Assignment 2

✍️ Written assignment

1. Read Deep Learning: An Introduction for Applied Mathematicians. Consider a network as defined in (3.1) and (3.2). Assume that $n_L = 1$, find an algorithm to calculate $\nabla a^{[L]}(x)$.

From (3.1), $a^{[1]} = x \in \mathbb{R}^{n_1}$.

From (3.2), $a^{[l]} = \sigma(W^{[l]} a^{[l-1]} + b^{[l]}) \in \mathbb{R}^{n_l}$, for $n_l = 2, \dots, L$

i.e. $a^{[l]} = \sigma(z^{[l]})$, where $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$.

Algorithm:

1. Set $a^{[l]} = x$
2. Compute $z^{[l]}$, for $l = 2, \dots, L$.
3. $\delta^{[L]} = \dfrac{\partial a^{[L]}}{\partial z^{[L]}}$. Since $n_L = 1$, $\delta^{[L]}$ is a scalar.
4. $\delta^{[l]} = \sigma'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]}$, for $l = L-1, \dots, 2$,

   where $\circ$ is Hadamard product (componentwise)
5. Using $z^{[2]} = W^{[2]} x + b^{[2]}$ to obtain

   $\nabla_x a^{[L]}(x) = \dfrac{\partial C}{\partial x} = (W^{[2]})^T \delta^{[2]} \in \mathbb{R}^{n_1}$ (vector).

🧑🏻‍💻 Programming assignment

1. Use a neural network to approximate the Runge function

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

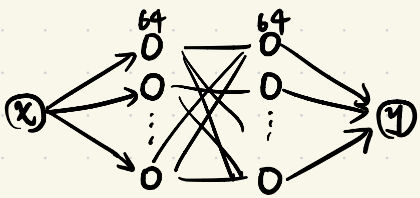
   Write a short report (1–2 pages) explaining method, results, and discussion including

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

Method: feedforward neural network

dataset: uniformly sampling 200 points from [-1, 1] and the data was split into 80% training and 20% validation sets.

network architecture:



using tanh activations.

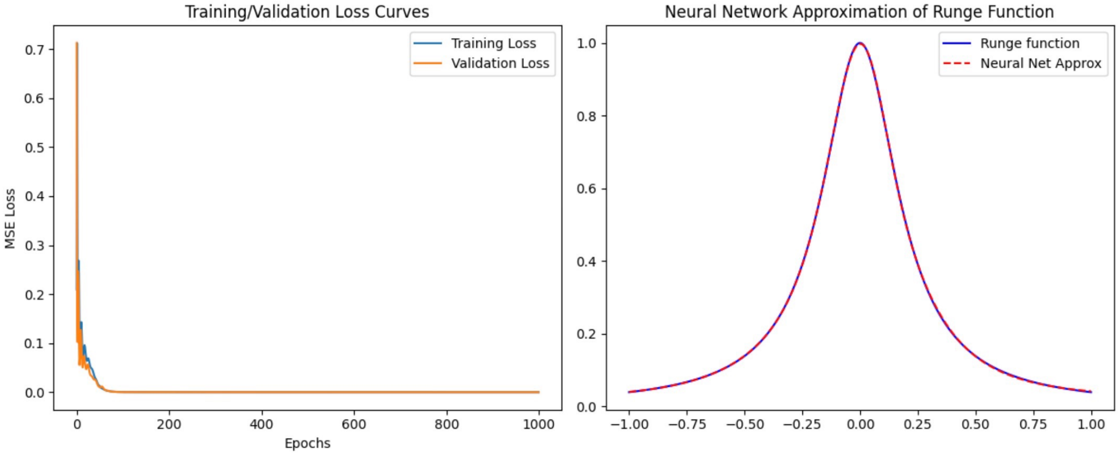
loss function: MSE

optimizer: Adam, learning rate 0.01, 1000 epoch

```python
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

#Runge function
def runge_function(x):
    return 1 / (1 + 25 * x**2)

#dataset and training/validation sets
x = np.linspace(-1, 1, 200).reshape(-1, 1).astype(np.float32)
y = runge_function(x).astype(np.float32)
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2,
random_state=42)
x_train_tensor = torch.from_numpy(x_train)
y_train_tensor = torch.from_numpy(y_train)
x_val_tensor = torch.from_numpy(x_val)
y_val_tensor = torch.from_numpy(y_val)

#neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(1, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 1)
        self.activation = nn.Tanh()

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

net = Net()

#training
criterion = nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr=0.01)

epochs = 1000
train_losses, val_losses = [], []

for epoch in range(epochs):
    #training
    optimizer.zero_grad()
    outputs = net(x_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    loss.backward()
    optimizer.step()

    #validation
    with torch.no_grad():
        val_outputs = net(x_val_tensor)
        val_loss = criterion(val_outputs, y_val_tensor)

    #record losses
    train_losses.append(loss.item())
    val_losses.append(val_loss.item())

    if (epoch+1) % 200 == 0:
        print(f"Epoch [{epoch+1}/{epochs}] "
              f"Train Loss: {loss.item():.6f}, "
              f"Val Loss: {val_loss.item():.6f}")

#training/validation loss curves
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(train_losses, label="Training Loss")
plt.plot(val_losses, label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("MSE Loss")
plt.title("Training/Validation Loss Curves")
plt.legend()

#test the network and plot approximation
x_test = np.linspace(-1, 1, 500).reshape(-1, 1).astype(np.float32)
y_test = runge_function(x_test)

x_test_tensor = torch.from_numpy(x_test)
y_pred = net(x_test_tensor).detach().numpy()

plt.subplot(1, 2, 2)
plt.plot(x_test, y_test, label="Runge function", color="blue")
plt.plot(x_test, y_pred, label="Neural Net Approx", color="red", linestyle="--")
plt.legend()
plt.title("Neural Network Approximation of Runge Function")

plt.tight_layout()
plt.show()

#MSE
with torch.no_grad():
    y_pred_tensor = net(x_test_tensor)
    test_mse = criterion(y_pred_tensor, torch.from_numpy(y_test))
    print(f"Final Test MSE: {test_mse.item():.6f}")
```

Results :

```
Epoch [200/1000] Train Loss: 0.000052, Val Loss: 0.000041
Epoch [400/1000] Train Loss: 0.000017, Val Loss: 0.000015
Epoch [600/1000] Train Loss: 0.000008, Val Loss: 0.000008
Epoch [800/1000] Train Loss: 0.000004, Val Loss: 0.000004
Epoch [1000/1000] Train Loss: 0.000002, Val Loss: 0.000002
```




```
Final Test MSE: 0.000002
```

- The training and validation loss curves both decreased smoothly ⇒ The network was able to learn the func. without overfitting.
- The trained network produced predictions that closely match the true Runge function.