Machine Learning
Assignment 2 Updated

CHANG Yung-Hsuan (張永璿)

111652004

eiken.sc11@nycu.edu.tw

September 19, 2025

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 $\nabla a^{[L]}(x)$.

Solution. The equation (3.1) in the paper is

$$a^{[1]} = x \in \mathbb{R}^{n_1}$$

and the equation (3.2) in the paper is

$$a^{[l]} = \sigma\left(W^{[l]} a^{[l-1]} + b^{[l]}\right) \in R^{n_l} \qquad \text{for} \qquad l = 2, 3, \ldots, L.$$

We can use the same trick as we did during lectures: forward passing and backpropagation, which is essentially the chain rule; we just need to be more careful about dimensions and products.

First, we define

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}.$$

Then, naively, we write

$$\begin{aligned} \frac{\partial a^{[l]}}{\partial a^{[l-1]}} &= \frac{\partial a^{[l]}}{\partial z^{[l]}} \cdot \frac{\partial z^{[l]}}{\partial a^{[l-1]}} \\ &= \sigma'\left(z^{[l]}\right) \cdot W^{[l]} \end{aligned}$$

where

$$\sigma'\left(z^{[l]}\right) = \begin{pmatrix} \sigma'\left(z_1^{[l]}\right) & & 0 \\ & \ddots & \\ 0 & & \sigma'\left(z_{n_l}^{[l]}\right) \end{pmatrix}$$

$$\in \mathbb{R}^{n_l \times n_l}$$

as $\sigma\left(z_i^{[l]}\right)$ only relies on the i-th input, and

$$W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}.$$

The results are quite nice. Thus, we can have

$$\begin{aligned} \frac{\partial a^{[L]}}{\partial a^{[1]}} &= \frac{\partial a^{[L]}}{\partial a^{[L-1]}} \cdot \frac{\partial a^{[L-1]}}{\partial a^{[L-2]}} \cdot \ldots \cdot \frac{\partial a^{[2]}}{\partial a^{[1]}} \\ &= \left(\sigma'\left(z^{[L]}\right) \cdot W^{[L]}\right) \cdot \left(\sigma'\left(z^{[L-1]}\right) \cdot W^{[L-1]}\right) \cdot \ldots \cdot \left(\sigma'\left(z^{[2]}\right) \cdot W^{[2]}\right) \\ &\in \mathbb{R}^{n_L \times n_1}, \end{aligned}$$

which suggests this is a row vector as $n_L = 1$. Hence, we can use the following algorithm to both compute and save (temporarily) data:

1. (Forward Pass)
   i. Set $a^{[1]} = x$.
   ii. For l in $(2, 3, \ldots, L)$:
      - $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ (Saved to calculate $s^{[l]}$.)
      - $a^{[l]} = \sigma\left(z^{[l]}\right)$ (Saved to calculate $z^{[l+1]}$.)
      - $s^{[l]} = \mathbf{1}_{n_l}{}^{\mathrm{T}} \operatorname{diag} \sigma'\left(z^{[l]}\right)$ (Saved for the backward pass part; $s^{[l]} \in \mathbb{R}^{1 \times n_l}$ as σ' is diagonal)
2. (Backward Pass)
   i. Set $g^{[L]} = a^{[L]} = 1$ as $n_L = 1$. (This initializes the gradient g.)
   ii. For l in $(L, L-1, \ldots, 2)$:
      - $g^{[l-1]} = \left(g^{[l]} \odot s^{[l]}\right) W^{[l]}$ (Elementwise product, then propagate through $W^{[l]}$.)

The output will be $g^{[1]}$, which is the row vector of the gradient $\nabla a^{[L]}(x)$.

2. There are unanswered questions during the lecture, and there are likely more questions we haven't covered. Take a moment to think about them and write them down here.

   Answer. How efficient does a hidden layer perform to approximate a monomial? Fix the number of data points to a constant or a scalar times the power of the monomial. Do we have a performance benchmark as we evaluate sorting algorithms to evaluate the effectiveness of a one-hidden layer neural network?

3. 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

   a. plot the true function and the neural network prediction together,

   b. show the training/validation loss curves, and

   c. compute and report errors (MSE or max error).

   Solution. As suggested, I use neural network with one input layer with one feature, two hidden layers with 64 neurons each, and one output layer with a neuron. The dataset consists of 100 evenly distributed data points with `train_test_split` (with `random_state=4`) to training set 70%, validation set 15%, and testing set 15%. I use tanh as the activation function, and MSE is the loss function, and the Adam is the optimizer.

   One can observe that the approximation is quite decent for x near 0 but it deviates away from 0. This coincides the Runge phenomenon.

   At around epoch 500, the validation loss (absolute error) becomes around 10^{-5}, and the testing loss reported is 1.2×10^{-5} for MSE and 6.753×10^{-3} for max error.

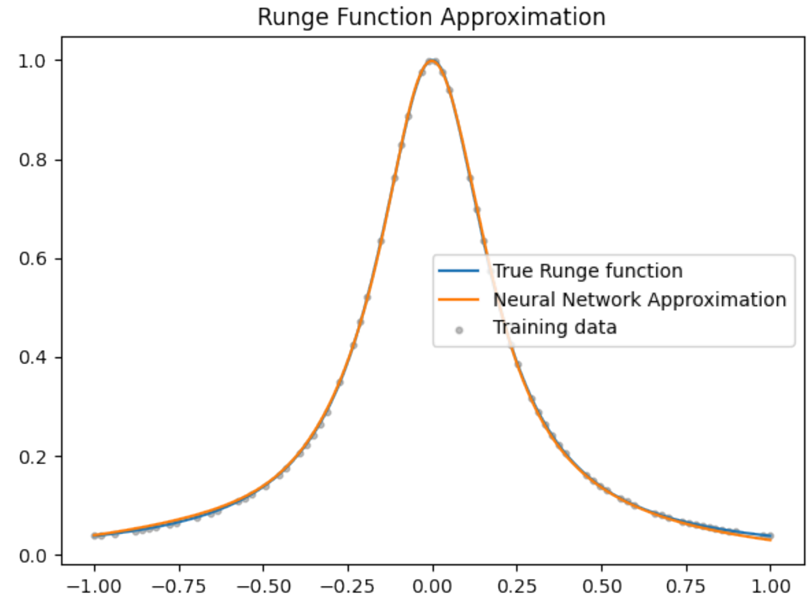

Figure 1. Runge Function Approximation with Neural Network

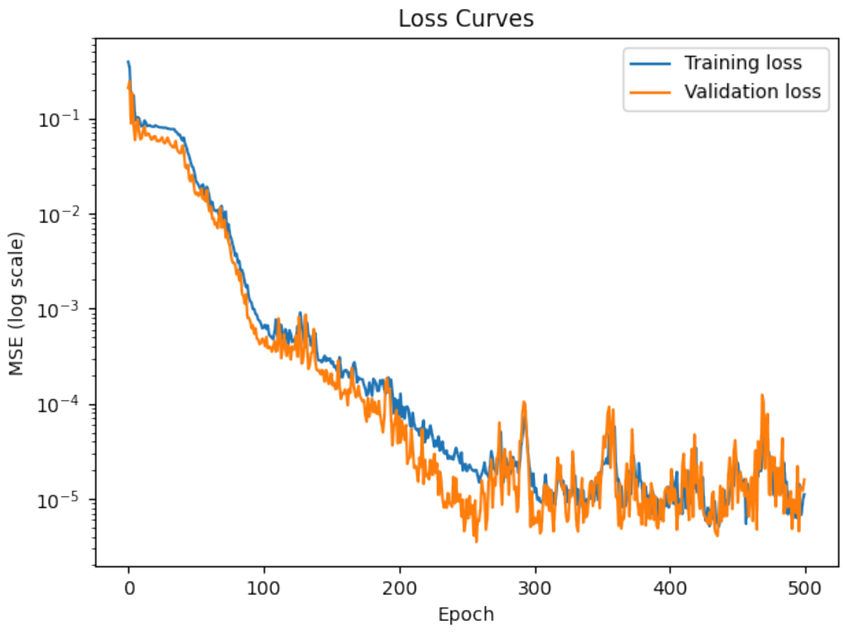

Figure 2. Training and Validation Loss Curves vs. Epoch

Raw Code:

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
```

```python
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
FIG_DIR = os.path.join(BASE_DIR, "figures")
os.makedirs(FIG_DIR, exist_ok=True)

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

# Data generation
N = 100
x = np.linspace(-1, 1, N).reshape(-1, 1)
y = runge(x)

# Split into train (70%) and temp (30%)
x_train, x_temp, y_train, y_temp = train_test_split(x, y, test_size=0.3,
random_state=4)
# Split temp into validation (15%) and test (15%)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size=0.5,
random_state=4)

# Build model
model = keras.Sequential([
    layers.Dense(64, activation="tanh", input_shape=(1,)),
    layers.Dense(64, activation="tanh"),
    layers.Dense(1)
])

model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.01),
              loss="mse")

# Train
history = model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=500,
    verbose=1
)

x_dense = np.linspace(-1, 1, 500).reshape(-1, 1)
y_dense_pred = model.predict(x_dense)

# Plot Runge function vs NN approximation
plt.figure(figsize=(7,5))
plt.plot(x_dense, runge(x_dense), label="True Runge function")
plt.plot(x_dense, y_dense_pred, label="Neural Network Approximation")
```

```python
plt.scatter(x_train, y_train, s=10, c="gray", alpha=0.5, label="Training data")
plt.legend()
plt.title("Runge Function Approximation")
plt.savefig(os.path.join(FIG_DIR, "runge_approx.png"), transparent=True)
# plt.show()

# Plot the loss curves
plt.figure(figsize=(7,5))
plt.plot(history.history["loss"], label="Training loss")
plt.plot(history.history["val_loss"], label="Validation loss")
plt.yscale("log")
plt.xlabel("Epoch")
plt.ylabel("MSE (log scale)")
plt.legend()
plt.title("Loss Curves")
plt.savefig(os.path.join(FIG_DIR, "loss_curves.png"), transparent=True)
# plt.show()

# Test set
test_mse = model.evaluate(x_test, y_test, verbose=0)
y_test_pred = model.predict(x_test)
max_err = np.max(np.abs(y_test - y_test_pred))

print(f"Test MSE: {test_mse:.6f}")
print(f"Max error on test set: {max_err:.6f}")
```