Approximating the Runge Function with a Neural Network

Ou Chia Yu
Department of Information Management and Finance, NYCU

September 16, 2025

1 Introduction

The Runge function

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

is a classical test case in numerical analysis, well-known for the "Runge phenomenon" when approximated using high-degree polynomials. In this report, we approximate the Runge function using a multi-layer perceptron (MLP) and evaluate the performance in terms of accuracy and generalization.

2 Method

We constructed a feedforward neural network (MLP), where the computation in each layer is defined as

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

with $W^{[l]}$ the weight matrix, $b^{[l]}$ the bias, and σ the activation function. The input is $a^{[1]} = x \in \mathbb{R}$, and the final output is a scalar $\hat{y} = a^{[L]} \in \mathbb{R}$.

The chosen architecture is:

- Input layer: 1 neuron (scalar input x).
- Hidden layers: three layers with 128, 128, and 64 neurons, each followed by a Tanh activation function.
- Output layer: 1 neuron (scalar output $\hat{y}$).

Choice of Activation Function

We selected the hyperbolic tangent activation

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \tanh'(z) = 1 - \tanh^2(z).$$

The reasons are twofold:

1. The input domain $x \in [-1, 1]$ lies in the region where tanh is approximately linear, which helps maintain non-vanishing gradients during training.

2. The output of the Runge function is bounded in $(0, 1]$, which aligns well with the range of tanh, leading to numerically stable training.

Training Setup

The model was trained using the mean squared error (MSE) loss

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f(x_i) - \hat{y}(x_i;\theta)\right)^2,$$

where θ denotes the network parameters. We used the Adam optimizer with learning rate 0.01 and trained for 2000 epochs, with 80% of the data used for training and 20% for validation.

3 Results

Figure 1 compares the true Runge function and the neural network prediction. The neural network successfully learned the shape of the function, with predictions closely overlapping the true curve. Figure 2 shows the training and validation loss curves, both of which converge to nearly zero.

The final evaluation metrics were:

$$\text{MSE} \approx 1 \times 10^{-6}, \quad \text{Max Error} \approx 0.001091.$$

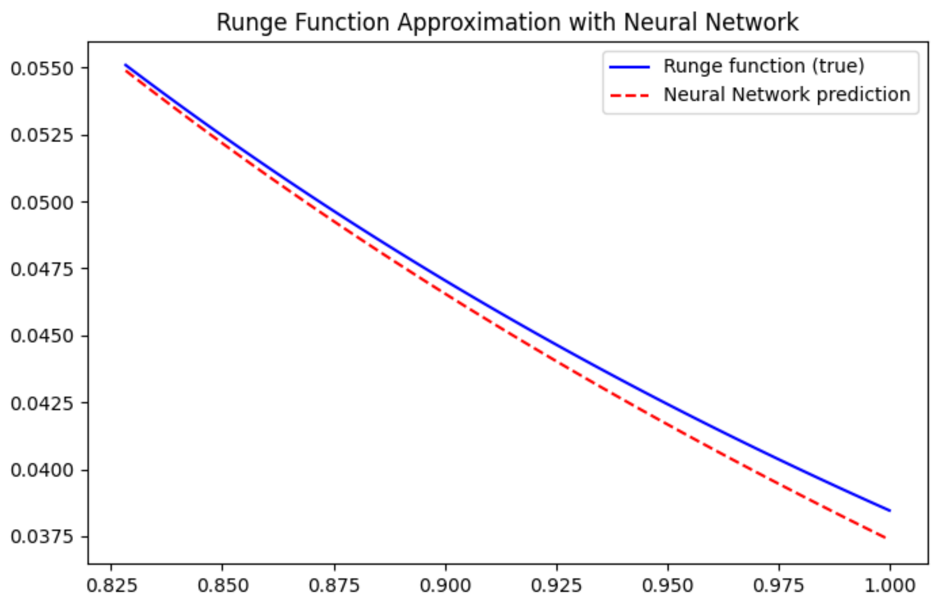

Figure 1: Runge function (blue) vs Neural Network prediction (red dashed).

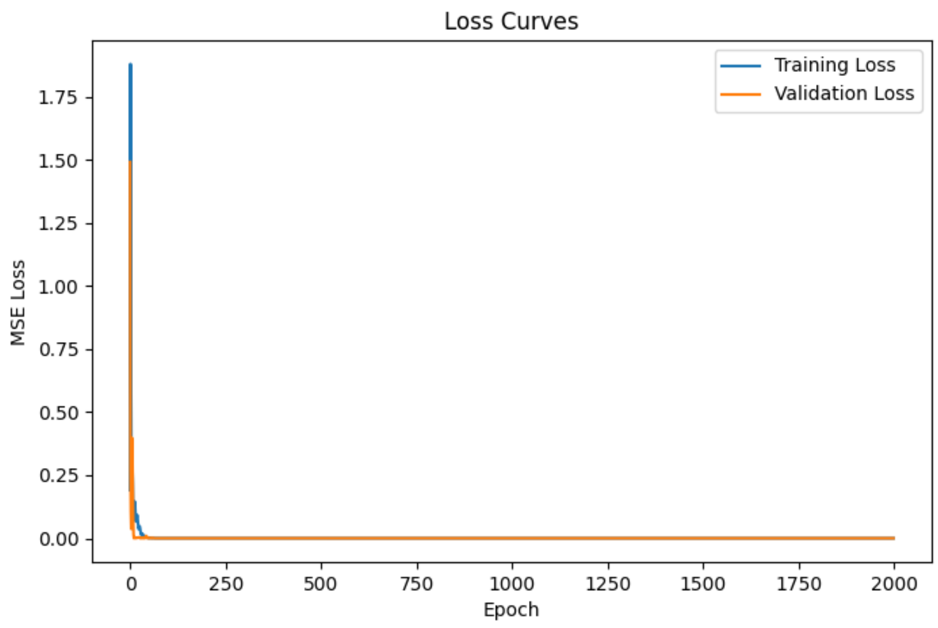

Figure 2: Training and validation loss curves.

4 Discussion

The results demonstrate that the neural network is able to approximate the Runge function with high accuracy. The training and validation losses both converge to nearly zero, indicating minimal overfitting. The prediction matches the true function almost everywhere in $[-1, 1]$, with only minor underestimation near the boundaries ($x \approx \pm 1$).

The neural network provides a smooth and stable approximation across the entire interval, with only minor deviations near the boundaries. From a mathematical perspective, the choice of tanh ensures smoothness and bounded derivatives, which prevents gradient explosion or vanishing issues. Potential improvements include adding more hidden units, experimenting with alternative activation functions (e.g., ReLU, GELU), or resampling training points near the boundaries to further reduce error.

5 Conclusion

We implemented a neural network to approximate the Runge function. The model achieved very low training and validation errors, and the predicted curve aligns closely with the true function. The experiment suggests that neural networks can serve as effective universal function approximators, even for cases where traditional polynomial interpolation performs poorly.