```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
torch.manual_seed(0)
np.random.seed(0)

def runge(x):
    return 1.0/(1.0+25.0*x**2)

N_total=2000
x_all = np.random.uniform(-1, 1, size=(N_total, 1)).astype(np.float32)
y_all = runge(x_all).astype(np.float32)

x_train, x_val, y_train, y_val = train_test_split(x_all, y_all, test_size=0.2,
random_state=1)#訓練集和測試集

X_train=torch.from_numpy(x_train)
Y_train=torch.from_numpy(y_train)
X_val=torch.from_numpy(x_val)
Y_val=torch.from_numpy(y_val)

class MLP(nn.Module):
    def __init__(self, hidden_sizes=[32, 32]):
        super().__init__()
        layers = []
        in_dim = 1
        for h in hidden_sizes:
            layers.append(nn.Linear(in_dim, h))
            layers.append(nn.Tanh())
            in_dim = h
```

```
        layers.append(nn.Linear(in_dim, 1))
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)

model =MLP(hidden_sizes=[64, 64])
device=torch.device('cpu')
model.to(device)



#training setup
lr=1e-4
epochs=500
batch_size=64
criterion=nn.MSELoss()
optimizer=optim.Adam(model.parameters(), lr=lr)
train_losses=[]
val_losses=[]

dataset = torch.utils.data.TensorDataset(X_train, Y_train)#let machine know the
data
loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
shuffle=True)



for ep in range(epochs):
    model.train()#start training by using training data
    running_loss=0.0
    for xb, yb in loader:
        xb=xb.to(device)
        yb=yb.to(device)
        optimizer.zero_grad()
        out=model(xb)#the prediction y by using model
```

```
        loss=criterion(out, yb)# the MSE of pred_y and y
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * xb.size(0)
    epoch_train_loss=running_loss/len(X_train)
    train_losses.append(epoch_train_loss)

    #test
    model.eval()
    with torch.no_grad():
        ypred_val=model(X_val.to(device))
        epoch_val_loss=criterion(ypred_val, Y_val.to(device)).item()
        val_losses.append(epoch_val_loss)

    if (ep+1)%25 == 0 or ep == 0:
        print(f'Epoch {ep+1}/{epochs} train MSE={epoch_train_loss:.6f} val
MSE={epoch_val_loss:.6f}')

#grid
x_grid=np.linspace(-1, 1, 500, dtype=np.float32).reshape(-1, 1)#convert to an
array with 1 column
y_true_grid=runge(x_grid)

model.eval()
with torch.no_grad():#計算張量時不計算梯度
    y_pred_grid=model(torch.from_numpy(x_grid)).cpu().numpy()#create tensor from
a numpy array and numpy() converts a tensor object into an numpy.ndarray objec

#compute metrics
with torch.no_grad():
    y__val_pred=model(X_val).cpu().numpy()
mse_val=np.mean((y__val_pred-y_val)**2)
max_abs_err_val=np.max(np.abs(y__val_pred-y_val))
```

```
print("\nFinal validation results:")
print(f"Validation MSE: {mse_val:.8f}")
print(f"Validation max absolute error: {max_abs_err_val:.8f}")

#plot
plt.figure(figsize=(6, 4))
plt.plot(x_grid, y_true_grid, label='true f(x)')
plt.plot(x_grid, y_pred_grid, label='NN prediction', linestyle='--')
plt.scatter(x_train[:200], y_train[:200], s=10, alpha=0.3, label='train samples
(subset)')
plt.legend()
plt.title('Runge function: true vs NN prediction')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.tight_layout()
plt.show()

#loss curves
plt.figure(figsize=(6, 4))
plt.plot(train_losses, label='train MSE')
plt.plot(val_losses, label='val MSE')
plt.yscale('log')
plt.xlabel('epoch')
plt.ylabel('MSE (log scale)')
plt.legend()
plt.title('Training and validation loss curves')
plt.tight_layout()
plt.show()
```