Machine Learning Assignment 02

313652008 黃睿帆

September 14, 2025

Programming assignment

Use a neural network to approximate the Runge function

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

Here we using AI tools (NOTE: Some of the following Python codes are generated by chat-GPT, and the Python codes were runned by Colab), to acheive our goal:

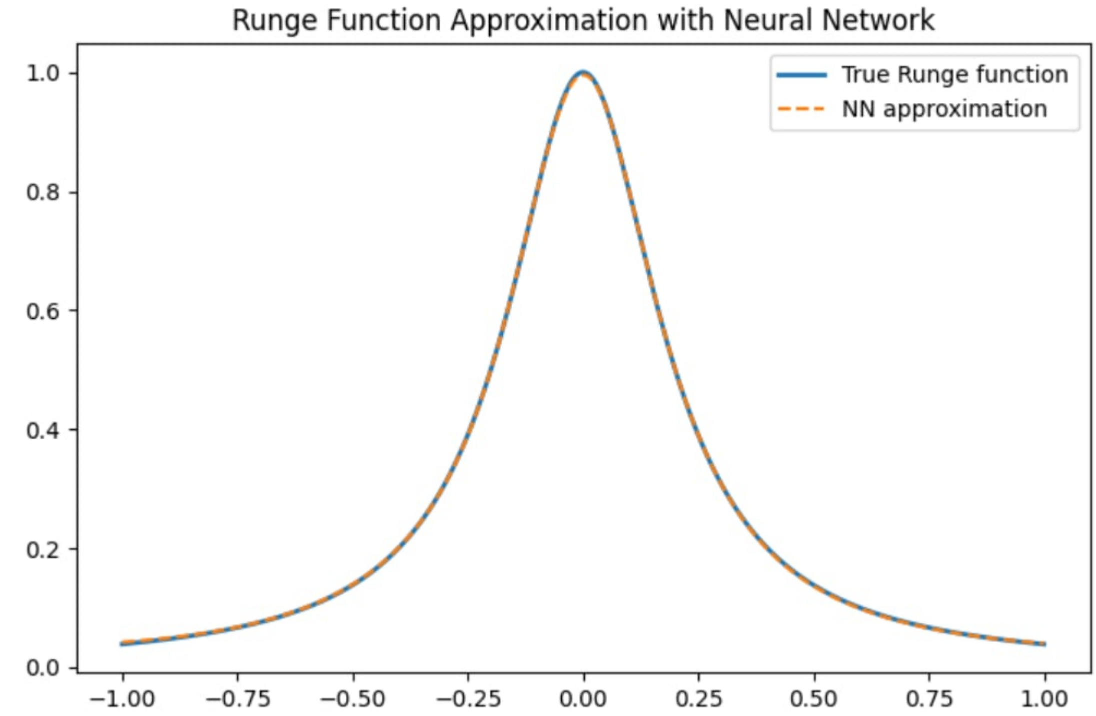
1. Plotting the true function and the neural network prediction together.
2. Showing the training/validation loss curves.
3. Computing and report errors (MSE or max error).

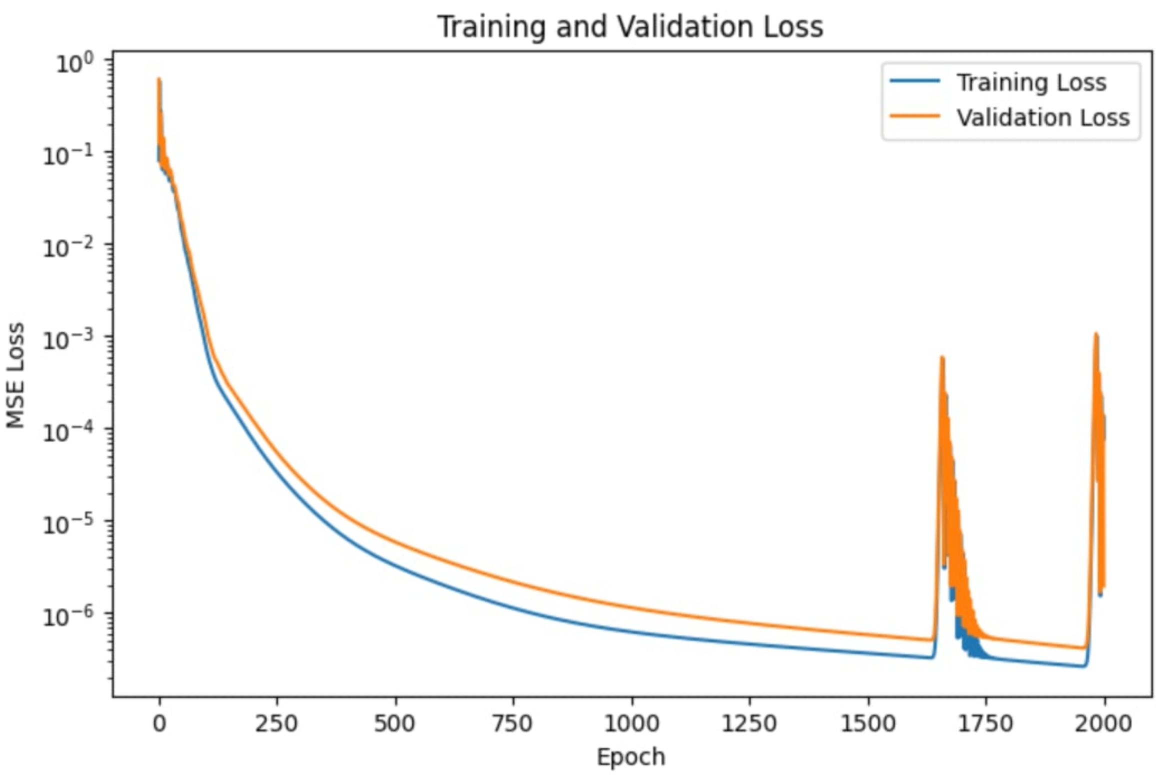
1. Using hyperbolic tangent tanh

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

# --- Define Runge function ---
def runge(x):
    return 1.0 / (1 + 25 * x**2)

# --- Training data ---
np.random.seed(0)
N_train, N_val = 200, 100
x_train = np.random.uniform(-1, 1, N_train)
y_train = runge(x_train)
x_val = np.random.uniform(-1, 1, N_val)
y_val = runge(x_val)

# Convert to torch tensors
x_train_t = torch.tensor(x_train, dtype=torch.float32).unsqueeze(1)
y_train_t = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
x_val_t = torch.tensor(x_val, dtype=torch.float32).unsqueeze(1)
y_val_t = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)

# --- Neural network model ---
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(1, 50),
            nn.Tanh(),
            nn.Linear(50, 50),
            nn.Tanh(),
            nn.Linear(50, 1)
        )
    def forward(self, x):
        return self.layers(x)
```

```
model = Net()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# --- Training loop ---
n_epochs = 2000
train_losses, val_losses = [], []

for epoch in range(n_epochs):
    # Training
    model.train()
    optimizer.zero_grad()
    y_pred = model(x_train_t)
    loss = criterion(y_pred, y_train_t)
    loss.backward()
    optimizer.step()

    # Validation
    model.eval()
    with torch.no_grad():
        y_val_pred = model(x_val_t)
        val_loss = criterion(y_val_pred, y_val_t)

    train_losses.append(loss.item())
    val_losses.append(val_loss.item())

# --- Predictions ---
x_plot = np.linspace(-1, 1, 500)
y_true = runge(x_plot)

with torch.no_grad():
    y_pred_plot = model(torch.tensor(x_plot, dtype=torch.float32).unsqueeze(1)).numpy().flatten()

# --- Compute errors ---
mse = np.mean((y_true - y_pred_plot)**2)
max_err = np.max(np.abs(y_true - y_pred_plot))

print(f"Mean Squared Error: {mse:.6f}")
print(f"Max Error: {max_err:.6f}")

# --- Plot true function vs prediction ---
plt.figure(figsize=(8,5))
plt.plot(x_plot, y_true, label="True Runge function", linewidth=2)
plt.plot(x_plot, y_pred_plot, label="NN approximation", linestyle='--')
plt.legend()
plt.title("Runge Function Approximation with Neural Network")
plt.show()

# --- Plot training/validation loss ---
plt.figure(figsize=(8,5))
plt.plot(train_losses, label="Training Loss")
plt.plot(val_losses, label="Validation Loss")
plt.yscale("log")
plt.xlabel("Epoch")
plt.ylabel("MSE Loss")
plt.legend()
plt.title("Training and Validation Loss")
plt.show()
```

```
Mean Squared Error: 0.000002
Max Error: 0.004867
```





2. Using polynomials

Note that using equispaced nodes is east to have Runge's phenomenon, which is different from the result using Chebyshev nodes.

```
import numpy as np
import matplotlib.pyplot as plt

# --- Define Runge function ---
def runge(x):
    return 1.0 / (1 + 25 * x**2)

# --- Settings ---
N_nodes = 15   # number of interpolation nodes (degree = N_nodes-1)
x_plot = np.linspace(-1, 1, 500)
y_true = runge(x_plot)

# --- 1. Equispaced nodes interpolation ---
x_eq = np.linspace(-1, 1, N_nodes)
y_eq = runge(x_eq)
coeff_eq = np.polyfit(x_eq, y_eq, N_nodes-1)
y_eq_poly = np.polyval(coeff_eq, x_plot)

# --- 2. Chebyshev nodes interpolation ---
k = np.arange(1, N_nodes+1)
x_cheb = np.cos((2*k-1)/(2*N_nodes) * np.pi)  # Chebyshev nodes in [-1,1]
y_cheb = runge(x_cheb)
coeff_cheb = np.polyfit(x_cheb, y_cheb, N_nodes-1)
y_cheb_poly = np.polyval(coeff_cheb, x_plot)

# --- Compute errors ---
def compute_errors(y_true, y_pred):
    mse = np.mean((y_true - y_pred)**2)
    max_err = np.max(np.abs(y_true - y_pred))
    return mse, max_err

mse_eq, max_eq = compute_errors(y_true, y_eq_poly)
mse_cheb, max_cheb = compute_errors(y_true, y_cheb_poly)

print("Equispaced Polynomial Approximation:")
print(f"  MSE = {mse_eq:.6e}, Max Error = {max_eq:.6e}")
print("Chebyshev Polynomial Approximation:")
print(f"  MSE = {mse_cheb:.6e}, Max Error = {max_cheb:.6e}")

# --- Plot results ---
plt.figure(figsize=(9,6))
plt.plot(x_plot, y_true, 'k', linewidth=2, label="True Runge function")
plt.plot(x_plot, y_eq_poly, 'r--', label="Equispaced polynomial")
plt.plot(x_plot, y_cheb_poly, 'b--', label="Chebyshev polynomial")
plt.scatter(x_eq, y_eq, color='red', s=40, marker='o', label="Equispaced nodes")
plt.scatter(x_cheb, y_cheb, color='blue', s=40, marker='x', label="Chebyshev nodes")
plt.legend()
plt.title(f"Polynomial Interpolation of Runge Function (degree {N_nodes-1})")
plt.show()
```

```
Equispaced Polynomial Approximation:
  MSE = 3.132079e+00, Max Error = 7.191540e+00
Chebyshev Polynomial Approximation:
  MSE = 6.219337e-04, Max Error = 4.659411e-02
```