ML Week3 Assignment

WANG TZU YI

September 21, 2025

1 Introduction

In this assignment, I use a neural network to approximate both the Runge function and its derivative. The activation functions include `Tanh` and `LeakyReLU`.

The definition of the Runge function is:

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

2 Methodology

2.1 Model Architecture

We use a multi-layer perceptron (MLP) to approximate the Runge function. The mathematical formulation is as follows:

$$f(x) = W_3 \cdot \sigma_2 \left(W_2 \cdot \sigma_1 \left(W_1 \cdot x + b_1 \right) + b_2 \right) + b_3$$

where σ_1, σ_2 are activation functions, and W_i, b_i are the weight matrices and bias vectors, respectively. The network architecture is shown below:

Layer	Units	Activation
Input	1	-
Dense 1	4	Tanh or LeakyReLU
Dense 2	4	Tanh or LeakyReLU
Output	1	Linear

Table 1: Architecture of the MLP model used to approximate the Runge function.

2.2 Loss Function

In this task, the neural network is trained not only to approximate the Runge function $f(x)$, but also its first-order derivative $f'(x)$. Therefore, the overall loss function is composed of two components:

1. **Function Loss**: Measures the mean squared error (MSE) between the predicted function value $\hat{f}(x)$ and the true function value $f(x)$:

$$\mathcal{L}_{\text{func}} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{f}(x_i) - f(x_i) \right)^2$$

2. **Derivative Loss**: Measures the MSE between the predicted derivative $\hat{f}'(x)$ (computed via automatic differentiation) and the true derivative $f'(x)$:

$$\mathcal{L}_{\text{deriv}} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{f}'(x_i) - f'(x_i) \right)^2$$

The total loss is the sum of both components:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{func}} + \mathcal{L}_{\text{deriv}}$$

This combined loss ensures that the network learns not only to fit the function values but also to capture the underlying shape and curvature of the function. The derivative loss encourages smoother and more physically consistent approximations, especially in regions where the function changes rapidly.

2.3 Assumptions

- The training data are sampled from a smooth and noise-free function.
- The input data x are normalized to lie within the interval $[-1, 1]$.
- A feedforward neural network with two hidden layers and Tanh or LeakyReLU activations is sufficient to approximate the underlying function.
- The loss function (mean squared error) is minimized via the Adam optimizer and converges to a good local minimum.

3 Experiments

3.1 Datasets

To generate the dataset, I uniformly sampled 200 points in the interval $[-1, 1]$ using `np.linspace`. This provides dense coverage across the domain, particularly near the edges where the Runge function exhibits rapid changes.

The dataset was then split into:

- **80% for training** – used to fit the neural network.
- **20% for validation** – used to evaluate the model's generalization performance.

3.2 Hyperparameters Setting

The model is trained using the Adam optimizer with a learning rate of 0.001, and the the training is performed for 500 epochs with a batch size of 32.

3.3 Evaluation Metrics

To quantitatively assess the model's performance in approximating the Runge function, we use four standard regression metrics: **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, **Maximum Absolute Error (Max Error)**, and the **Coefficient of Determination** (R^2).

Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

MSE penalizes larger errors more severely due to the square term, making it sensitive to outliers. A lower MSE indicates better overall performance in terms of squared deviation.

Maximum Absolute Error (Max Error):

$$\text{Max Error} = \max_i |\hat{y}_i - y_i|$$

This metric indicates the worst-case prediction error. It is useful for identifying extreme deviations, which may be especially important in applications where robustness is critical.

4 Results

4.1 Quantitative Results

Model	MSE(fx)	MAX(fx)	MSE(dfx)	MAX(dfx)
MLP (Tanh)	0.000030	0.013460	0.002088	0.104605
MLP (LeakyReLU)	0.000395	0.049640	0.008853	0.192673

Table 2: Quantitative evaluation of model performance on the Runge function.

4.2 Prediction

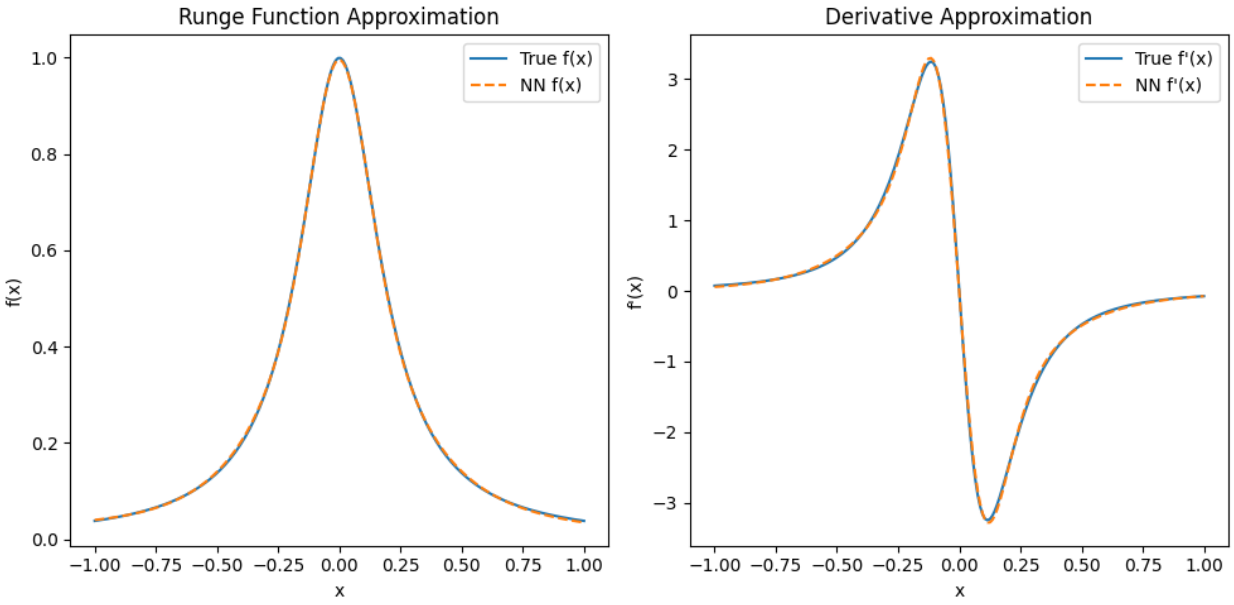

Figure 1: Prediction of Runge Function and it Derivative

4.3 Loss curve

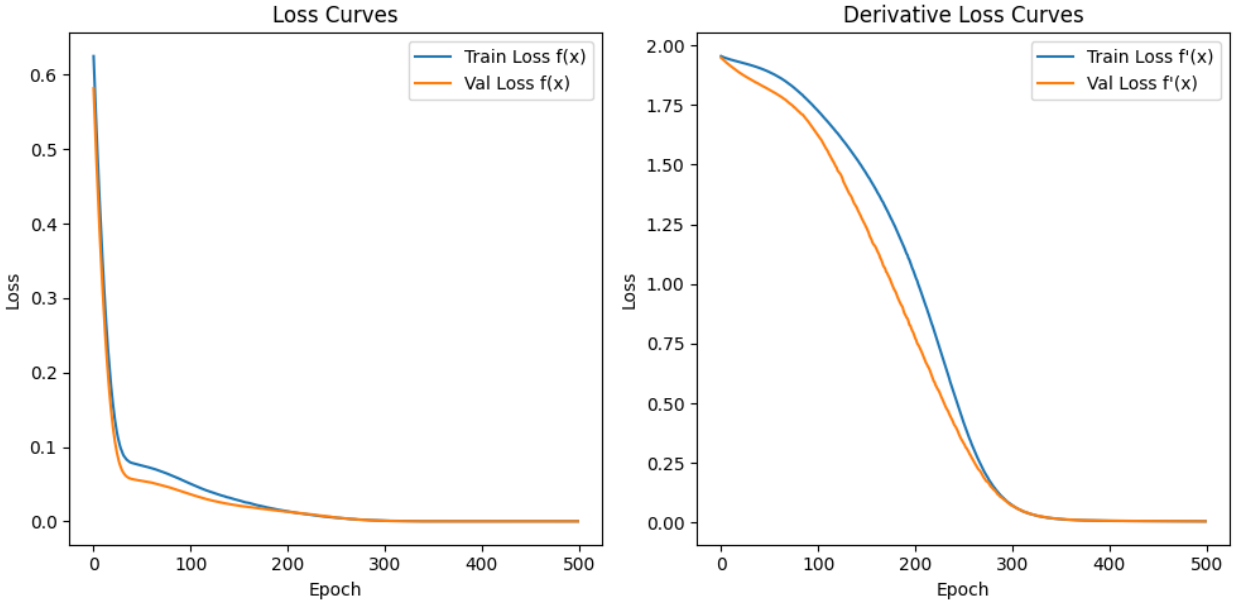

Figure 2: Loss of Runge Function and it Derivative

5 Discussion

The experimental results demonstrate that a relatively shallow neural network, equipped with Tanh activations, is capable of effectively approximating both the Runge function and its first-order derivative over the interval $[-1, 1]$. The model achieved low mean squared error (MSE) on both the function values and the derivatives, indicating that the network was able to capture not only the function's values but also its smoothness and curvature.

Notably, the inclusion of the derivative loss in the training objective played a significant role in encouraging the network to learn a more consistent shape, particularly near the boundaries of the domain where the Runge function exhibits sharper transitions. The derivative supervision served as a regularizing signal, reducing oscillations that typically occur in function-only fitting tasks.

The current loss formulation uses equal weights for function and derivative losses. In practice, tuning the trade-off between the two could yield better results depending on the specific application.

6 Conclusion

In this study, we implemented a simple neural network to approximate the Runge function $f(x) = \frac{1}{1+25x^2}$ along with its derivative $f'(x)$. By incorporating both function loss and derivative loss into the training process, the model achieved accurate predictions for both quantities, as verified through quantitative metrics (MSE and Max Error) and visualizations.

Overall, this experiment validates the effectiveness of using derivative supervision in function approximation tasks and paves the way for more structured learning in scientific and engineering applications.