**基于前馈神经网络的函数近似**

**学号：PB210000224 姓名：陈鸿绪 日期：3.29.2024**

1. **实验原理：**

**网络结构**：前馈神经网络由输入层、隐藏层和输出层组成。输入层负责接收外部数据，隐藏层对数据进行非线性变换，输出层则输出网络的最终结果。实验中我们将探究网络层数、宽度给训练拟合带来的影响。

**激活函数**：在隐藏层和输出层中，每个神经元都通过一个激活函数进行非线性变换。实验中将探究的激活函数包括ReLU、Tanh、LeakyReLU。

**损失函数**：损失函数用于衡量网络输出与实际值之间的差异。常用的损失函数包括均方误差（MSE）、l1损失、Huber损失等。训练过程中，通过最小化损失函数来优化网络的权重和偏置。我们将探究不同损失函数带来的影响。实验中将探究的误差类型为MSE、l1损失、损失。

**权重和偏置**：网络中的每个连接都有一个权重，每个神经元都有一个偏置。这些权重和偏置在训练过程中通过反向传播算法进行更新，以最小化网络输出与实际值之间误差。

**二． 实验环境：**

|  |  |  |
| --- | --- | --- |
|  | 表格1. 版本与配置 | |
| Python | | **3.11.8** |
| Pytorch(GPU) | | **2.2.1** |
| Numpy: | | **1.26.4** |
| Pandas | | **2.2.1** |
| CPU | | **Intel(R) Core(TM) i9-14900HX** |
| GPU | | **NVIDIA GeForce RTX 4070 Laptop GPU** |

**三． 实验过程：**

**数据准备**：本次实验中，我们使用随机均匀生成的数据集，对于每一个N，生成横坐标在[0, 16]中均匀分布的函数上的点坐标，并存入CSV文件中。

**构建网络以及超参数选择**：由于任务是拟合单变量函数，故输入层、输出层必须固定size为1。隐藏层深度、宽度、激活函数、学习率、batch size、epoch等均为命令行运行程序时的附带参数。

**初始化权重和偏置**：随机初始化网络中的权重和偏置。

**前向传播**：将输入数据通过网络进行前向传播，计算网络输出。

**计算损失**：使用损失函数计算网络输出与实际值之间的差异。并画出模型在验证集上的loss变化图像。

**反向传播**：根据损失函数的梯度，使用反向传播算法更新网络的权重和偏置。

**迭代训练**：重复步骤4-6，直到网络性能达到要求或达到预设的迭代次数。将模型超参数和参数全部存入checkpoint文件中。最后选出最佳超参数进行测试集评估。

**探究超参数影响**：实验中探究一系列超参数对模型性能的影响（包括batch size、损失函数类型、激活函数类型、模型宽度、模型深度、学习率）。

**四． 关键代码展示：**

**前馈神经网络代码：**

1. class MLPs(nn.Module):

2.

3. def \_\_init\_\_(self, size\_list: list, activate: str):

4. super(MLPs, self).\_\_init\_\_()

5. self.size\_list = size\_list

6. self.activate = activate

7. self.MLPs\_layer = self.\_\_make\_layer()

8.

9. def \_\_make\_layer(self):

10. layers = []

11. for i in range(len(self.size\_list))[:-1]:

12. layers.append(nn.Linear(self.size\_list[i], self.size\_list[i + 1]))

13. if i != len(self.size\_list)-2:

14. if self.activate == "ReLU":

15. layers.append(nn.ReLU())

16. elif self.activate == "LeakyReLU":

17. layers.append(nn.LeakyReLU())

18. elif self.activate == "Tanh":

19. layers.append(nn.Tanh())

20. return nn.Sequential(\*layers)

21.

22. def forward(self, x):

23. x = self.MLPs\_layer(x)

24. return x

25.

**模型训练代码：**

1. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

2. print(device)

3. model = MLPs(size\_list, activate)

4.

5. if path and load\_model:

6. model.load\_state\_dict(checkpoint['model\_state\_dict'])

7. model.to(device)

8. optimizer = opt.Adam(model.parameters(), lr=lr)

9. #scheduler = StepLR(optimizer, step\_size=1, gamma=0.99)

10.

11. criterion = nn.MSELoss() # 定义损失函数MSE

12. criterion = criterion.to(device)

13.

14. for epoch in range(num\_epochs):

15. progress\_bar(epoch, num\_epochs, prefix='process: ', length=50)

16. model.train() # 设置模型为训练模式

17. for i,(inputs,true\_y) in enumerate(train\_iter):

18. inputs = inputs.to(device)

19. true\_y = true\_y.to(device) # 将输入和标签移至设备

20. optimizer.zero\_grad() # 清零梯度

21. outputs = model(inputs) # 前向传播

22. loss = criterion(outputs, true\_y) # 计算损失

23. loss.backward() # 反向传播:

24. optimizer.step() # 更新模型参数

25. #scheduler.step()

26. model.eval()

**五． 超参数分析：**

为了减少篇幅，**这里只展示N为10000的结果**，且默认隐藏层每一层的size都保持一致。

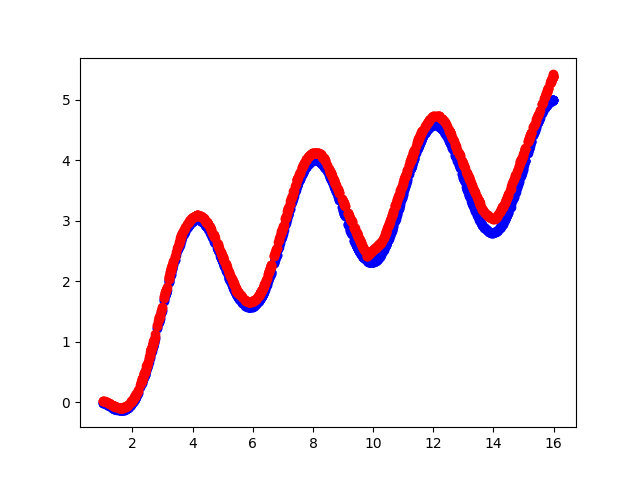
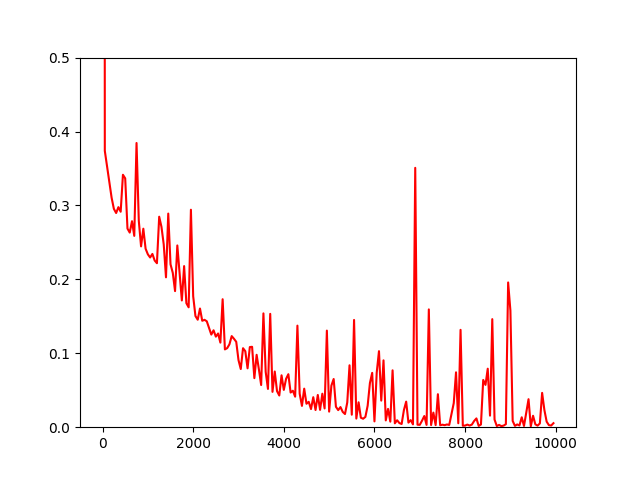
1. **网络深度：**

我们在这一小节中固定以下参数的值：

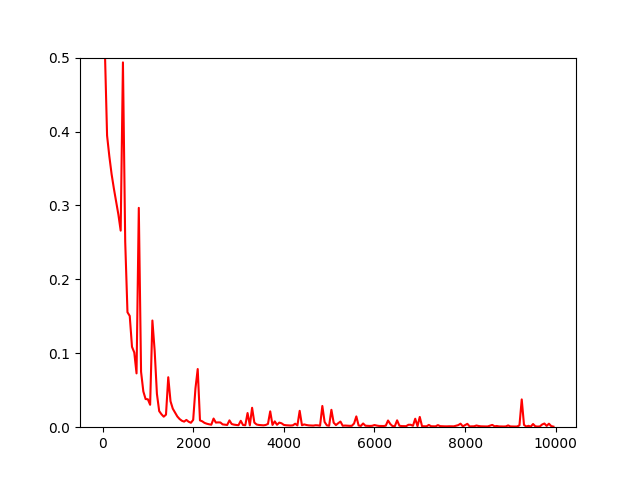
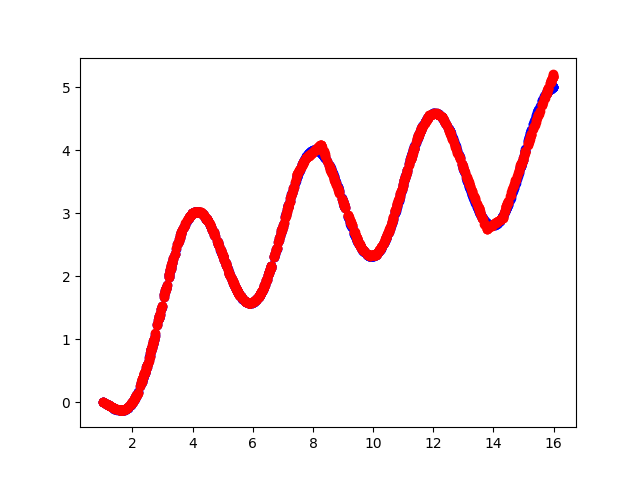
num\_epochs: 10000

batch size: 2048

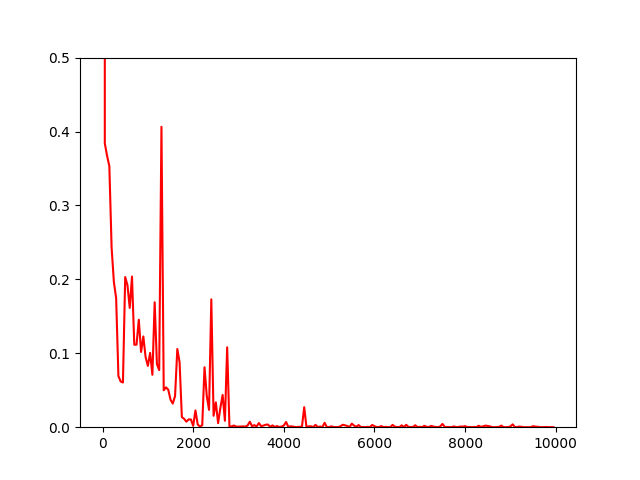
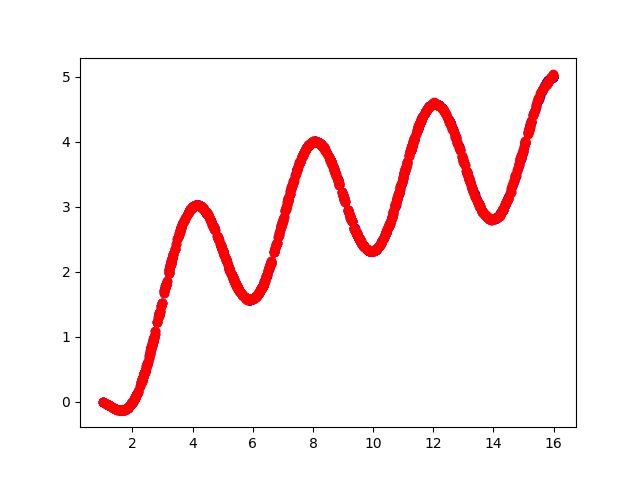
learning rate: 0.003

Activation Function: ReLU

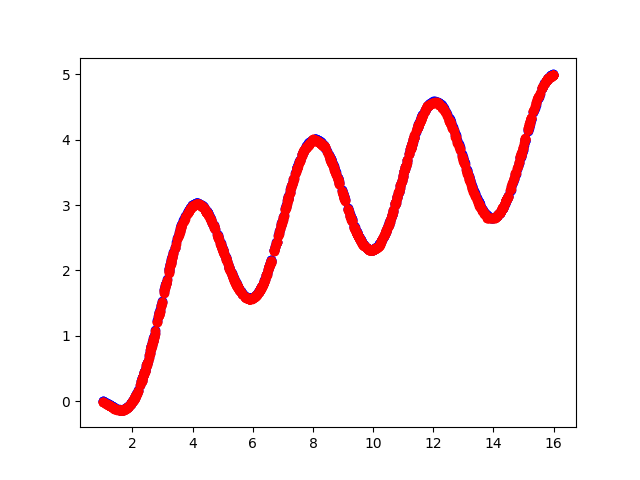
**图1. model size list: [1, 1024, 1]，running\_loss: 0.02403175**

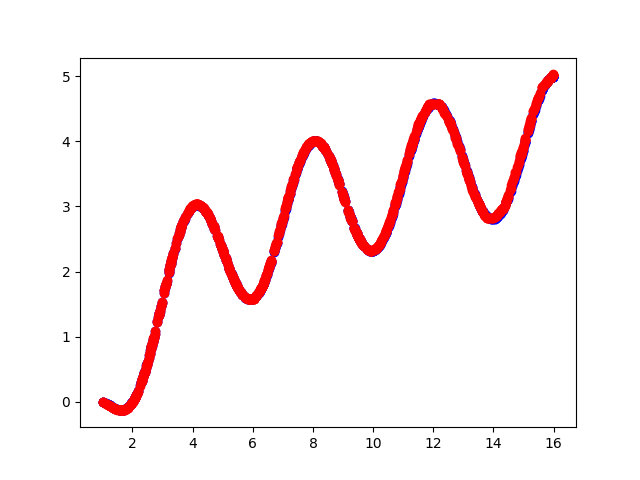
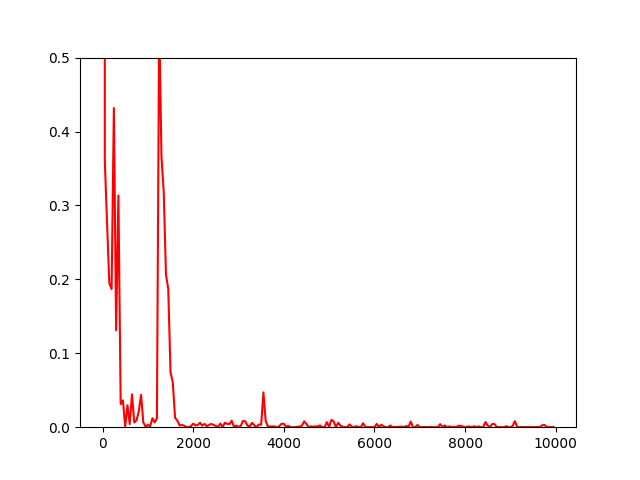


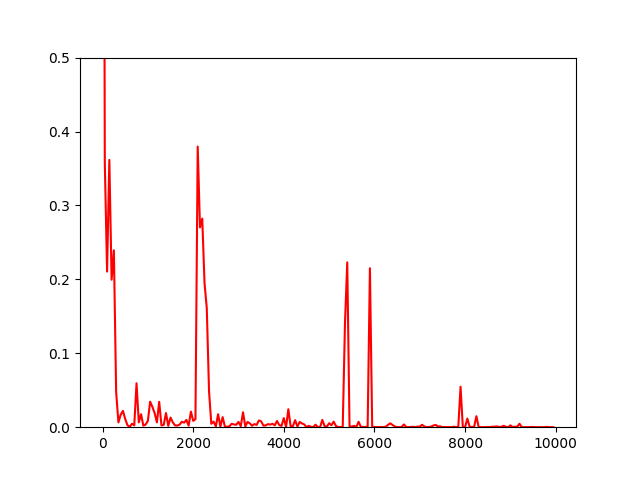
**图2. model size list: [1, 1024, 1024, 1], running\_loss: 0.00098391**



**图3. model size list: [1, 1024×4, 1], running\_loss: 4.467e-05**



**图4. model size list: [1, 1024×6, 1], running\_loss: 0.00021222**

**图5. model size list: [1, 1024×8, 1]. running\_loss: 0.00029199**

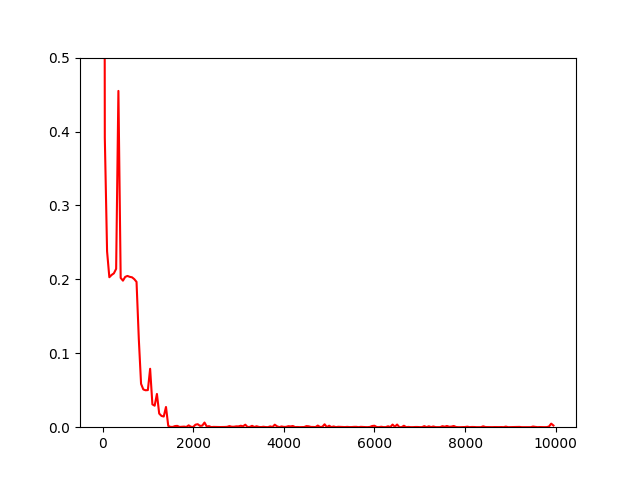
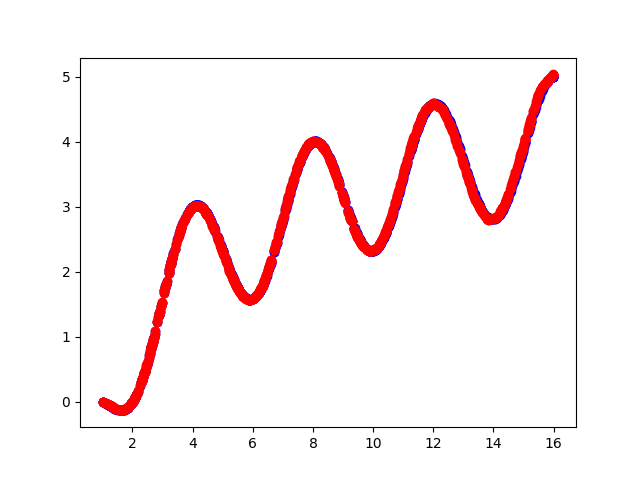
1. **网络宽度：**

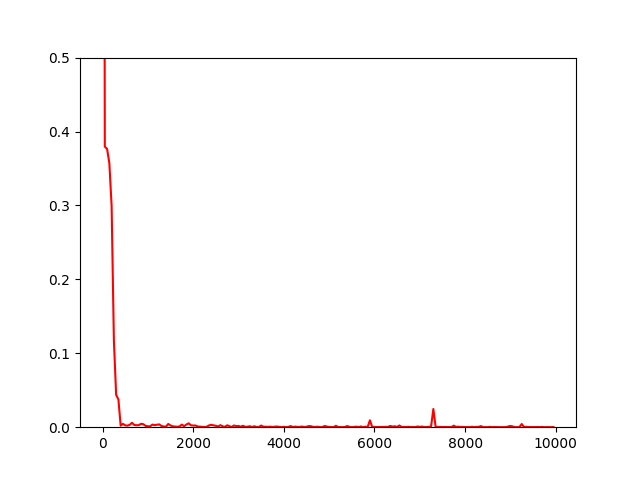
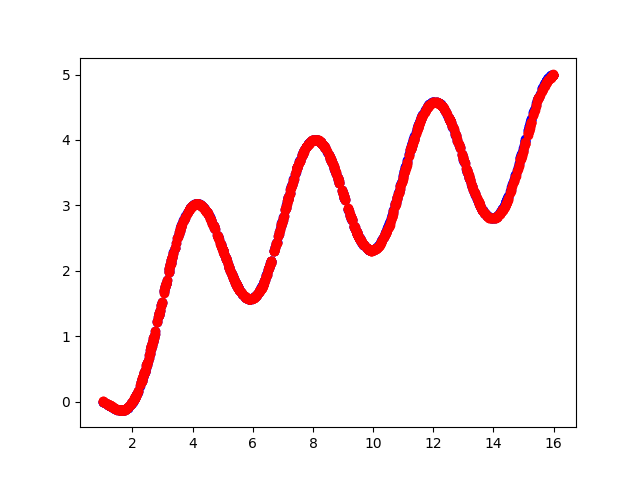
我们在这一小节中固定以下参数的值：

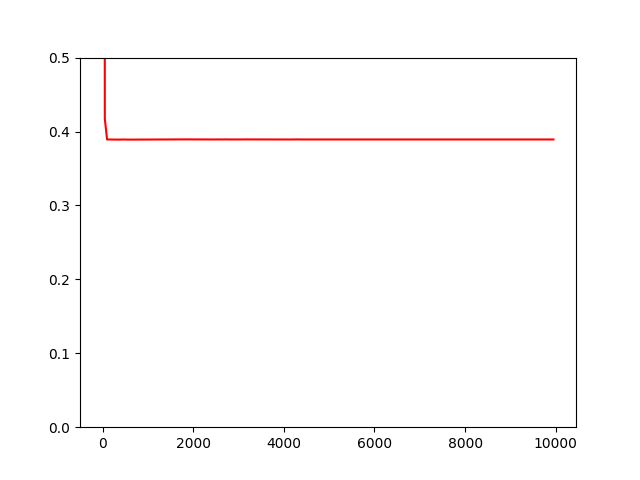
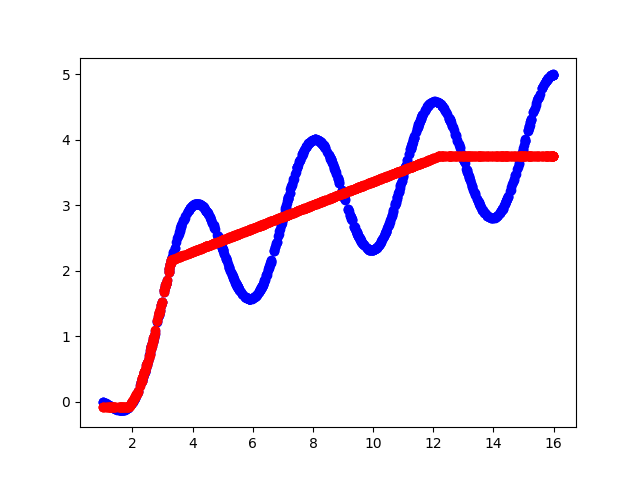
num\_epochs: 10000

batch size: 2048

learning rate: 0.003

Activation Function: ReLU

**图6. model size list: [1, 512×4, 1], running\_loss: 0.00066505**

******图7. model size list: [1, 64×4, 1], running\_loss: 0.00018237**

**图8. model size list: [1, 5×4, 1], running\_loss: 0.41658613**

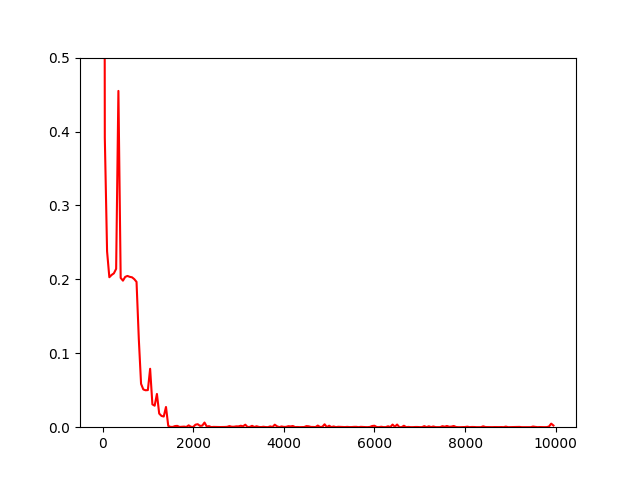
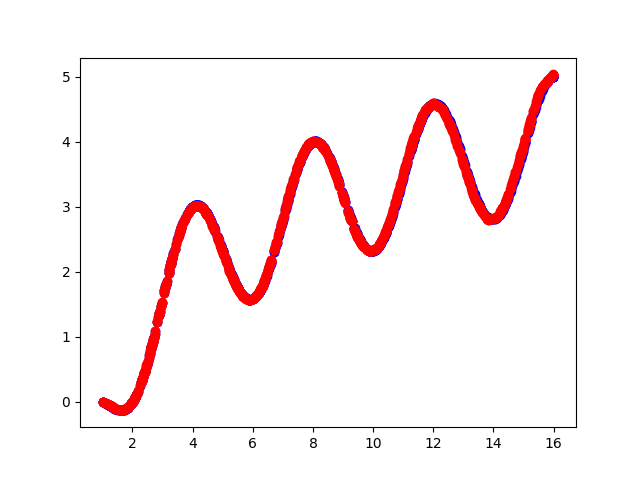
1. **激活函数：**

我们在这一小节中固定以下参数的值：

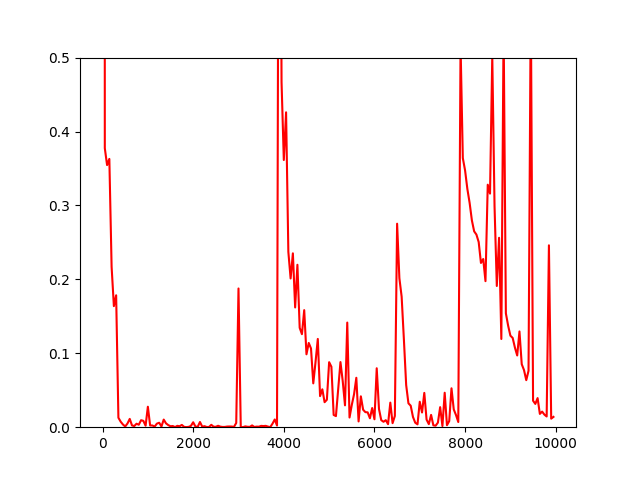
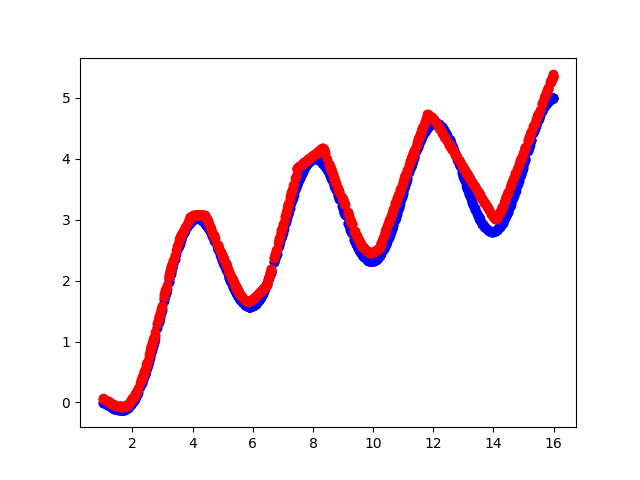
num\_epochs: 10000

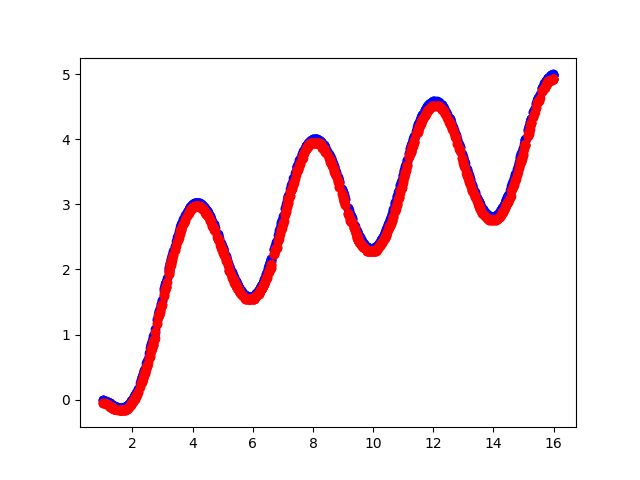
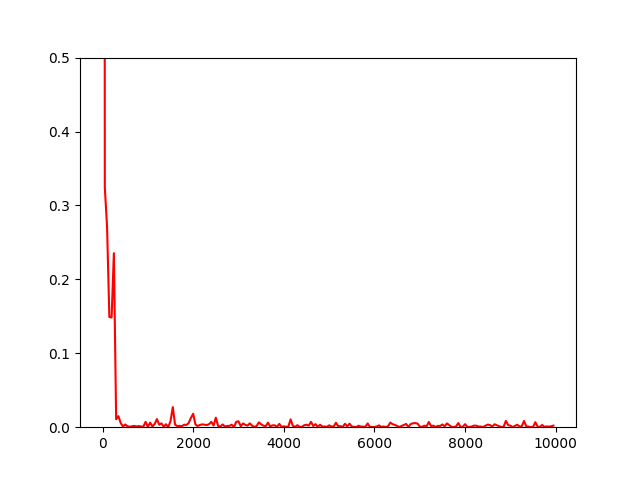
batch size: 2048

learning rate: 0.003

model size list: [1, 512, 512, 512, 1]

**图9. 激活函数: ReLU, running\_loss: 0.000665055**



**图10. 激活函数: LeakyReLU, running\_loss: 0.02475779**

**图11. 激活函数:Tanh, running\_loss: 0.00388890**

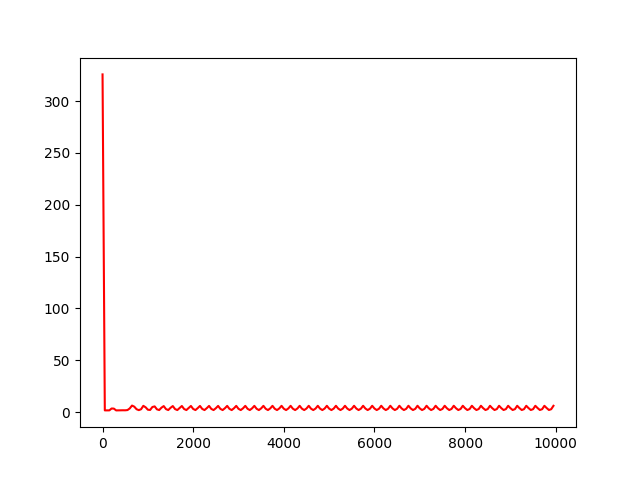
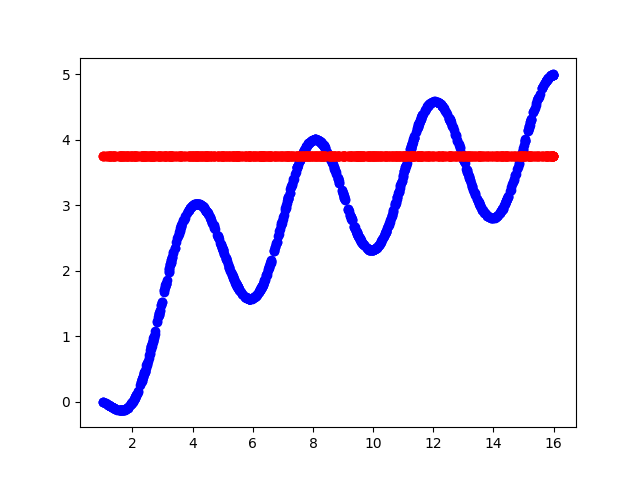
1. **学习率：**

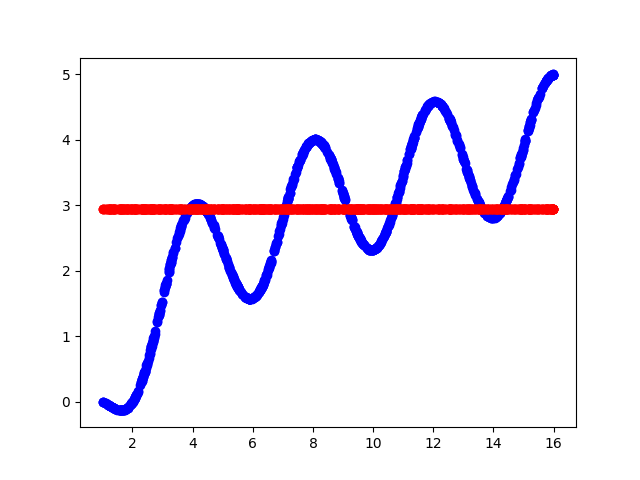
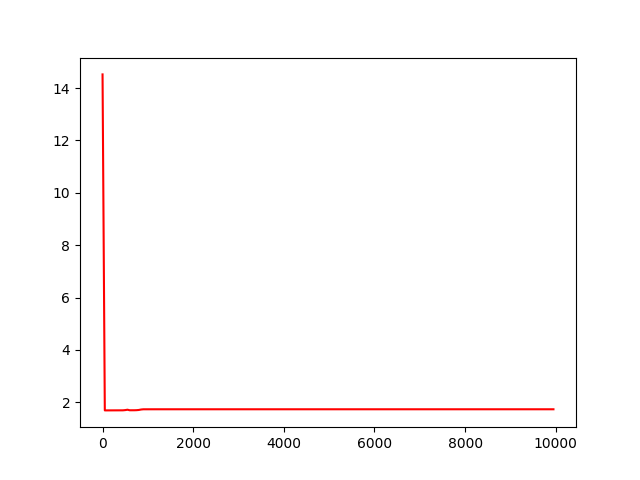
我们在这一小节中固定以下参数的值：

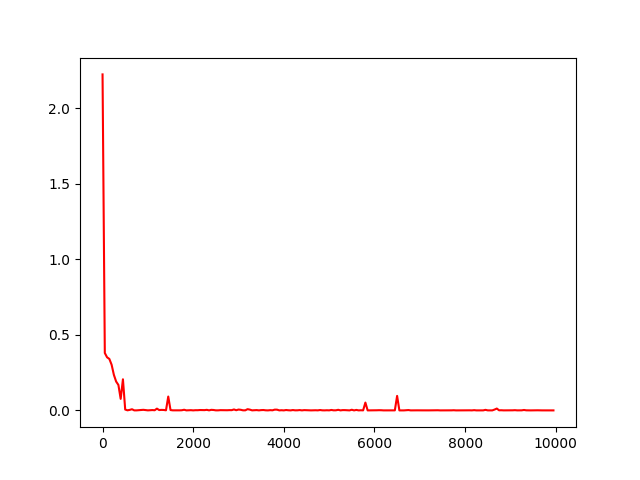
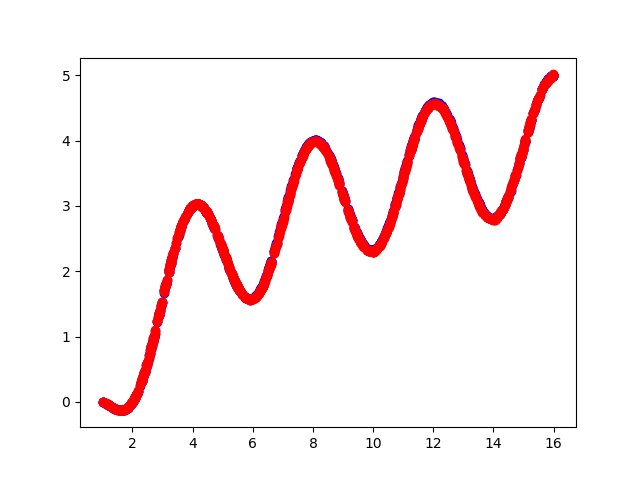
num\_epochs: 10000

batch size: 2048

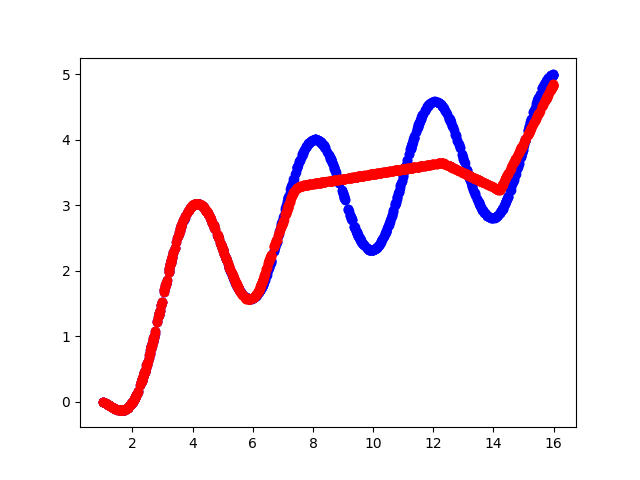
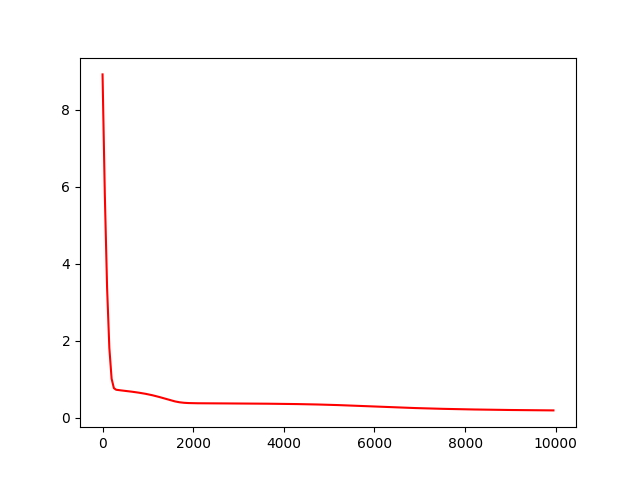
model size list: [1, 512, 512, 512, 1]

ctivation Function: ReLU

**图12. 学习率: 0.1, running\_loss: 2.69729543**

**图13. 学习率: 0.01, running\_loss: 1.71281552**

**图14. 学习率: 0.001, running\_loss: 0.00023515**



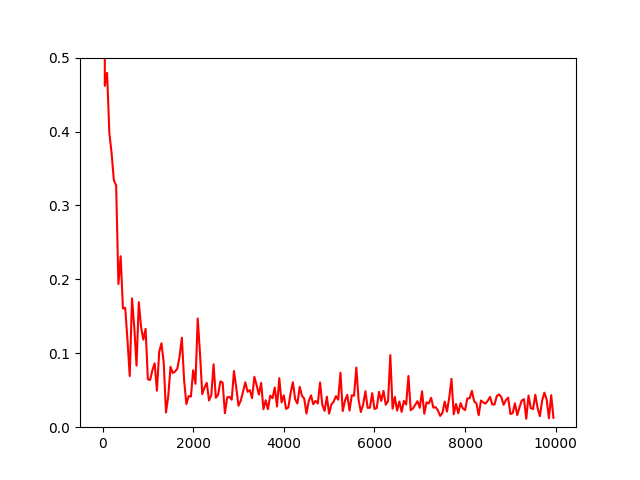
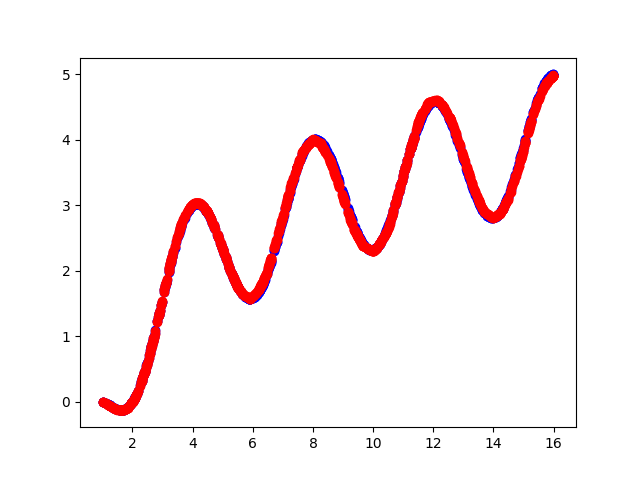
**图15. 学习率: 1e-6, running\_loss: 0.20517459**

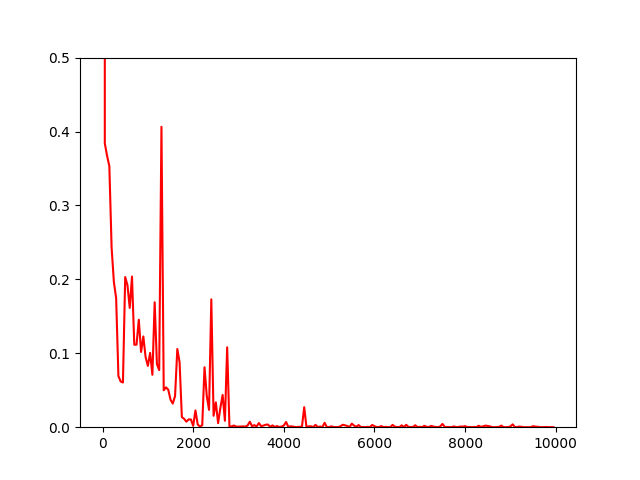
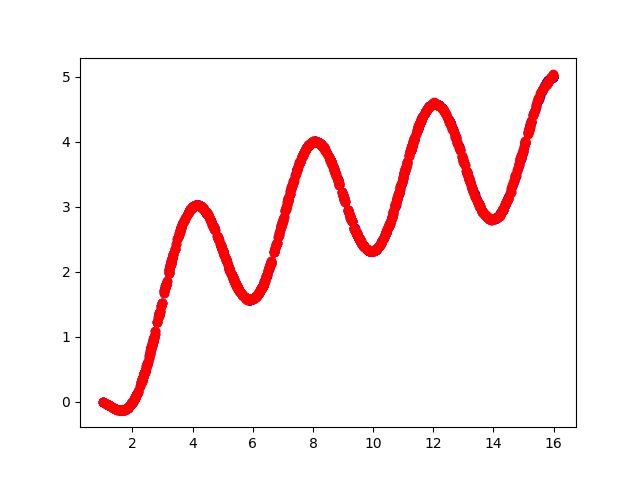
1. **损失函数类型：**

num\_epochs: 10000

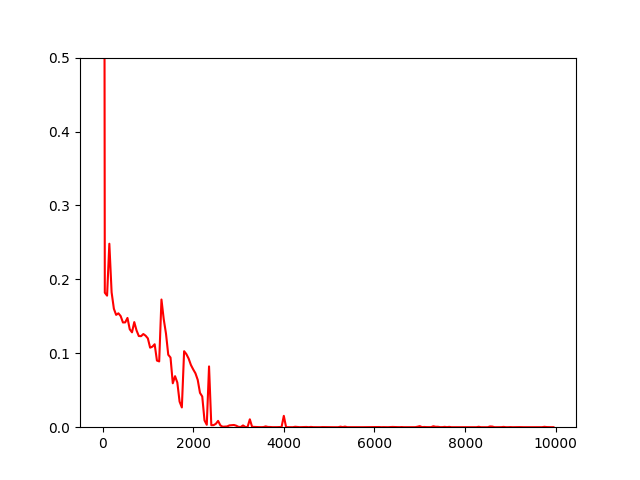
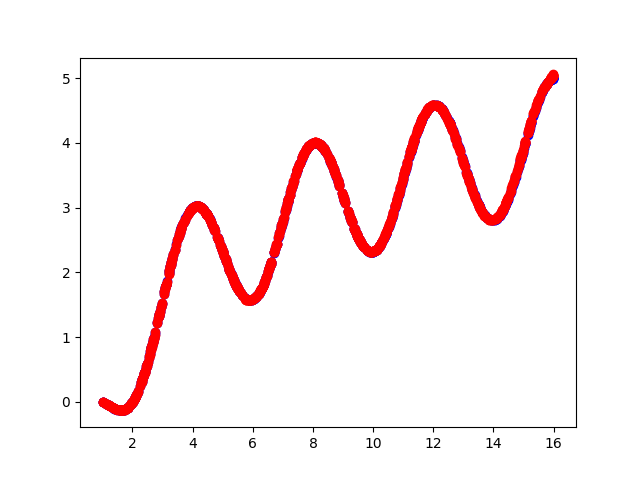
batch size: 2048

model size list: [1, 1024, 1024, 1024, 1]

ctivation Function: ReLU

 **图16. 损失函数：L1损失, running\_loss: 0.02023888**

**图17. 损失函数：MSE损失, running\_loss: 4.467e-05**



**图18. 损失函数：Huber损失, running\_loss: 4.898e-5**

1. **Batch Size：**

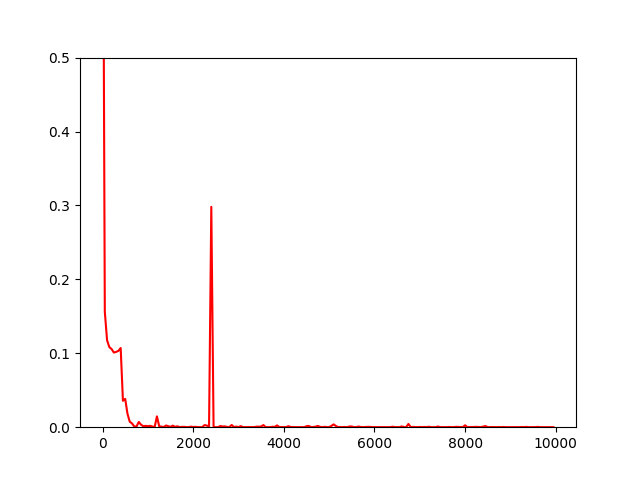
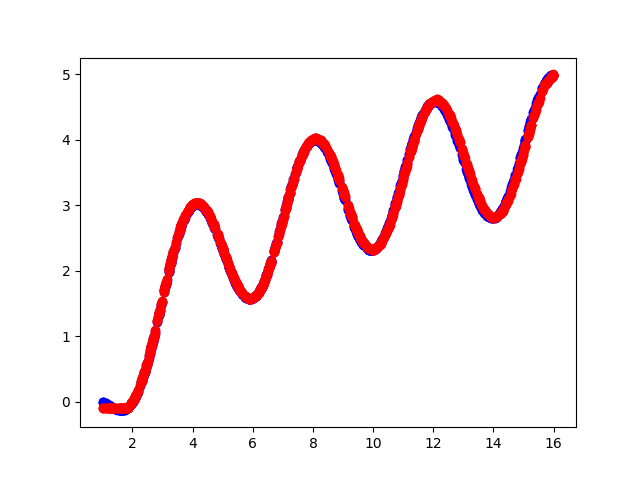
我们在这一小节中固定以下参数的值：

num\_epochs: 10000

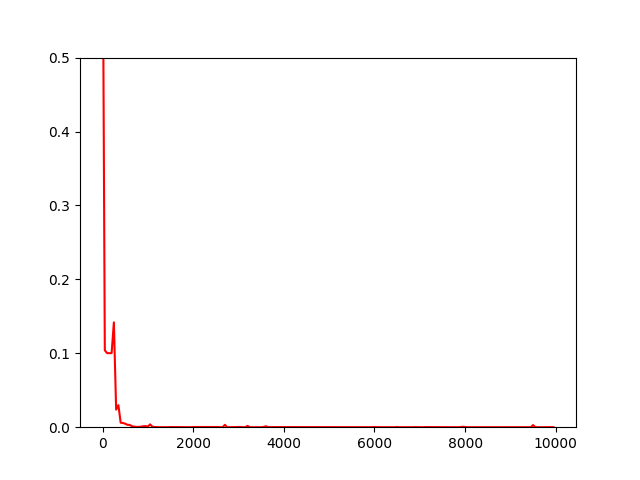
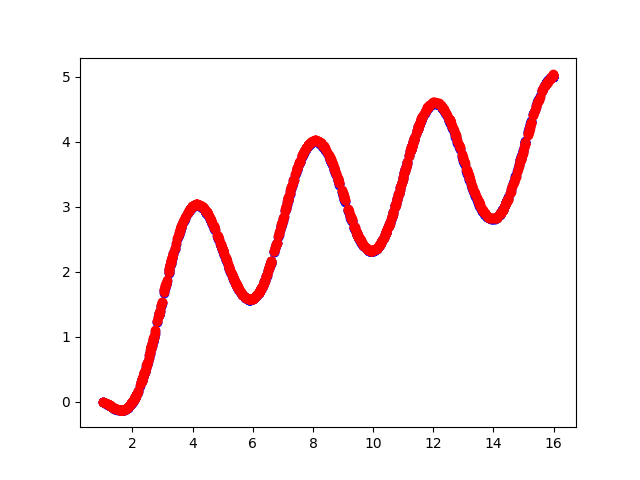
learning rate: 0.003

model size list: [1, 512, 512, 512, 1]

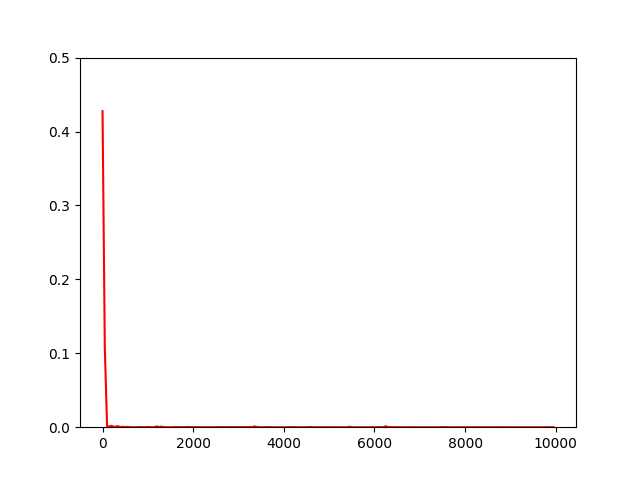
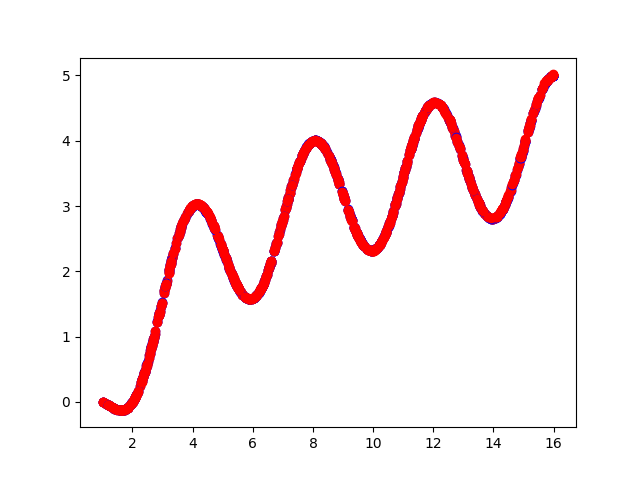
Ctivation Function: ReLU



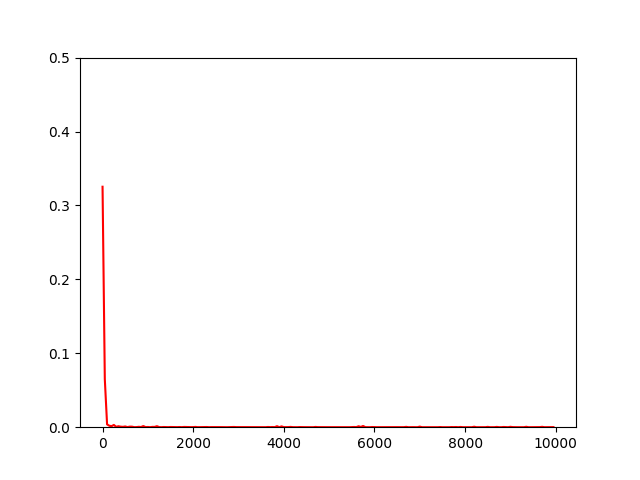
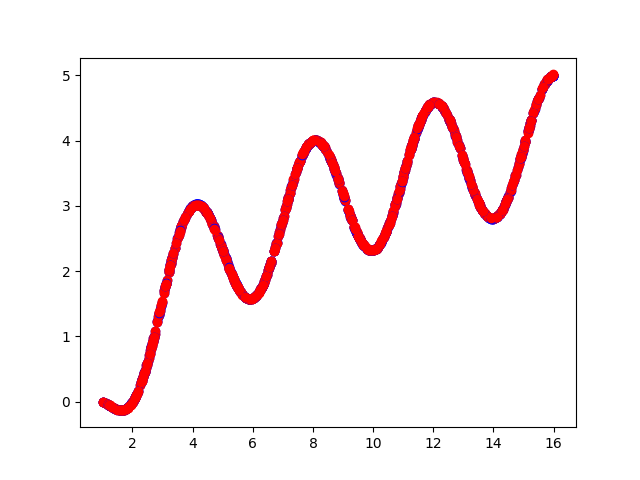
**图19. batch size：2048, running\_loss: 0.00068986**

**图20. batch size：1024, running\_loss: 8.9595e-05**

****

**图21. batch size：512, running\_loss:** **1.5380e-05**

****

**图22. batch size：256, running\_loss:** **9.8899e-06**

1. **分析：**

下面我们将通过分析以上列出的一系列性能图，阐述超参数对于模型性能的影响。

**网络深度：**在图1到图5中展示了固定网络宽度为1024的条件下，网络深度对于模型收敛效果的影响。在网络只有一层的时候，在验证集上的loss量级大致在1e-2级别，两层为1e-4级别，四层达到最优，量级为1e-5。然而4层后网络层数的增加并没有带来验证集上损失量级的减少。所以我们可以观察到，并不是网络层数越深越好，因为网络层数越深，理论上我们需要更多数据或者更多的epoch让模型得到更好的训练拟合。需要通过实践、并结合具体情况选择适合的层数。

**网络宽度：**保持模型深度为4，改变模型的宽度为512、64、5，我们可以发现训练之后的模型在验证集上的 loss在模型宽度为5的时候最大。结合验证集上的拟合图像图8，可以发现模型处于欠拟合状态。这其实是由于在宽度为5的时候，模型的参数量过少，导致模型可以拟合的函数复杂度不高，理论上我们可以在模型宽度较小的时候，通过增加模型深度来提高泛化性能（实验中保存的N=200和N=2000 checkpoint文件可以表现这点）.

**激活函数：**在实验中尝试了三种不同的激活函数，分别为ReLU, Tanh, LeakyReLU。图9到图11中展示了其余超参数相同，但是在不同激活函数情形下模型在验证集上的性能。其中ReLU激活函数效果最佳、Tanh其次、最后是LeakyReLU。同时实验过程中也可以注意到ReLU的计算效率也是最好的。

**学习率：**图12到图15记录了学习率逐渐从1e-1降低到1e-6过程中，模型在验证集上的表现。1e-1到1e-2我们可以注意到，验证集上的 loss一直得不到有效降低，模型也得不到很好的拟合（甚至一直为一条直线），这是因为在学习率比较大的时候，模型一方面无法去有效逐步逼近最佳参数，另一方面网络可能会出现“Dead ReLU”现象，即某一层的输出会几乎全部为负值，经过ReLU会被截断成0，一旦该层出现这样的情况会使得模型的向后传播算法无法有效执行，模型无法更新迭代。所以我们观察到在较小的学习率1e-3开始时，模型在验证集上得到良好拟合。如果学习率过小，如图12所示为1e-6的学习率，模型每次循环无法得到充分更新，所以在给定最大epoch下，模型处于欠拟合的时候结束了训练。

**损失函数：**图16到图18记录了只在不同损失函数类型下，模型在验证集上的表现。我们使用了L1、MSE以及Huber损失函数，但是根据验证集上的性能显示MSE的表现效果最佳，Huber损失函数效果略低于MSE，L1损失在验证集得到的loss量级比前者大了3个数量级，这是因为原函数是一个完全光滑的曲线，然而这三个损失函数中，只有L1损失拟合出的曲线没有良好的光滑性质，所以最后验证集上表现的效果L1最差。

**Batch Size：**图19到图21记录了不同batch size大小下（2024、1024、512和256），模型在验证集上的表现。实验过程中可以明显感知到batch size越大，训练模型速度越快（前提是在显存没有占满的情况）。然而还可以发现在batch size越小的时候，模型在验证集上的loss越小，在batch size为256的时候loss甚至达到了1e-6的量级，这是因为较小的batch size通常有助于模型更好地泛化到新数据，因为它们在训练过程中接触到了更多的数据组合。这种随机性有助于减少过拟合，并可能使模型在未见过的数据上表现更好。

**五．最佳参数选择和测试集表现：**

最后在经过一系列比较后我们导入如下最佳参数为：

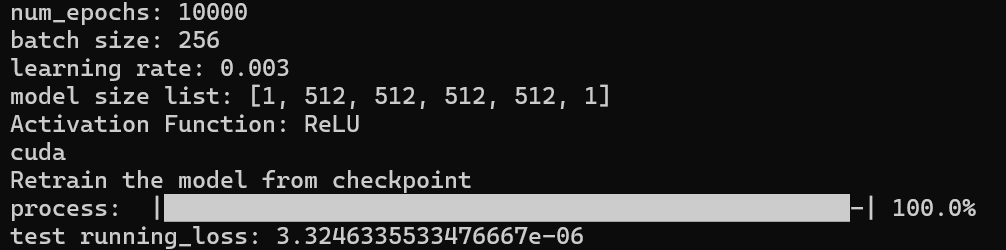
num\_epochs: 10000

batch size: 256

learning rate: 0.003

model size list: [1, 512, 512, 512, 512, 1]

Activation Function: ReLU

最后在测试集上的loss为：3.3246335533476667e-06。这也就表明对于该超参数，前馈神经网络已经很好拟合了该函数。