Using the method of Assignment 2 to approximate $f'(x)$.

First, by the method of calculus, $f'(x) = \frac{-50x}{(1+25x^2)^2}$, for all $x \in [-1, 1]$.

Point pick (uniformly), let $y_n = f'(-1+\frac{2n}{n}) = \frac{-50(-1+\frac{2n}{n})}{[1+25(-1+\frac{2n}{n})^2]^2}$ for all $n = 0, 1, 2, \cdots, 100$.

Construct tanh neural network with 3 layers. (一個隱藏層 with 20個神經元)

Loss: $L = \frac{1}{101}\sum_{n=0}^{100}(\hat{y}_n - y_n)^2$.

Optimizer: SGD. (let r = learning rate)

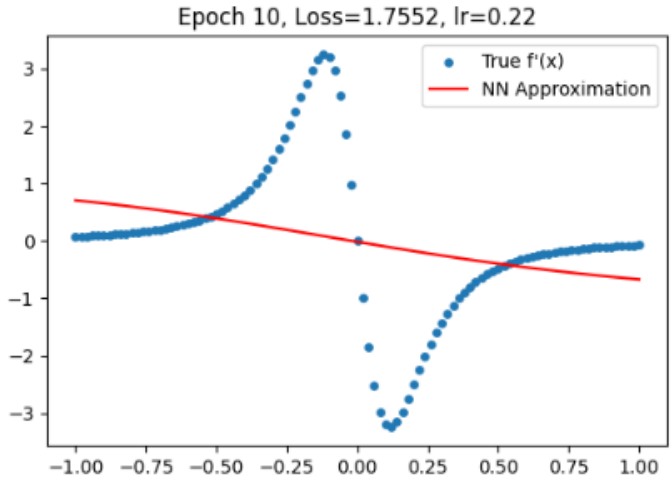
* Training process.

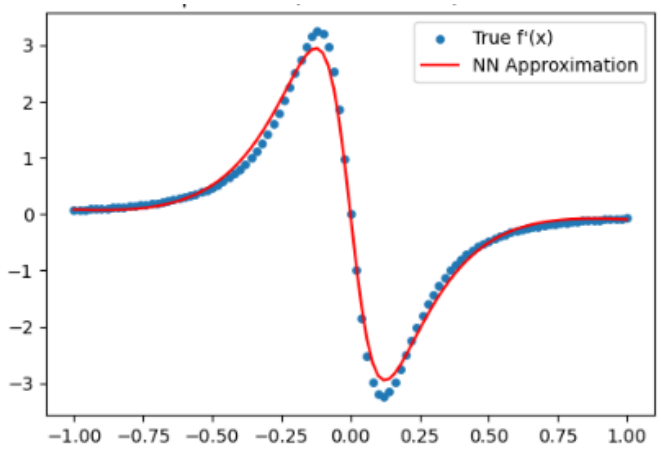
根據上面的 setting，最初先使用 $r = 0.1$ or $r = 0.3$ 做嘗試，發現 $r = 0.1$ 時，速度過慢，執行百次皆沒有太大的進展，而 $r = 0.3$ 則是過大，無法逼近。

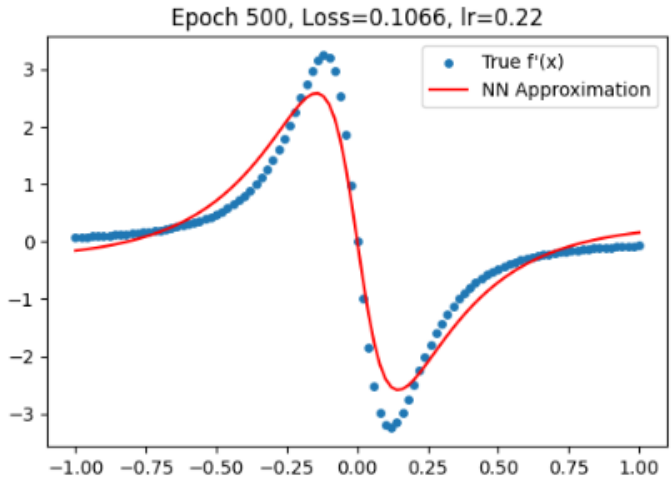
最後，選定 $r = 0.22$ 執行 500 次，$r = 0.1$ 執行 500 次，$r = 0.05$ 執行 500 次，$r = 0.01$ 執行 500 次。

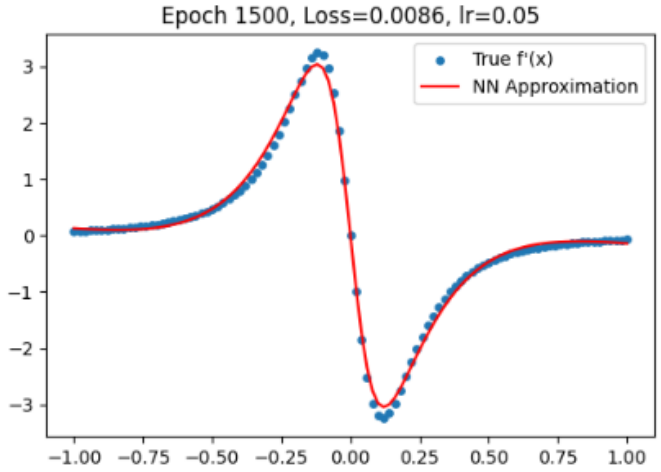
以下為 code (provide by discussing with ChatGPT)，和圖 (getting by google colab)

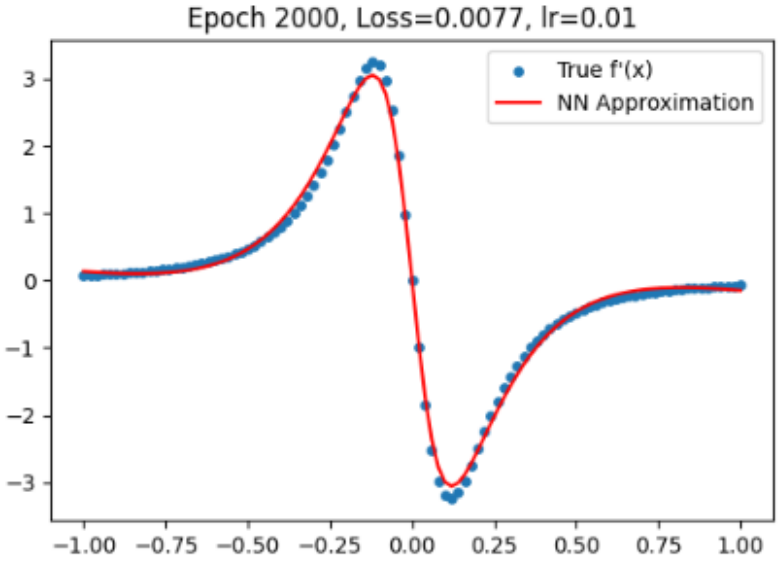
```python
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

# 目標函數的一階導數
def runge_derivative(x):
    return -50 * x / (1 + 25 * x**2) ** 2

# 生成訓練資料
x_train = torch.linspace(-1, 1, 101).unsqueeze(1)
y_train = runge_derivative(x_train)

# 定義神經網路
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.hidden = nn.Linear(1, 20)
        self.output = nn.Linear(20, 1)
        self.act = torch.tanh

    def forward(self, x):
        return self.output(self.act(self.hidden(x)))

net = Net()
criterion = nn.MSELoss()
optimizer = optim.SGD(net.parameters(), lr=0.22)  # 初始 lr

# 訓練參數
epochs = 2000
lr_schedule = [
    (0, 500, 0.22),
    (500, 1000, 0.10),
    (1000, 1500, 0.05),
    (1500, 2000, 0.01)
]

loss_history = []

for epoch in range(epochs):
    # 根據當前 epoch 調整學習率
    for start, end, lr in lr_schedule:
        if start <= epoch < end:
            for g in optimizer.param_groups:
                g['lr'] = lr

    # 前向傳播
    outputs = net(x_train)
    loss = criterion(outputs, y_train)

    # 反向傳播
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    loss_history.append(loss.item())

    # 繪圖條件：第 10 次 or 每 500 次
    if (epoch+1) == 10 or (epoch+1) % 500 == 0:
        plt.figure(figsize=(6,4))
        plt.scatter(x_train.detach().numpy(), y_train.detach().numpy(), label="True f'(x)", s=15)
        plt.plot(x_train.detach().numpy(), outputs.detach().numpy(), 'r-', label="NN Approximation
        # 找出當前 lr
        current_lr = optimizer.param_groups[0]['lr']
        plt.title(f"Epoch {epoch+1}, Loss={loss.item():.4f}, lr={current_lr}")
        plt.legend()
        plt.show()

# 最後輸出 Loss 曲線
plt.figure(figsize=(6,4))
plt.plot(loss_history)
plt.title("Loss Curve")
plt.xlabel("Epoch")
plt.ylabel("MSE Loss")
plt.show()
```











第二題

要求同時逼近 $f(x)$, $f'(x)$.

我選擇延用了大部分原本的設定. (tanh neural network)

單隱藏層 with 20 神經元.

$L_f = \frac{1}{101}\sum_{n=0}^{100}(\hat{y}_f - y_f)^2$ $\quad L_{f'} = \frac{1}{101}\sum_{n=0}^{100}(\hat{y}_{f'} - y_{f'})^2$

唯一較不同的是，I set a loss function with two component f(x), f'(x) be defined as $L_{f,f'} = 0.5 L_f + 0.5 L_{f'}$

是 L_f、$L_{f'}$ 的加權平均。

Optimizer : SGD (r = learning rate).

Training process.

有了前面兩次用經驗，我初始便將 r 設定在 0.2 執行 100 ~ 300 次.

最後發現在 120 次的時候有最好的有效進展。

並且後面慢慢的嘗試，the final setting : r = 0.2 , 120 times.

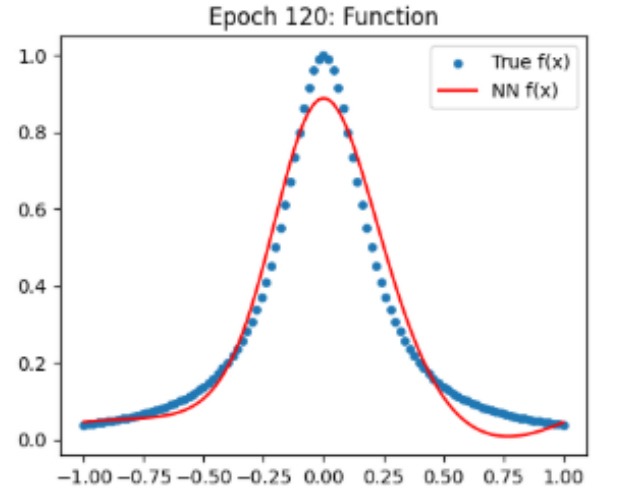
r = 0.02 , 120 times.

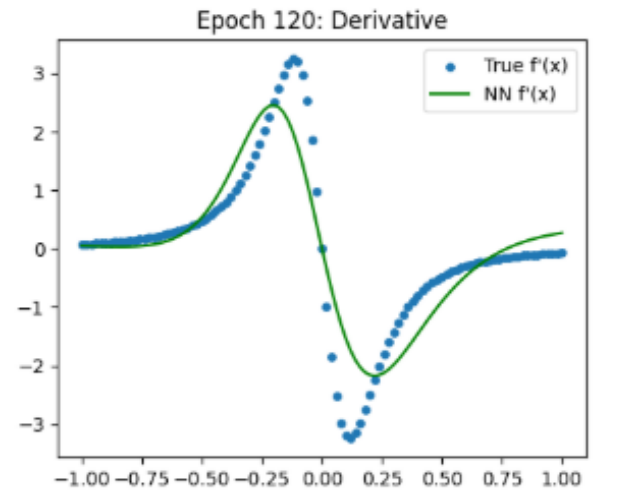
r = 0.005 , 240 times.

Processing pictural (因為設定相同，使用的是幾乎一樣的 code，就不再貼一次了)

r = 0.2 ↓

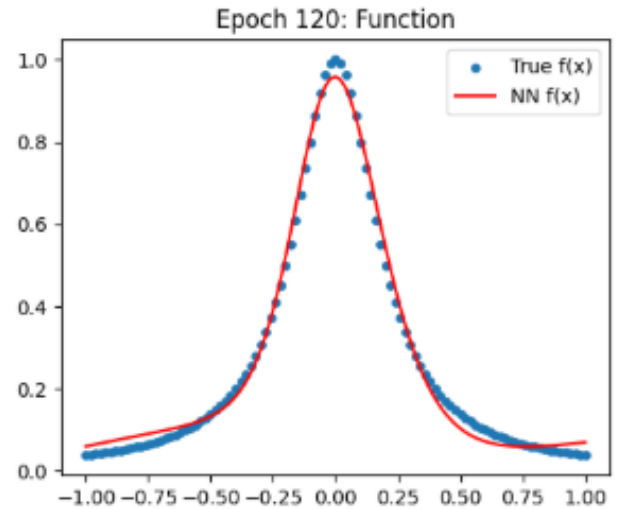
Epoch 120: Loss_f=0.0027, Loss_df=0.3382, Loss_total=0.1705

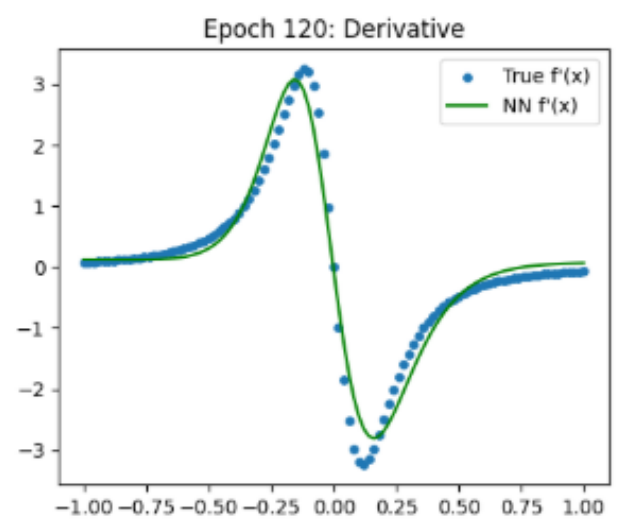




r = 0.02 ↓

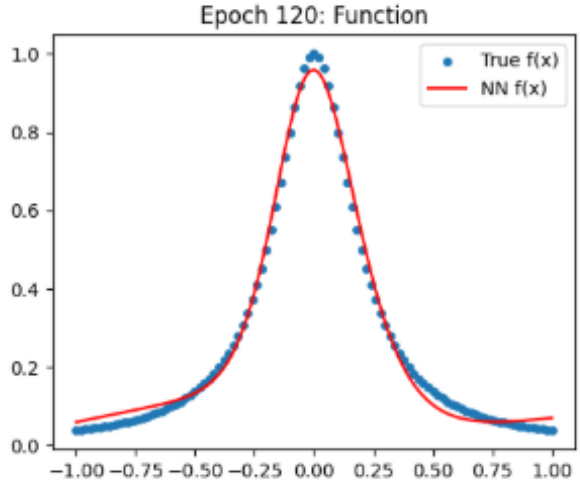
Epoch 120: Loss_f=0.0006, Loss_df=0.0799, Loss_total=0.0402

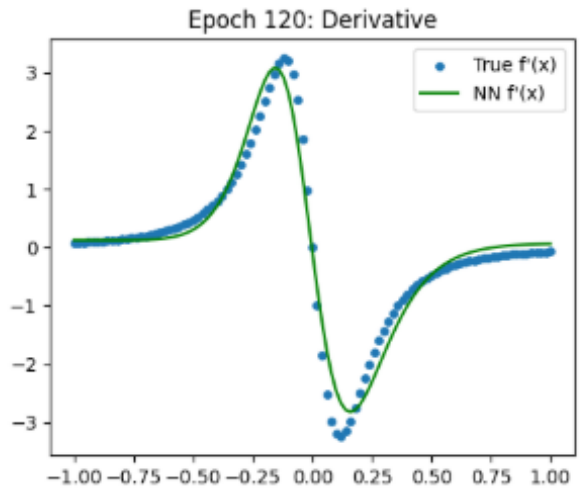




r = 0.005

Epoch 120: Loss_f=0.0005, Loss_df=0.0772, Loss_total=0.0389





Epoch 240: Loss_f=0.0005, Loss_df=0.0747, Loss_total=0.0376