实验一实验报告

实验过程

1. 首先选择pytorch作为网络框架，并下载对应的GPU版本

    ```
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    ```

2. 创建训练，验证，测试集，按照8:1:1分配并确定组成
3. 模型搭建

    ```
    class FeedForwardNN(nn.Module):
        def __init__(self):
            super(FeedForwardNN, self).__init__()
            self.fc1 = nn.Linear( in_features: 1,  out_features: 64)
            self.fc2 = nn.Linear( in_features: 64,  out_features: 64)
            self.fc3 = nn.Linear( in_features: 64,  out_features: 64)
            #self.fc4 = nn.Linear(64, 64)
            self.fc4 = nn.Linear( in_features: 64,  out_features: 1)
            self.relu = nn.ELU()

        def forward(self, x):
            x = self.relu(self.fc1(x))
            x = self.relu(self.fc2(x))
            x = self.relu(self.fc3(x))
            #x = self.relu(self.fc4(x))
            x = self.fc4(x)
            x = x.squeeze(-1)
            return x
    ```

    选择全连接层，网络深度为4，宽度为64，采用ELU作为激活函数
4. 模型训练

```python
def train(model, train_loader, optimizer, criterion, epochs=100):
    model.train()
    for epoch in range(epochs):
        for inputs, targets in train_loader:
            inputs = inputs.to(device)
            targets = targets.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()
```

将数据从train_loader中批次取出，移动到GPU上。将模型参数的梯度清零，输入到模型，计算输出。根据输出计算损失值，进而得到参数梯度。根据优化器优化参数，减少损失

5. 利用 matplotlib 绘图
6. 运行程序观察图像与损失值，进行调参分析

调参分析

绘图以n=2000为例

前面几个参数模型为relu，训练轮数100，虽然没有收敛但是可以更直观看出其余参数对于图像产生的影响

网络深度：

3个全连接层时

```
Validation MSE for N=200: 0.3365563154220581
Validation MSE for N=2000: 0.3455285727977527
Validation MSE for N=10000: 0.0024750658776611O9
```

4个全连接层时

```
Validation MSE for N=200: 0.2990545928478241
Validation MSE for N=2000: 0.13346628844738007
Validation MSE for N=10000: 0.0010245888261124492
```

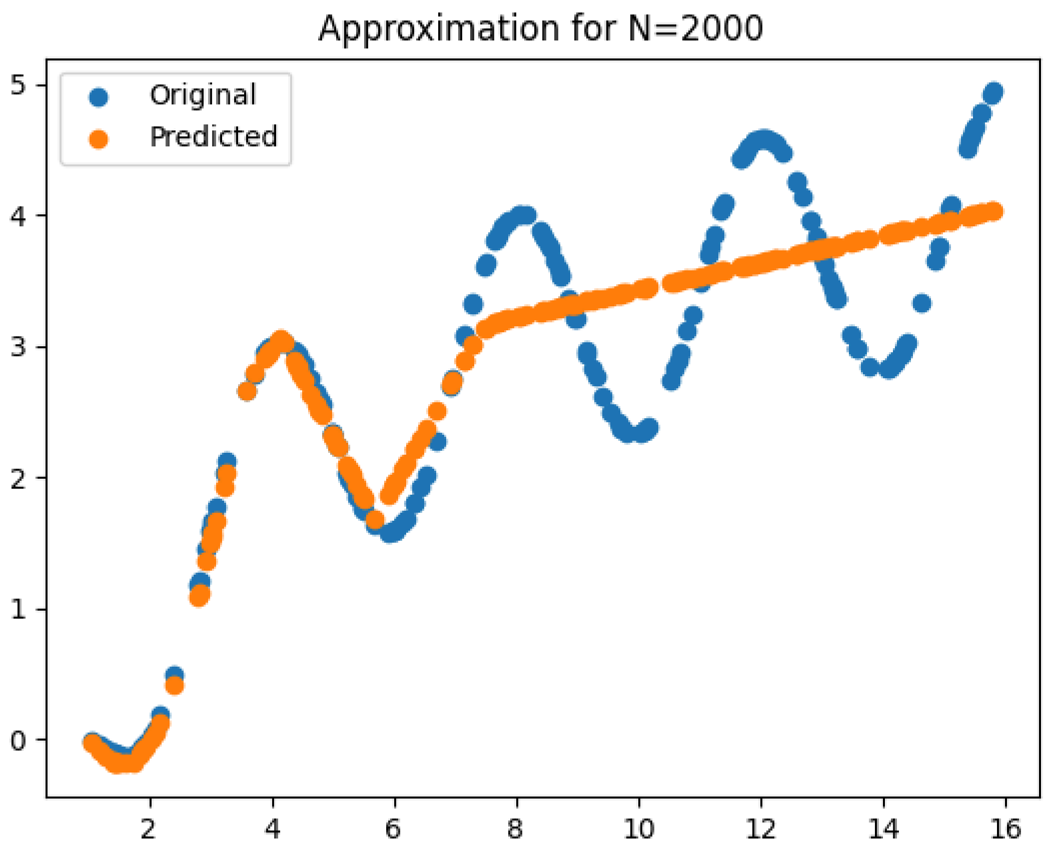
5个全连接层时

```
Validation MSE for N=200: 0.3367709517478943
Validation MSE for N=2000: 0.0524495467543602
Validation MSE for N=10000: 0.0009454597020521 76
```

> 因此,在具有一定数据量的情况下,提升网络深度对于模型的效果可以有效提升

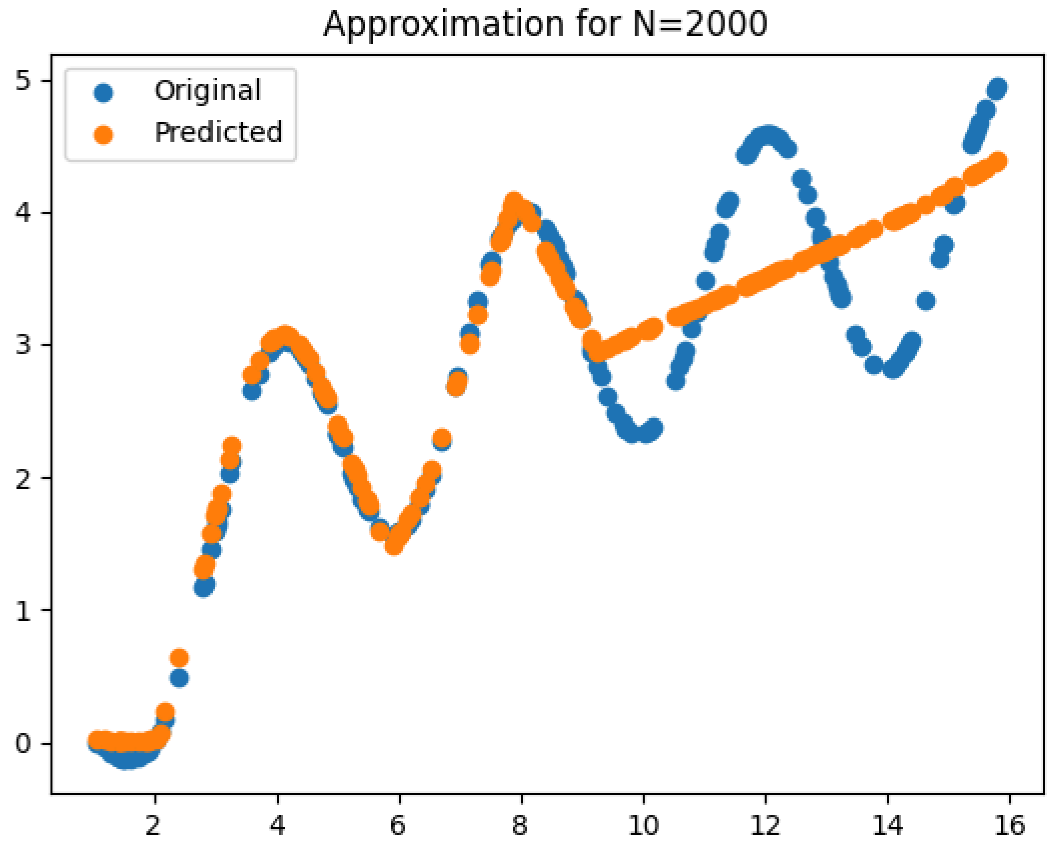
学习率

lr=0.001



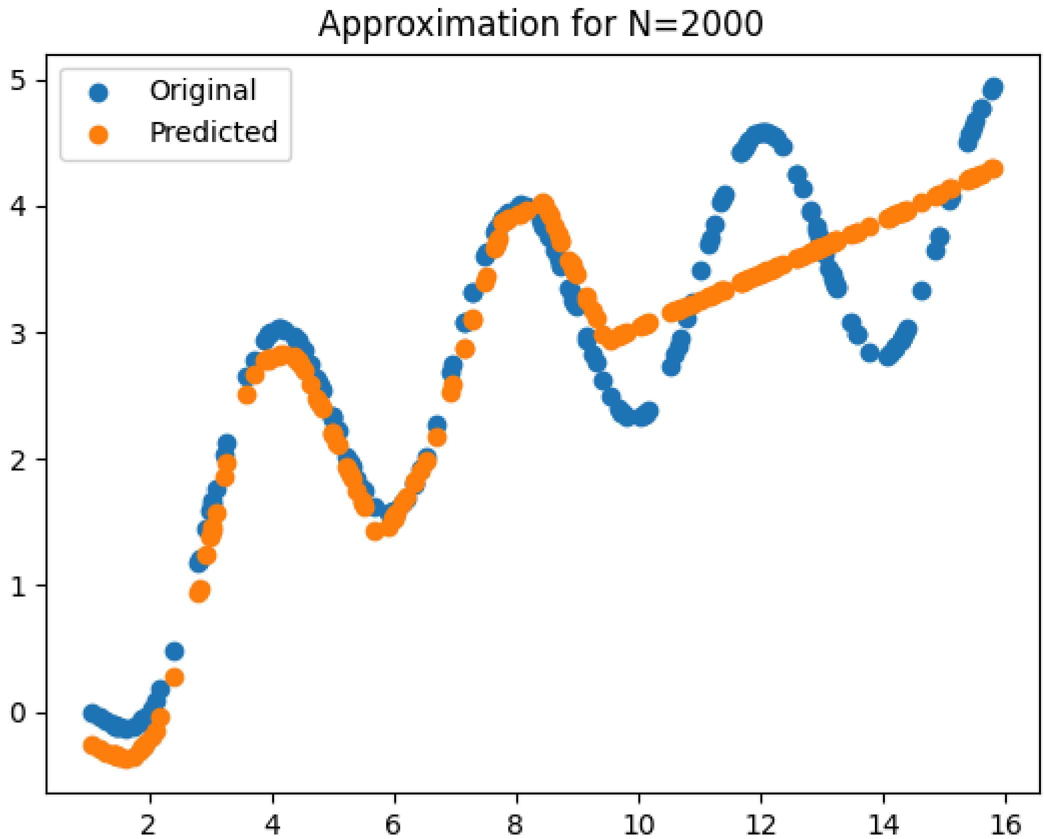

```
Validation MSE for N=200: 0.3365563154220581
Validation MSE for N=2000: 0.34552857279777527
Validation MSE for N=10000: 0.002475065877661109
```

lr = 0.005




```
Validation MSE for N=200: 0.325824111700058
Validation MSE for N=2000: 0.07084895670413971
Validation MSE for N=10000: 0.003432628232985735
```

lr = 0.01

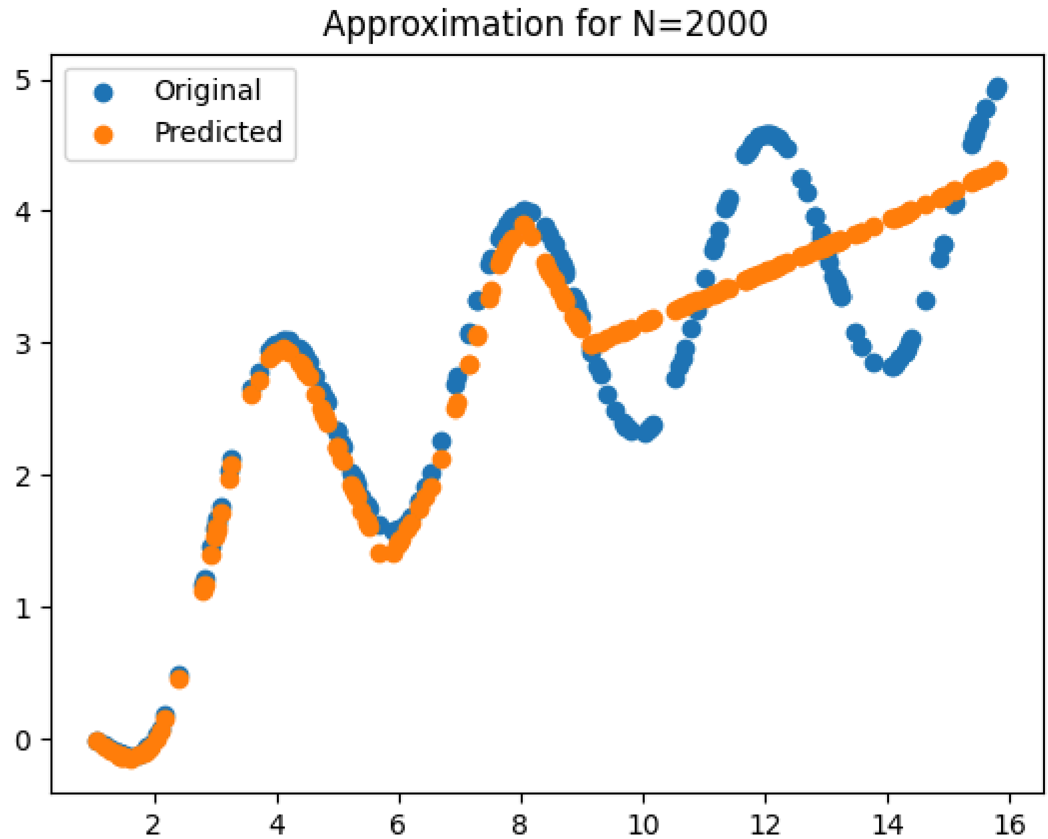



```
Validation MSE for N=200: 0.7299246191978455
Validation MSE for N=2000: 0.09395433217287064
Validation MSE for N=10000: 0.00865836814045906
```

> 对于数据量较小的数据集来说,学习率较小的效果更好.学习率设置过大，可能会导致模型在训练过程中震荡，甚至无法收.
> 对于数据量较大的数据集,学习率较大的效果偏好,学习率设置过小，模型的训练可能会陷入局部最优
> 综合来讲，lr等于0.005较优

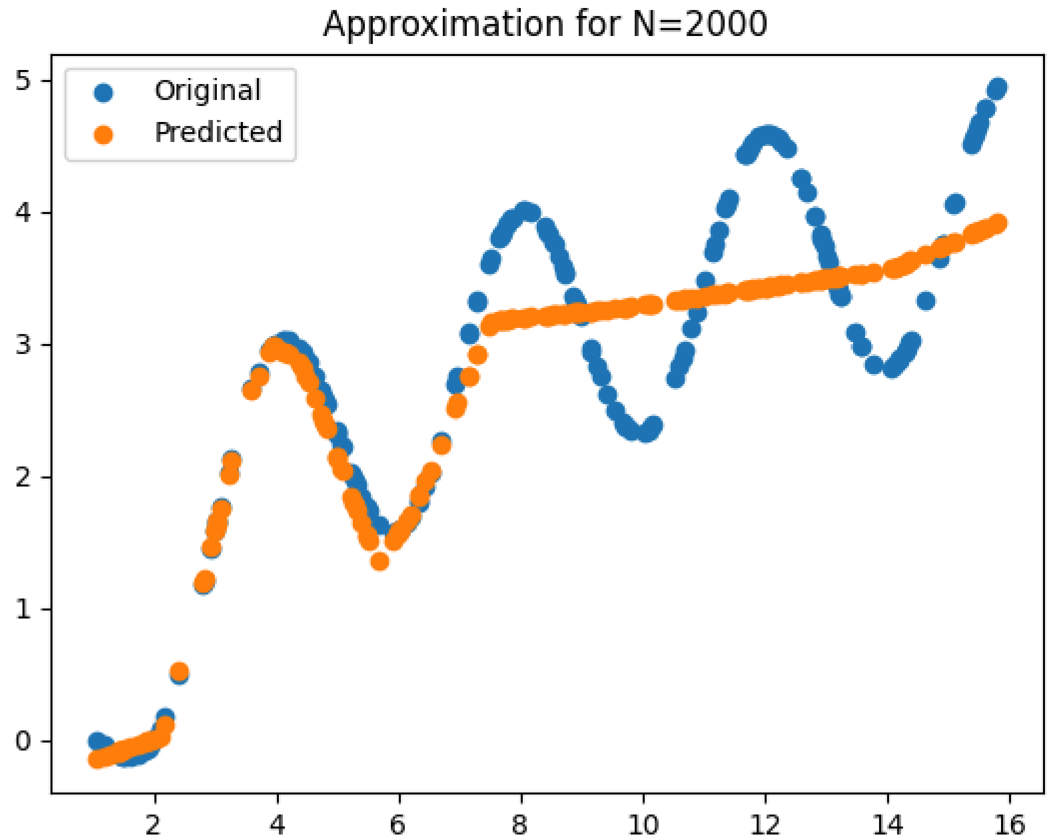
网络宽度

nw=64



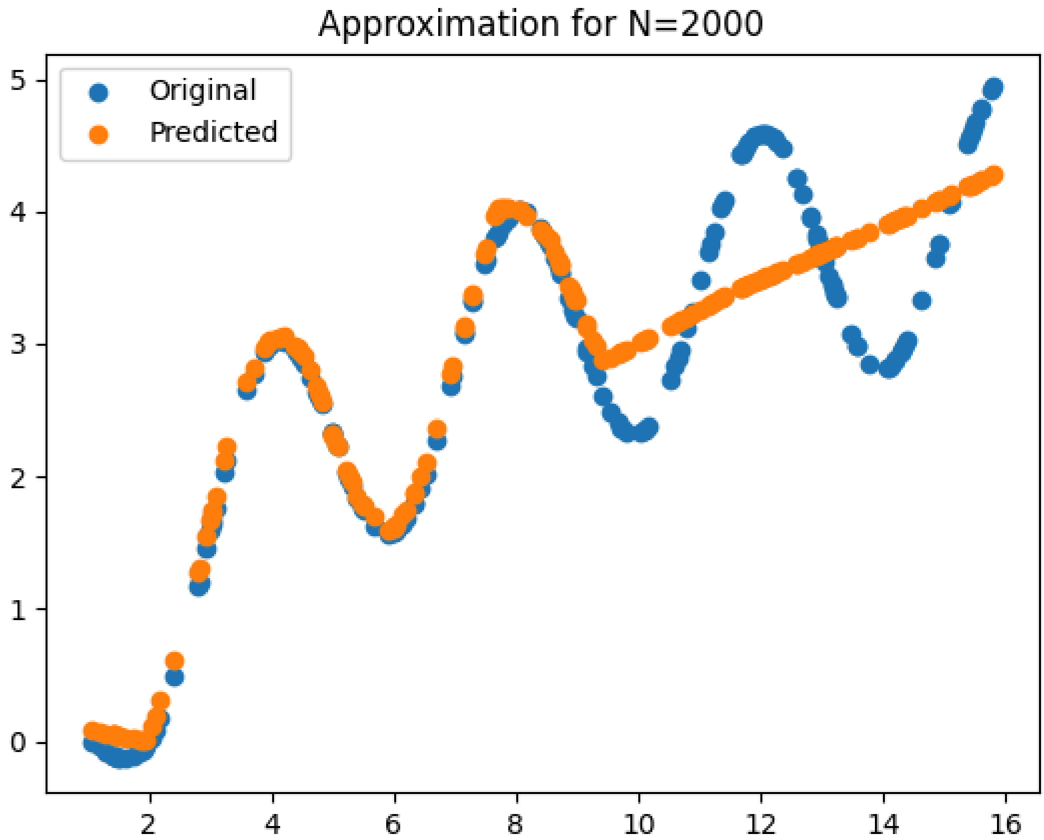

```
Validation MSE for N=200: 0.28364476561546326
Validation MSE for N=2000: 0.06791859865188599
Validation MSE for N=10000: 0.00027579313609749O8
```

nw=32




```
Validation MSE for N=200: 0.6298813223838806
Validation MSE for N=2000: 0.23928135633468628
Validation MSE for N=10000: 0.0026015974581241608
```

nw=128

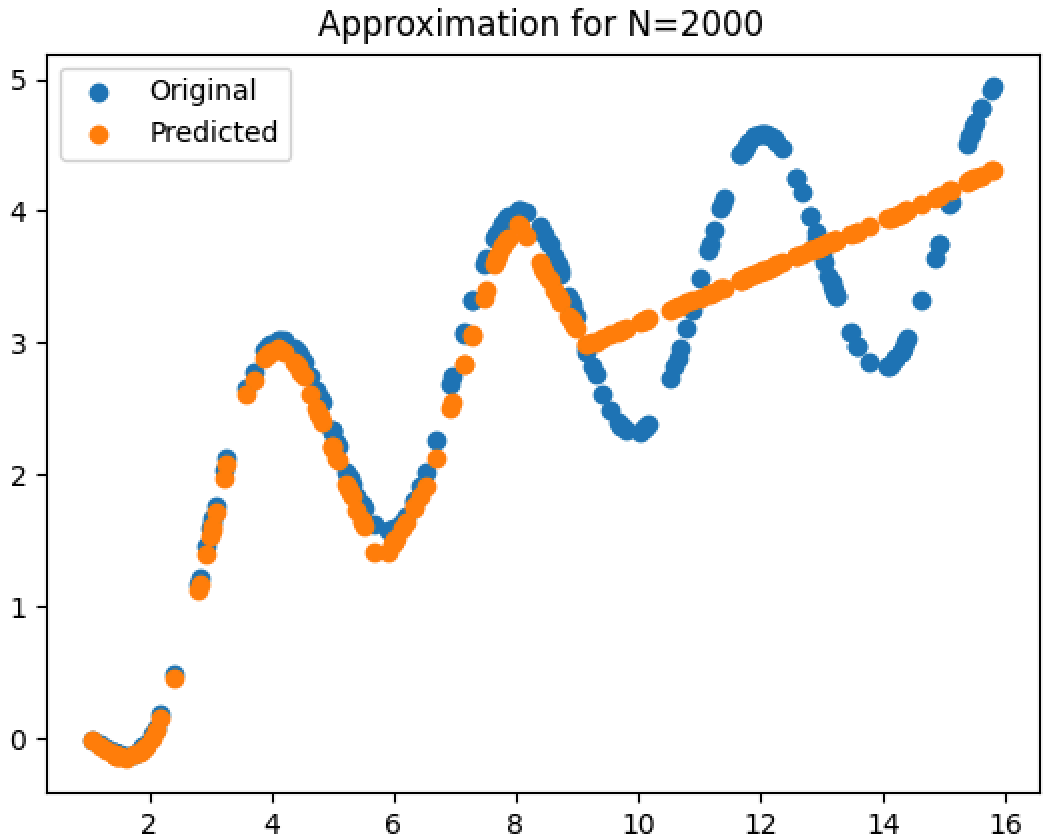



```
Validation MSE for N=200: 0.3015021085739136
Validation MSE for N=2000: 0.08028784394264221
Validation MSE for N=10000: 0.007895978167653084
```

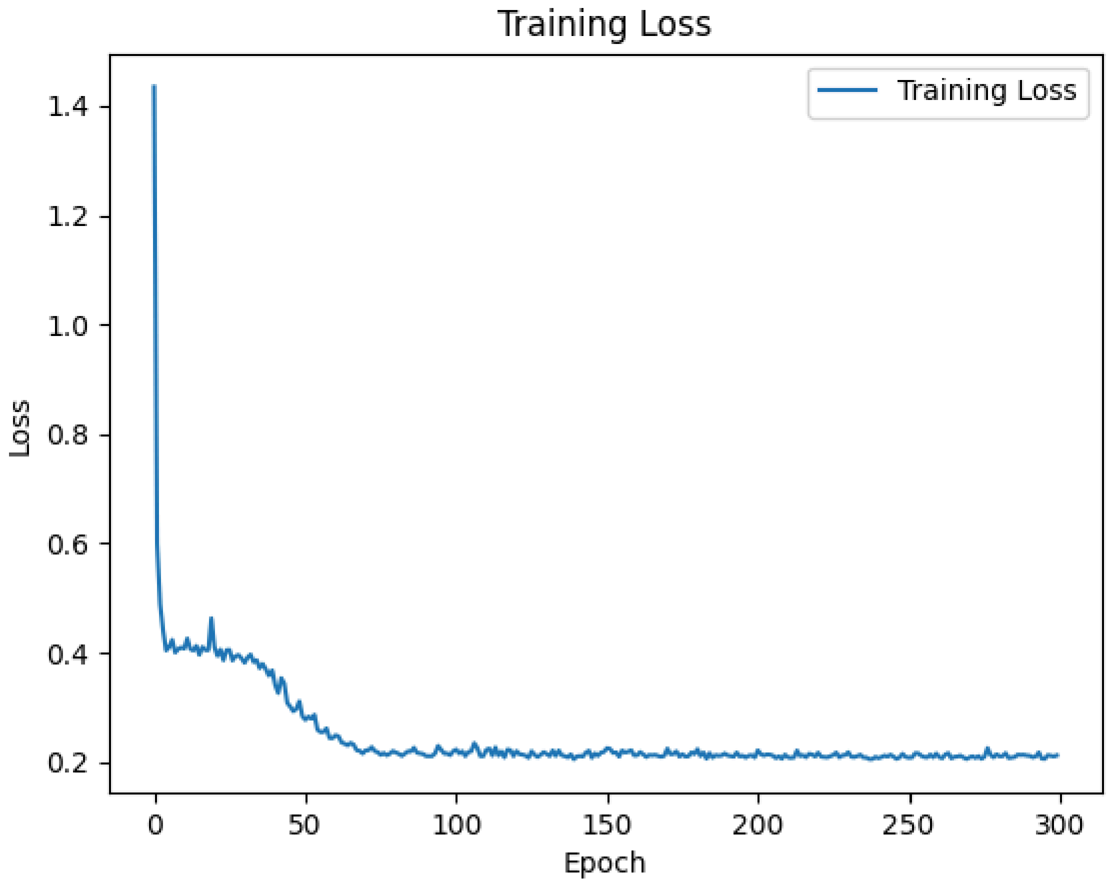
> 显然，n=64时在任何数据量下都是最优的，既防止过拟合，也具有一定的学习能力

激活函数

relu 函数

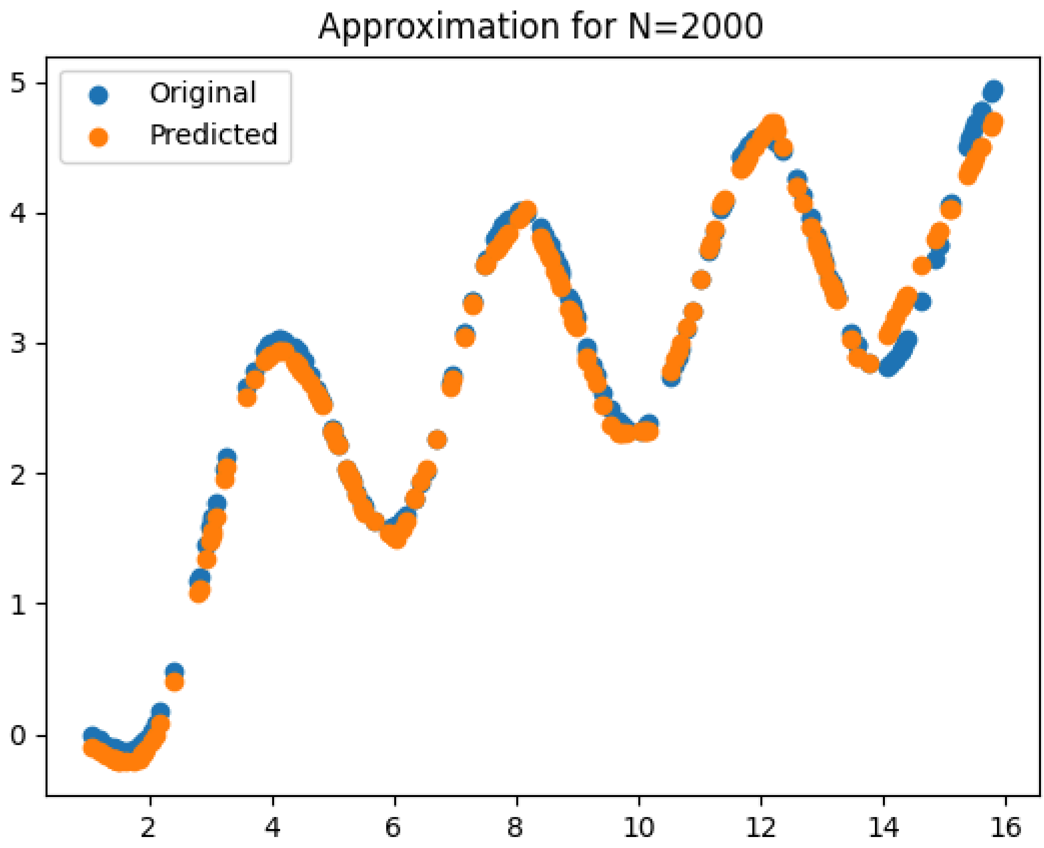



```
Validation MSE for N=200: 0.28364476561546326
Validation MSE for N=2000: 0.06791859865188599
Validation MSE for N=10000: 0.0002757931360974908
```

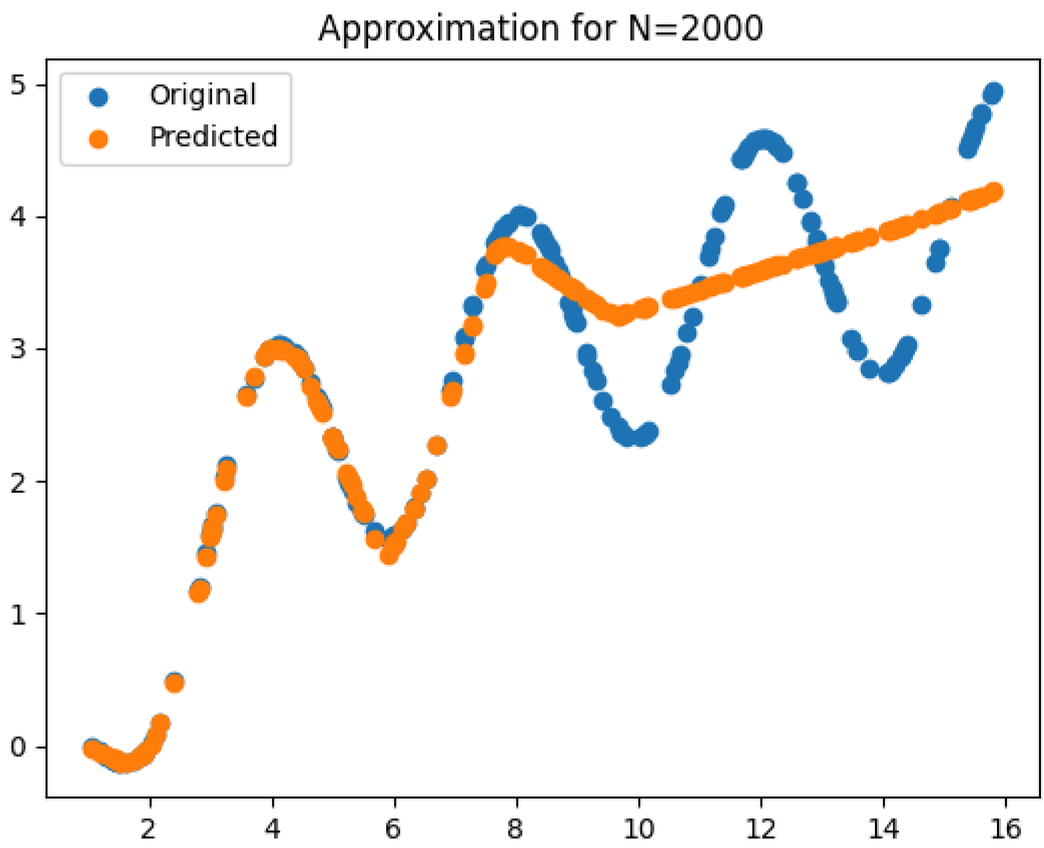


约70次开始收敛

训练轮数改为300



Leaky ReLU函数 超参数0.01

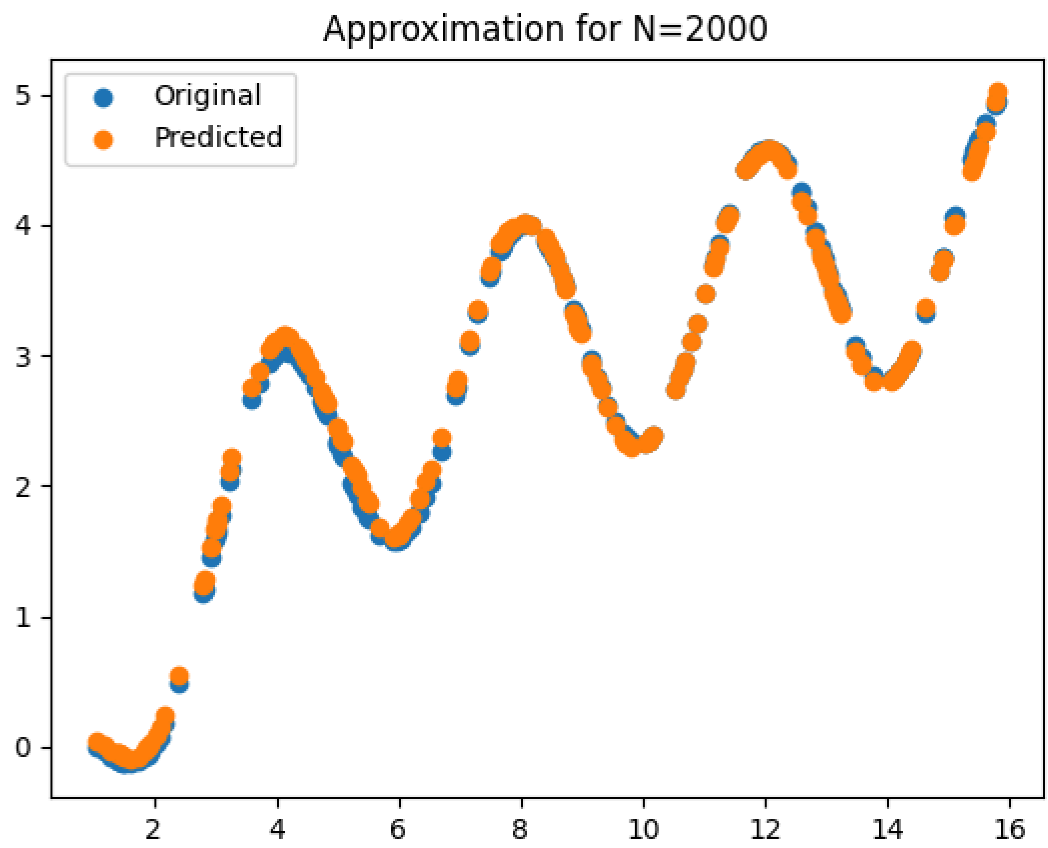



```
Validation MSE for N=200: 0.3095279335975647
Validation MSE for N=2000: 0.0728711411356926
Validation MSE for N=10000: 0.0003172760480083525
```

超参数 0.001

```
Validation MSE for N=200: 0.28443220257759094
Validation MSE for N=2000: 0.05788317322731018
Validation MSE for N=10000: 0.0009769375901669264
```

ELU 函数




```
Validation MSE for N=200: 0.32457461953163147
Validation MSE for N=2000: 0.006613452453166246
Validation MSE for N=10000: 8.574274397687986e-05
```

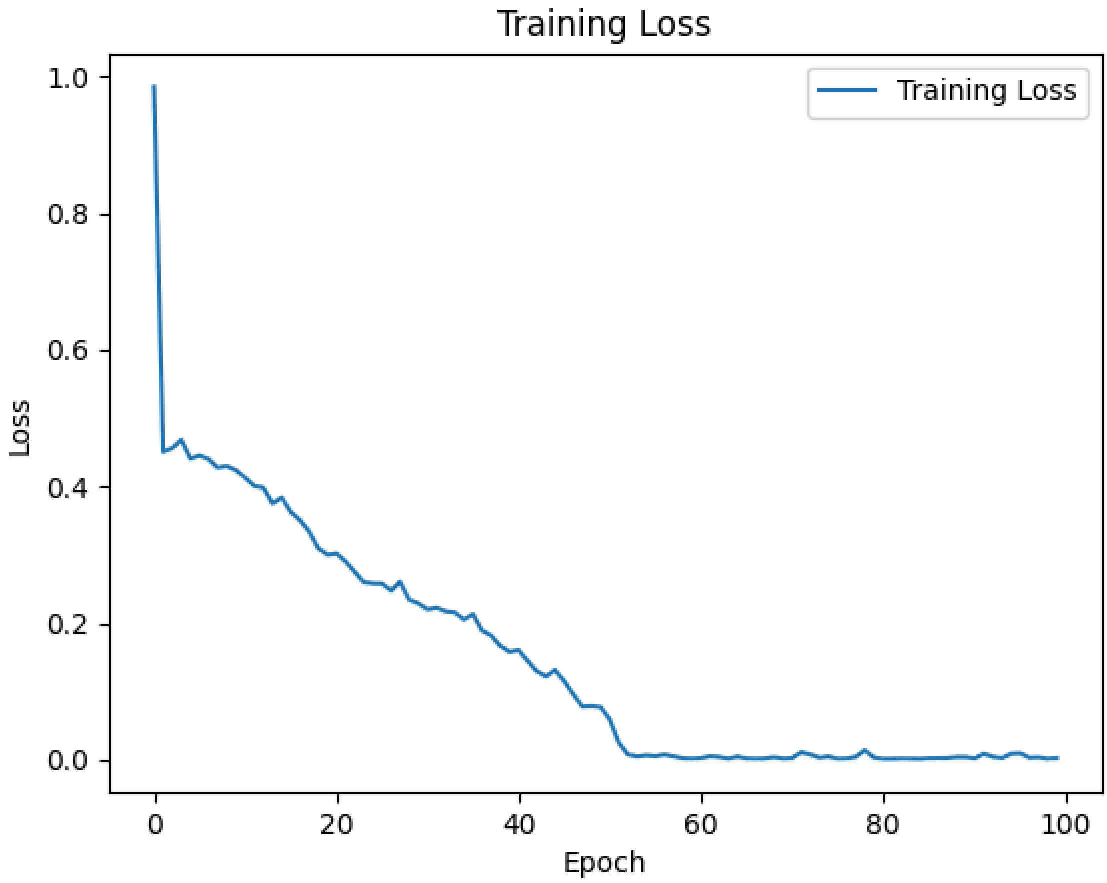
> 显然,以ELU为激活函数所得结果显著好于其他函数,这是因为ELU函数相比ReLU和Leaky ReLU，有更好的平滑性，这意味着在负区间内，ELU有一个非零的梯度，可以缓解梯度消失问题。 此外，ELU函数的另一个优点是它能够将神经元的输出近似标准化为零均值，这可以加快学习速度，因为它使得梯度下降方向更接近于最小值

训练轮数

epochs=100

```
Validation MSE for N=200: 0.32457461953163147
Validation MSE for N=2000: 0.006613452453166246
Validation MSE for N=10000: 8.574274397687986e-05
```

损失值与训练次数如下，可以看出，大约50次就可以收敛



epochs=300

```
Validation MSE for N=200: 0.15593673288822174
Validation MSE for N=2000: 0.0010014723520725965

Validation MSE for N=10000: 0.00018924933101516217
```

最终测试

```
Test MSE for N=200: 0.2819254398345947
Test MSE for N=2000: 0.0031431771349161863
Test MSE for N=10000: 0.00024385677534155548
```