Programming assignment

2025 Fall - 機器學習

111652001　　吳文生

Use a neural network to approximate the Runge function

$$f(x) = \frac{1}{1+25x^2},\ x \in [-1,\ 1]$$

Write a short report (1–2 pages) explaining method, results, and discussion including

- Plot the true function and the neural network prediction together.
- Show the training/validation loss curves.
- Compute and report errors (MSE or max error).

Our network has two hidden layers of 32 neurons each, with tanh activation and a linear output layer.

The training set consists of 1200 uniformly sampled input points with corresponding function values, and the validation set consists of 300 points. The training loop runs for 100 epochs, recording training and validation loss each epoch.

Results

1. The neural network prediction closely matches the true Runge function. The two curves almost completely overlap with no obvious deviation.

2. The training and validation losses both quickly decrease and stabilize, indicating good learning and generalization. The loss curves show a stable and effective training process.

3. The validation set MSE is about 0.000001, and max error about 0.00259, demonstrating high approximation precision and effectively capturing the numerical behavior of the Runge function.

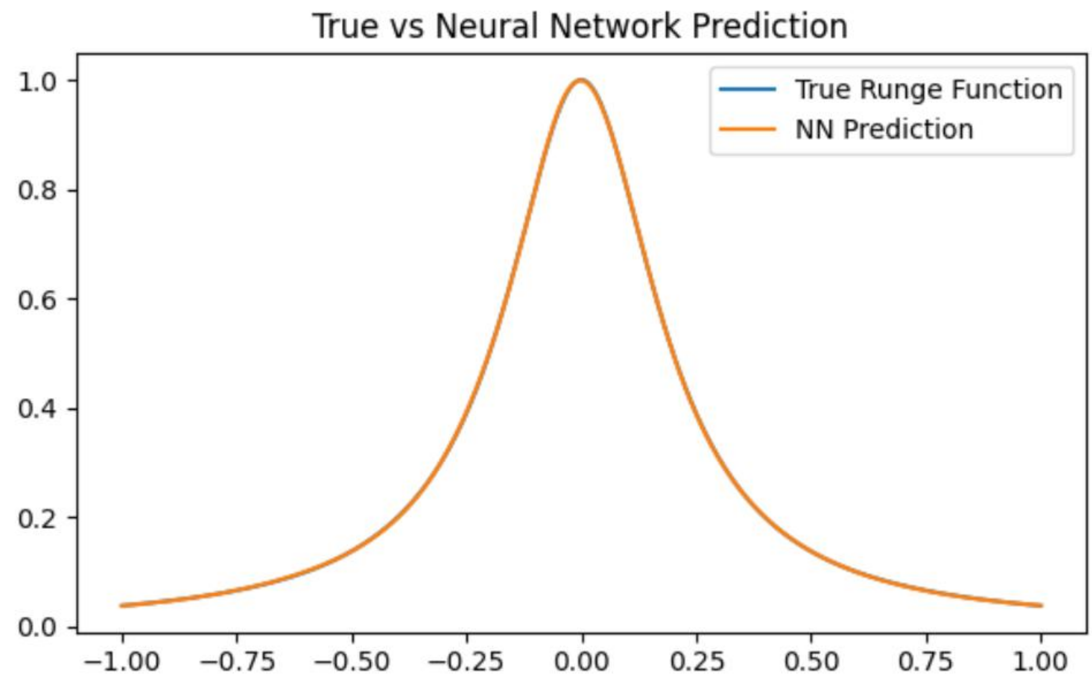

Figure 1   True vs Neural Network Prediction

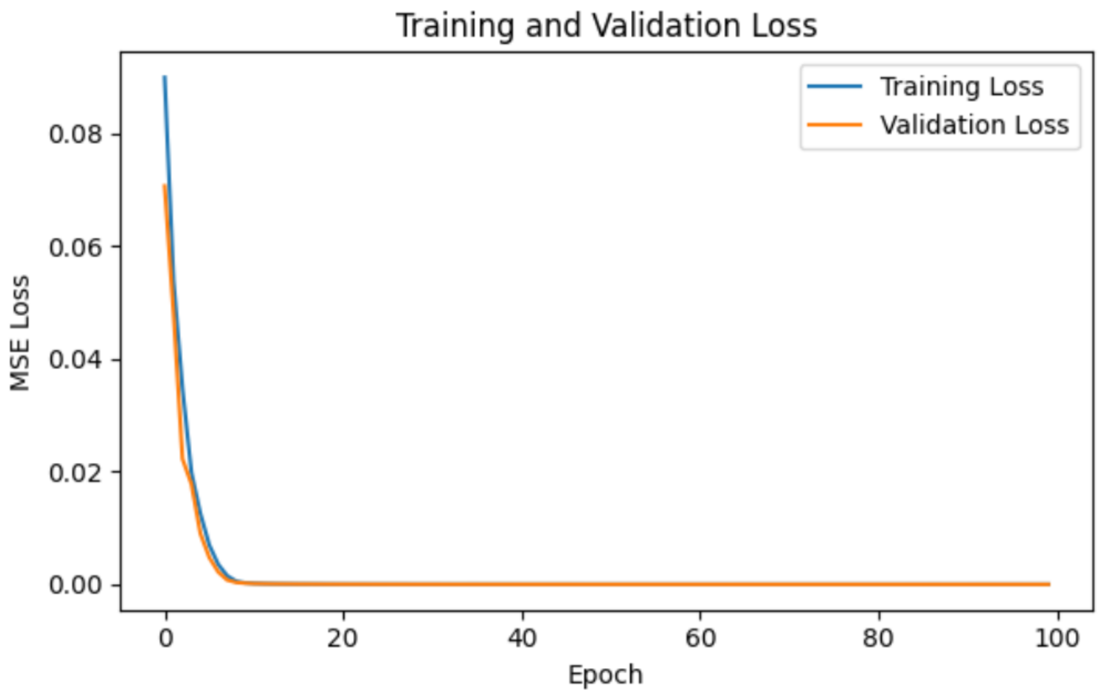

Figure 2   Training and Validation Loss

```
MSE: 0.000001, Max Error: 0.00259
```

Figure 3   MSE and max error

Code

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import matplotlib.pyplot as plt

# Data Preparation
def runge_function(x):
    return 1.0 / (1 + 25 * x**2)

 # Sample points
 np.random.seed(0)
 x_train = np.linspace(-1, 1, 1200)
 y_train = runge_function(x_train)
 x_val = np.linspace(-1, 1, 300)
 y_val = runge_function(x_val)

 # PyTorch tensors
 x_train_tensor = torch.FloatTensor(x_train).view(-1, 1)
 y_train_tensor = torch.FloatTensor(y_train).view(-1, 1)
 x_val_tensor = torch.FloatTensor(x_val).view(-1, 1)
 y_val_tensor = torch.FloatTensor(y_val).view(-1, 1)

 train_ds = TensorDataset(x_train_tensor, y_train_tensor)
 val_ds = TensorDataset(x_val_tensor, y_val_tensor)
 train_loader = DataLoader(train_ds, batch_size=128, shuffle=True)
 val_loader = DataLoader(val_ds, batch_size=128, shuffle=False)

 # Model definition (tanh activation)
 class Net(nn.Module):
     def __init__(self):
         super().__init__()
         self.fc1 = nn.Linear(1, 32)
         self.fc2 = nn.Linear(32, 32)
         self.fc3 = nn.Linear(32, 1)

     def forward(self, x):
         x = torch.tanh(self.fc1(x))
         x = torch.tanh(self.fc2(x))
         x = self.fc3(x)
         return x

 net = Net()
 optimizer = torch.optim.Adam(net.parameters(), lr=0.01)
 criterion = nn.MSELoss()

 # Training loop
 num_epochs = 100
 train_losses, val_losses = [], []

 for epoch in range(num_epochs):
     net.train()
     batch_train_losses = []
     for xb, yb in train_loader:
         optimizer.zero_grad()
         out = net(xb)
         loss = criterion(out, yb)
         loss.backward()
         optimizer.step()
```

```
        batch_train_losses.append(loss.item())
    train_losses.append(np.mean(batch_train_losses))

    net.eval()
    with torch.no_grad():
        val_pred = net(x_val_tensor)
        val_loss = criterion(val_pred, y_val_tensor)
        val_losses.append(val_loss.item())

# Plotting results
plt.figure(figsize=(12, 4))
# a. True vs Prediction
plt.subplot(1, 2, 1)
x_plot = np.linspace(-1, 1, 400)
y_true = runge_function(x_plot)
y_pred = net(torch.FloatTensor(x_plot).view(-1, 1)).detach().numpy()
plt.plot(x_plot, y_true, label="True Runge Function")
plt.plot(x_plot, y_pred, label="NN Prediction")
plt.legend()
plt.title("True vs Neural Network Prediction")

# b. Loss curves
plt.subplot(1, 2, 2)
plt.plot(train_losses, label="Training Loss")
plt.plot(val_losses, label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("MSE Loss")
plt.legend()
plt.title("Training and Validation Loss")

plt.tight_layout()
plt.show()

# Error computation
mse = np.mean((y_true - y_pred.flatten())**2)
max_error = np.max(np.abs(y_true - y_pred.flatten()))
print(f"MSE: {mse:.6f}, Max Error: {max_error:.5f}")
```