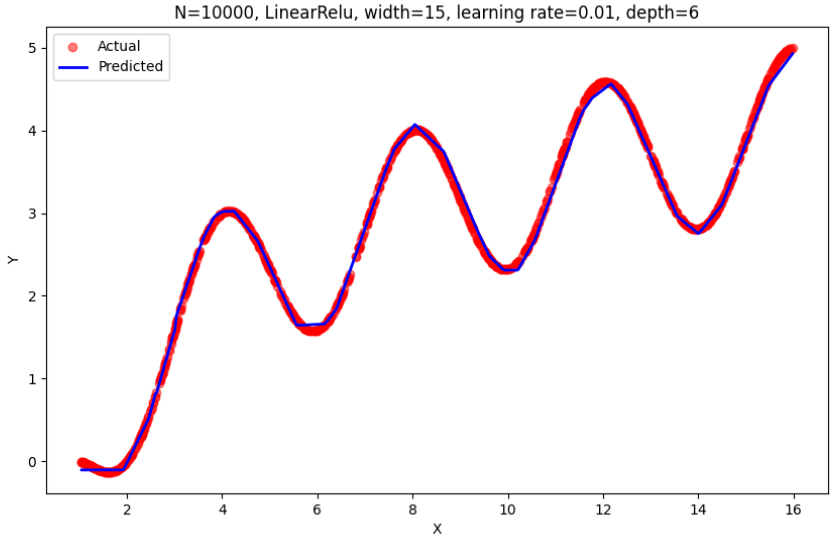

N=10000, LinearRelu, width=10, learning rate=0.01, depth=8
Actual
Predicted
X
Y

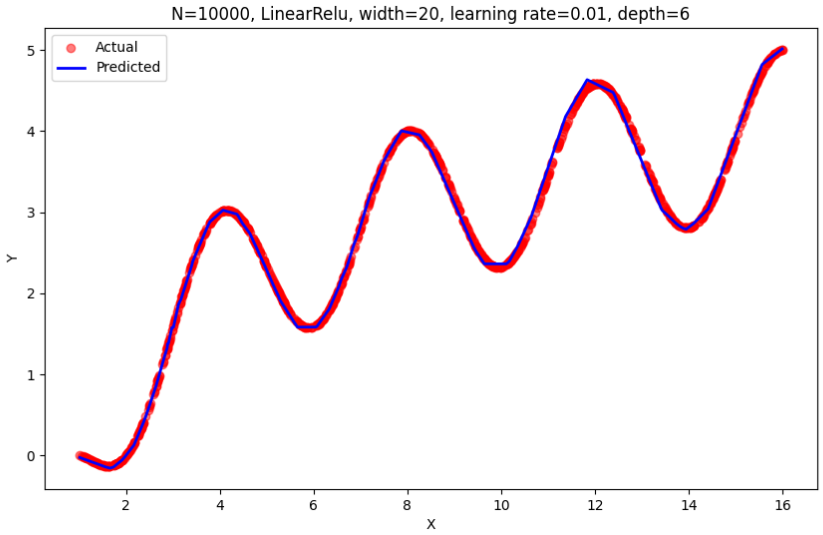


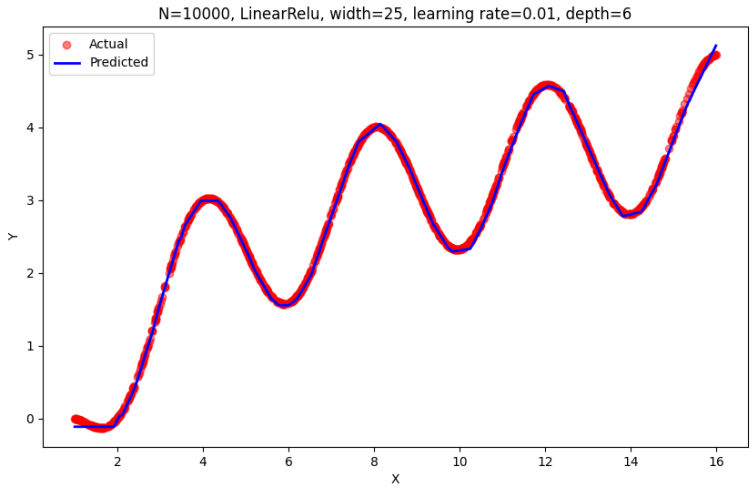
接着调整网络宽度，可以发现当神经网络宽度小时，拟合函数的能力很差，但当取得过大的时候会发现，局部样本反而会出现估计不准的情况，故将宽度固定为 200

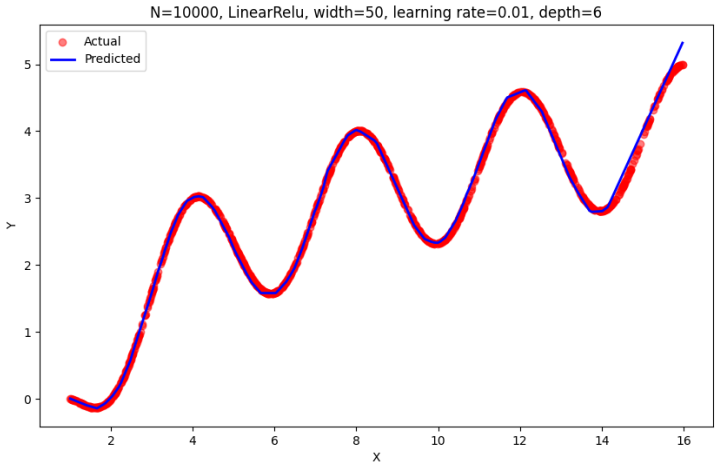


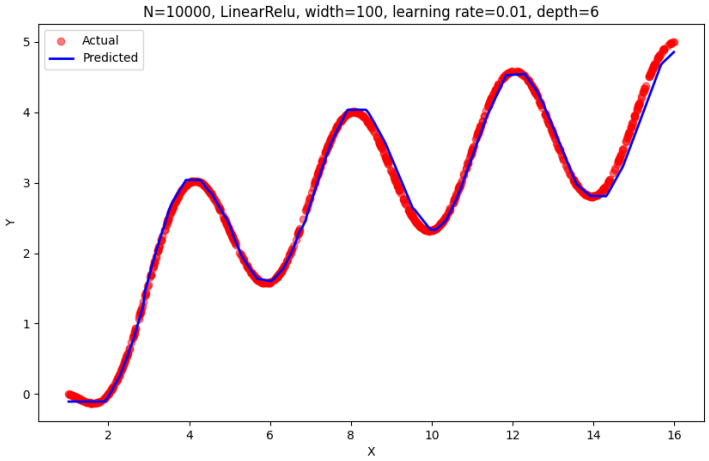


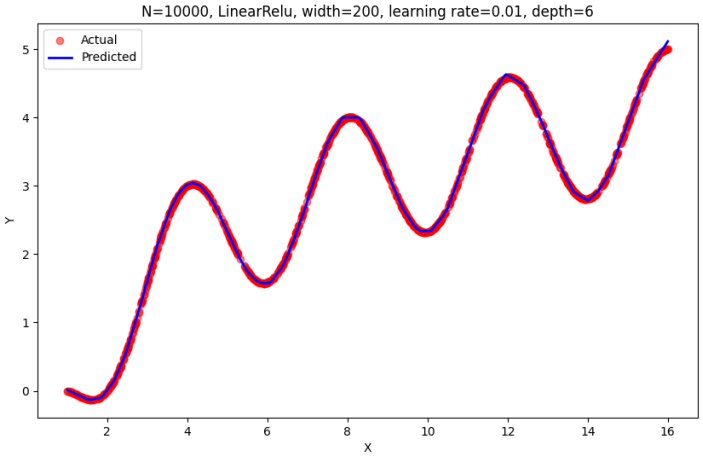

N=10000, LinearRelu, width=15, learning rate=0.01, depth=6
Actual
Predicted
X
Y

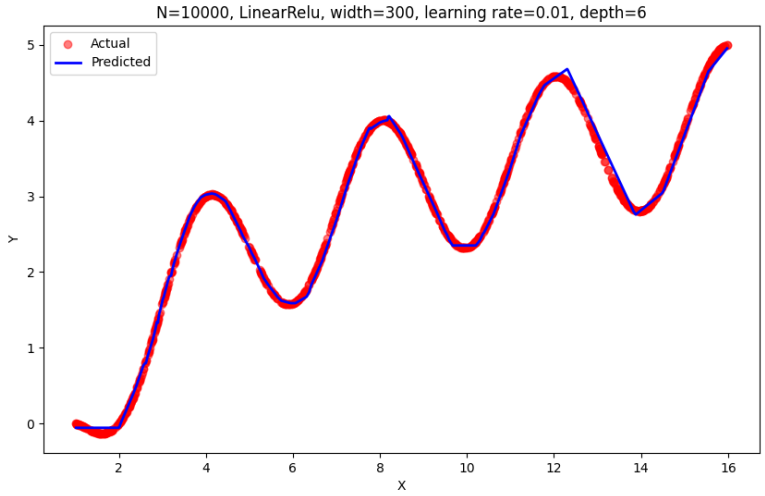

N=10000, LinearRelu, width=20, learning rate=0.01, depth=6
Actual
Predicted
X
Y

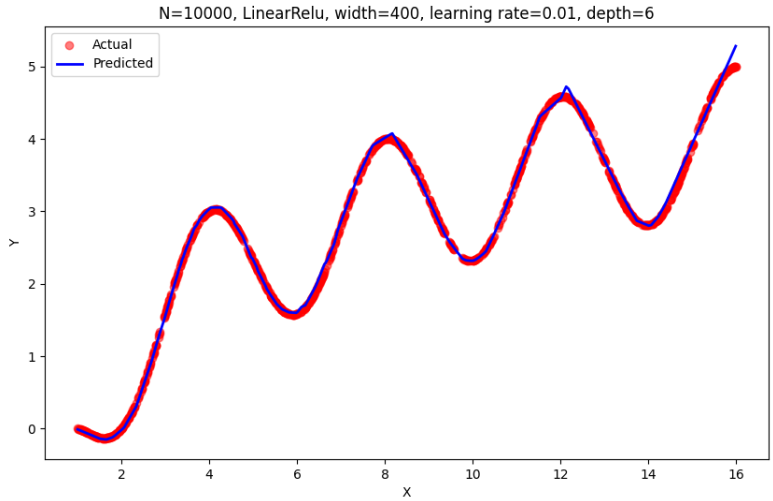

N=10000, LinearRelu, width=25, learning rate=0.01, depth=6
Actual
Predicted
X
Y


N=10000, LinearRelu, width=50, learning rate=0.01, depth=6
Actual
Predicted
X
Y

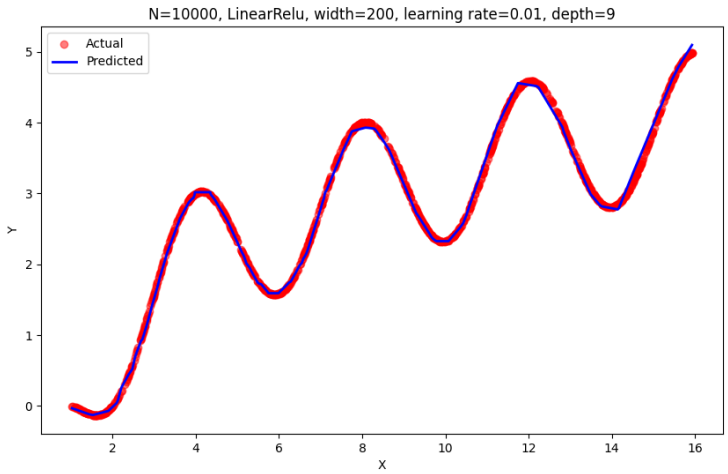

N=10000, LinearRelu, width=100, learning rate=0.01, depth=6
Actual
Predicted
X
Y

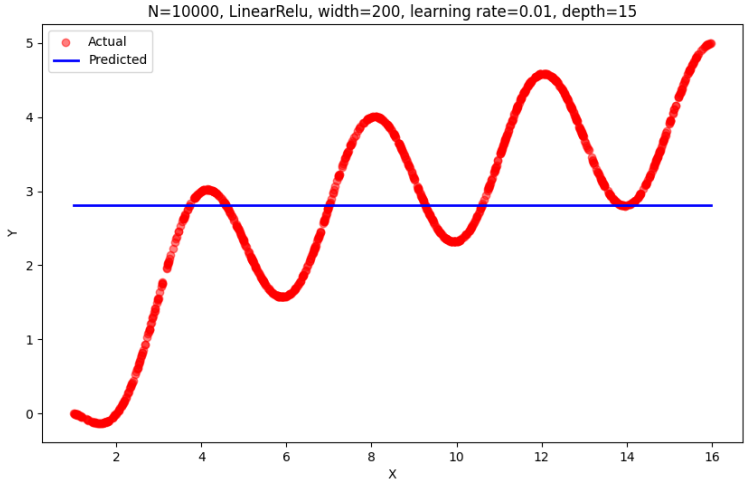

N=10000, LinearRelu, width=200, learning rate=0.01, depth=6
Actual
Predicted
X
Y





在调完宽度和深度后我进一步尝试了在宽度提升后是否更深的深度能有更好的性能，发现 depth=9 时性能最好，到 10 会一下子拟合成直线，给我检查的学姐告诉我是因为调参初始的学习率设为 0.01 太大导致损失函数震荡。





对于学习率的调参部分，我发现由于大部分参数已经调好了，导致经过 100 个 epoch 都训练的很好，故观察损失函数，挑选最后损失函数小且收敛快的，最后选定为 0.00006

lr0.0001

```
(py39) wuyux@Sui4:~/dl/pytorch/exp1$ python3 exp1.py
Epoch [10/100], Train Loss: 0.3931, Val Loss: 0.4178
Epoch [20/100], Train Loss: 0.2624, Val Loss: 0.2448
Epoch [30/100], Train Loss: 0.0348, Val Loss: 0.0105
Epoch [40/100], Train Loss: 0.0013, Val Loss: 0.0003
Epoch [50/100], Train Loss: 0.0008, Val Loss: 0.0001
Epoch [60/100], Train Loss: 0.0022, Val Loss: 0.0001
Epoch [70/100], Train Loss: 0.0027, Val Loss: 0.0014
Epoch [80/100], Train Loss: 0.0006, Val Loss: 0.0007
Epoch [90/100], Train Loss: 0.0000, Val Loss: 0.0001
Epoch [100/100], Train Loss: 0.0001, Val Loss: 0.0001
```

lr0.00001

```
(py39) wuyux@Sui4:~/dl/pytorch/exp1$ python3 exp1.py
Epoch [10/100], Train Loss: 0.5892, Val Loss: 0.5931
Epoch [20/100], Train Loss: 0.4049, Val Loss: 0.4045
Epoch [30/100], Train Loss: 0.4005, Val Loss: 0.3974
Epoch [40/100], Train Loss: 0.3982, Val Loss: 0.3959
Epoch [50/100], Train Loss: 0.3912, Val Loss: 0.3864
Epoch [60/100], Train Loss: 0.3874, Val Loss: 0.3749
Epoch [70/100], Train Loss: 0.3781, Val Loss: 0.3661
Epoch [80/100], Train Loss: 0.3695, Val Loss: 0.3572
Epoch [90/100], Train Loss: 0.3599, Val Loss: 0.3506
Epoch [100/100], Train Loss: 0.3408, Val Loss: 0.3359
```

lr0.00008:

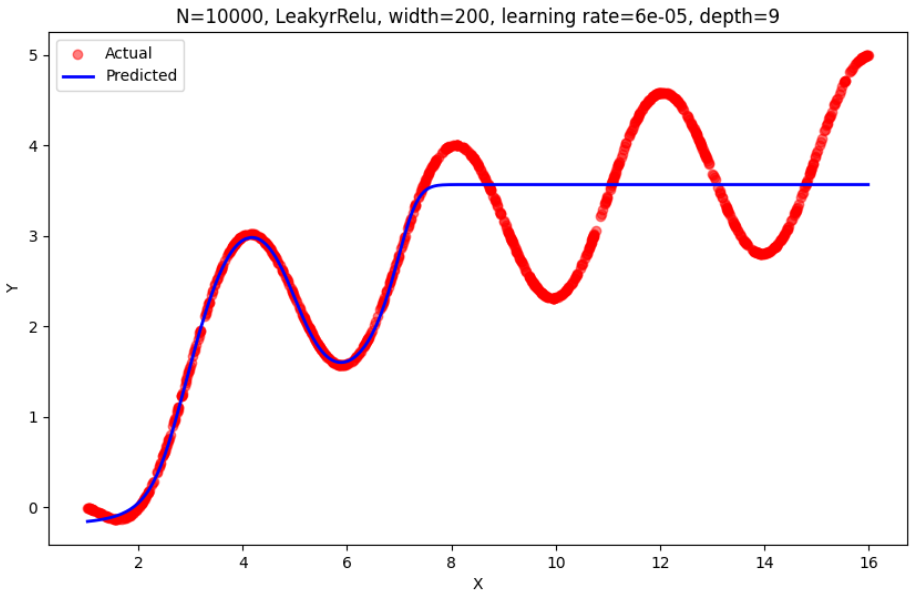
```
(py39) wuyux@Sui4:~/dl/pytorch/exp1$ python3 exp1.py
Epoch [10/100], Train Loss: 0.3943, Val Loss: 0.3937
Epoch [20/100], Train Loss: 0.2274, Val Loss: 0.2022
Epoch [30/100], Train Loss: 0.0011, Val Loss: 0.0006
Epoch [40/100], Train Loss: 0.0067, Val Loss: 0.0002
Epoch [50/100], Train Loss: 0.0003, Val Loss: 0.0003
Epoch [60/100], Train Loss: 0.0005, Val Loss: 0.0020
Epoch [70/100], Train Loss: 0.0007, Val Loss: 0.0007
Epoch [80/100], Train Loss: 0.0124, Val Loss: 0.0004
Epoch [90/100], Train Loss: 0.0016, Val Loss: 0.0000
Epoch [100/100], Train Loss: 0.0002, Val Loss: 0.0000
```

lr0.00006:

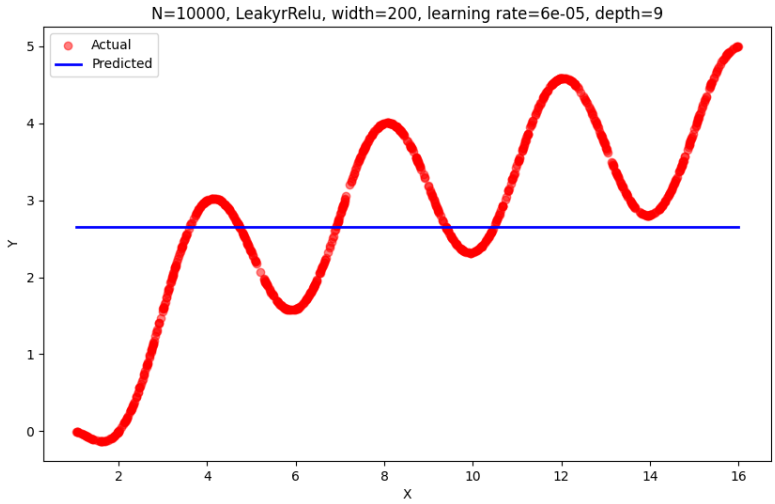
```
(py39) wuyux@Sui4:~/dl/pytorch/exp1$ python3 exp1.py
Epoch [10/100], Train Loss: 0.4108, Val Loss: 0.4068
Epoch [20/100], Train Loss: 0.3804, Val Loss: 0.3711
Epoch [30/100], Train Loss: 0.2523, Val Loss: 0.2387
Epoch [40/100], Train Loss: 0.0130, Val Loss: 0.0065
Epoch [50/100], Train Loss: 0.0011, Val Loss: 0.0026
Epoch [60/100], Train Loss: 0.0008, Val Loss: 0.0017
Epoch [70/100], Train Loss: 0.0000, Val Loss: 0.0001
Epoch [80/100], Train Loss: 0.0015, Val Loss: 0.0005
Epoch [90/100], Train Loss: 0.0000, Val Loss: 0.0000
Epoch [100/100], Train Loss: 0.0001, Val Loss: 0.0003
```

最后是激活函数的调整，softmax 和 sigmoid 效果都很差，上网查询似乎是因为这两个函数容易梯度消失，最后测试集选用 Relu

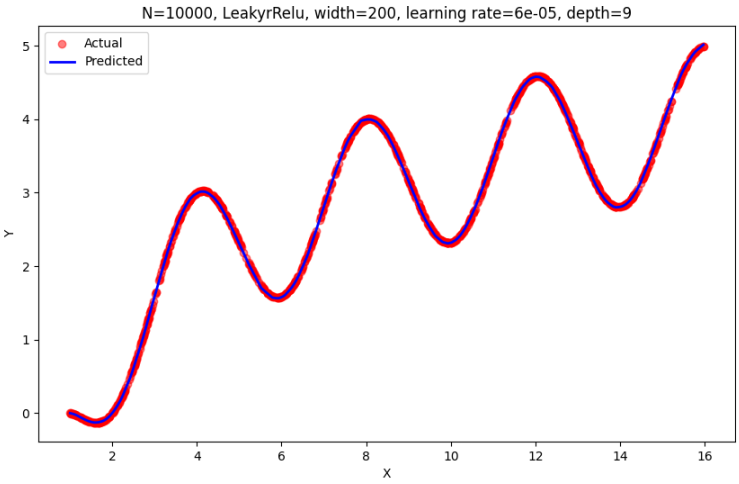
Sigmoid:（跑的时候忘记改图片激活函数名了）



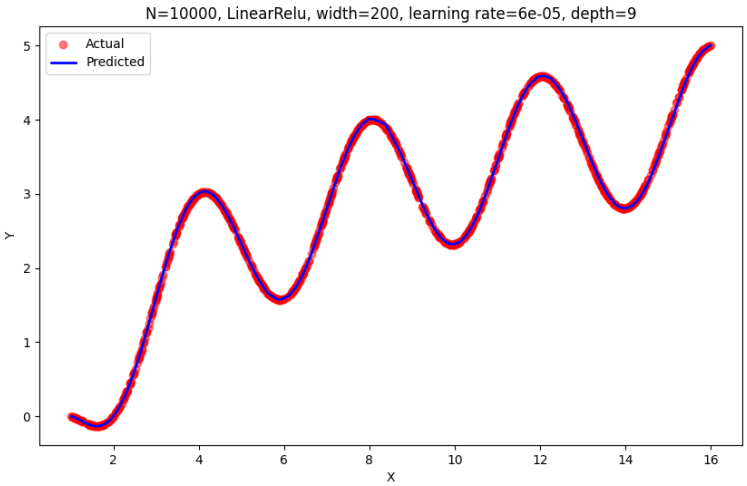
Softmax:



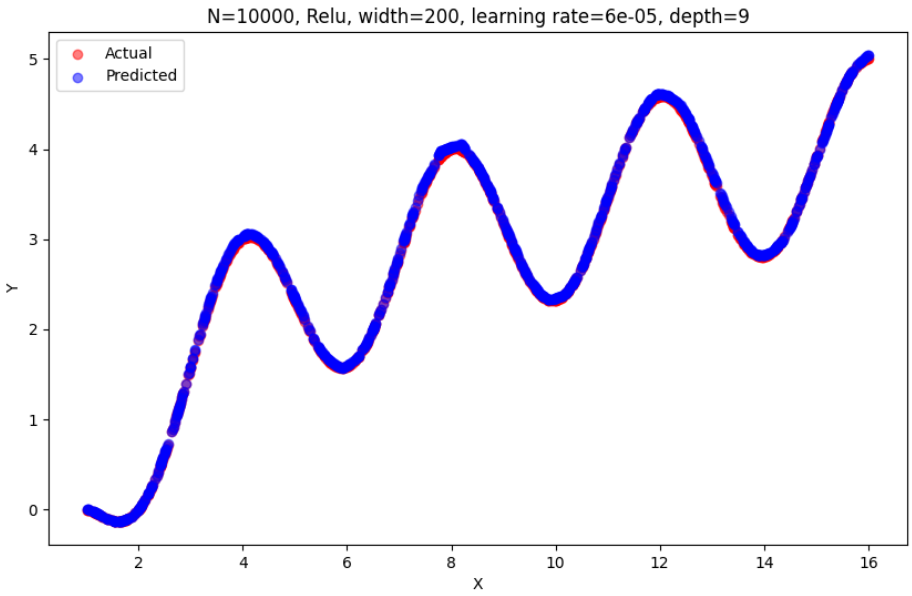
LeakyRelu：

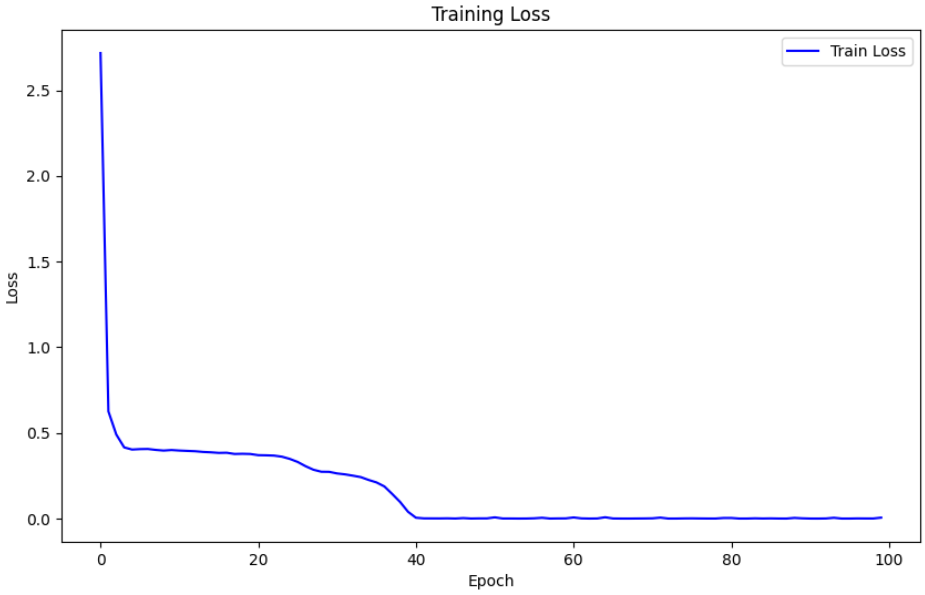


Relu 即原本调好参的结果：



最后测试集上测试部分根据助教的建议把预测值和实际值均用 scatter 绘制：


N=10000, Relu, width=200, learning rate=6e-05, depth=9
Actual
Predicted
Y
X
0
1
2
3
4
5
2
4
6
8
10
12
14
16


Training Loss
Train Loss
Loss
Epoch
0.0
0.5
1.0
1.5
2.0
2.5
0
20
40
60
80
100