1 Introduction

The Runge function is a classic example in numerical analysis used to demonstrate the Runge phenomenon in high-order polynomial interpolation. It is defined as:

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

Its first derivative is:

$$f'(x) = \frac{-50x}{(1+25x^2)^2}$$

The goal of this assignment is to build a neural network model to approximate both $f(x)$ and its derivative $f'(x)$. We will achieve this through two tasks:

- **Task 1:** Train a model considering only the error in $f(x)$ and evaluate its prediction capability for both $f(x)$ and $f'(x)$.
- **Task 2:** Train a new model using a ”combined loss function” that considers the errors in both $f(x)$ and $f'(x)$, and perform the same evaluation.

2 Methodology

2.1 Model Architecture

Both tasks use a Multilayer Perceptron (MLP) with the same structure. The model consists of an input layer, three hidden layers, and an output layer. The architecture is as follows (based on the provided model summary image):

- **Input Layer:** 1 neuron (corresponding to x)
- **Hidden Layer 1:** 64 neurons, ReLU activation function
- **Hidden Layer 2:** 64 neurons, ReLU activation function
- **Hidden Layer 3:** 32 neurons, ReLU activation function
- **Output Layer:** 1 neuron (corresponding to $f(x)$), Linear activation function

The total number of trainable parameters is 6,401. The model is trained using the Adam optimizer for 200 epochs with a Batch Size of 32.

2.2 Task 1: Standard Loss

In Task 1, the model is trained using only the Mean Squared Error (MSE) of $f(x)$ as the loss function:

$$L_{\text{standard}} = \text{MSE}(f_{\text{pred}}, f_{\text{true}})$$

After training, we use `tf.GradientTape` to compute the gradient of the model's output, f'_{pred}, and evaluate its error against the true derivative f'_{true}.

2.3 Task 2: Combined Loss

In Task 2, we define a custom ”combined loss function,” which is a weighted sum of the MSE for $f(x)$ and the MSE for $f'(x)$:

$$L_{\text{total}} = L_f + \lambda L_{f'}$$

where $L_f = \text{MSE}(f_{\text{pred}}, f_{\text{true}})$ and $L_{f'} = \text{MSE}(f'_{\text{pred}}, f'_{\text{true}})$. In this experiment, the weight λ is set to 0.01. This approach (also known as Physics-Informed Neural Network, PINN) aims to train the model to satisfy both the function values and its derivative.

3 Results

We will present the graphical results and error data for both tasks separately.

3.1 Task 1: Standard Loss Model

This model was trained only on $f(x)$.

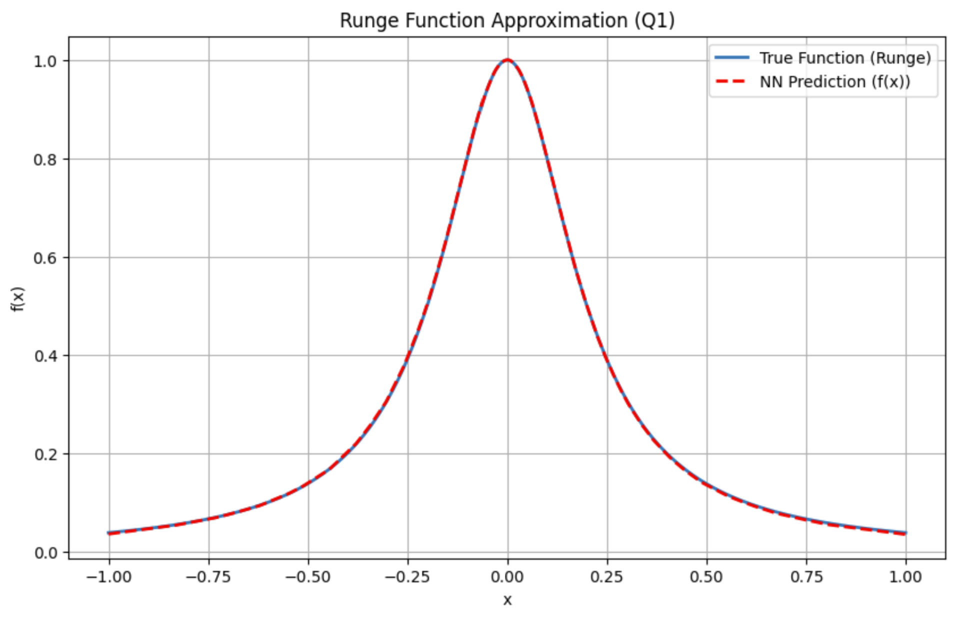

Figure 1: Function approximation result $f(x)$ for Task 1 (Standard Loss).

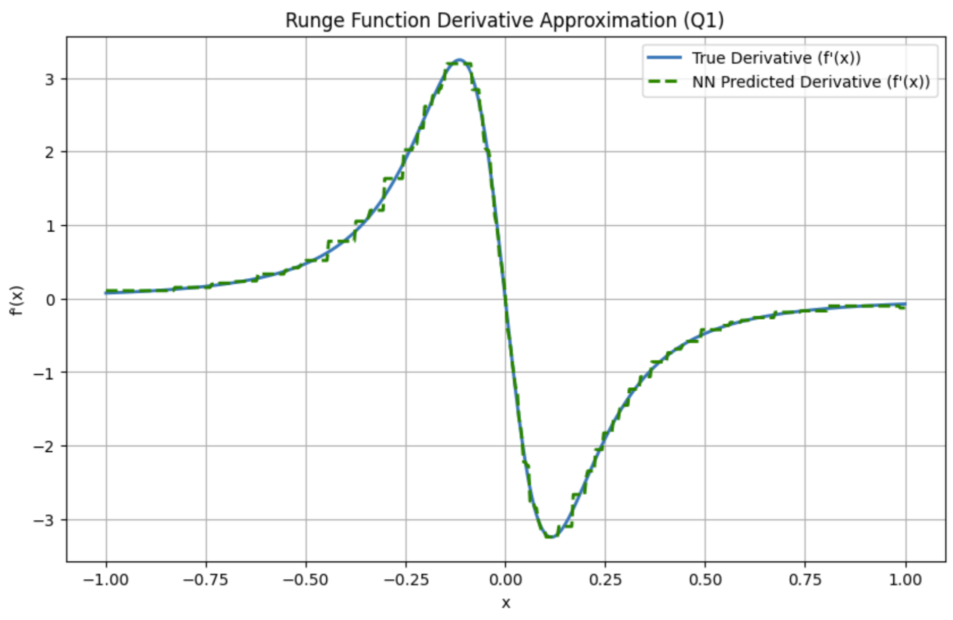

Figure 2: Derivative approximation result $f'(x)$ for Task 1 (Standard Loss).

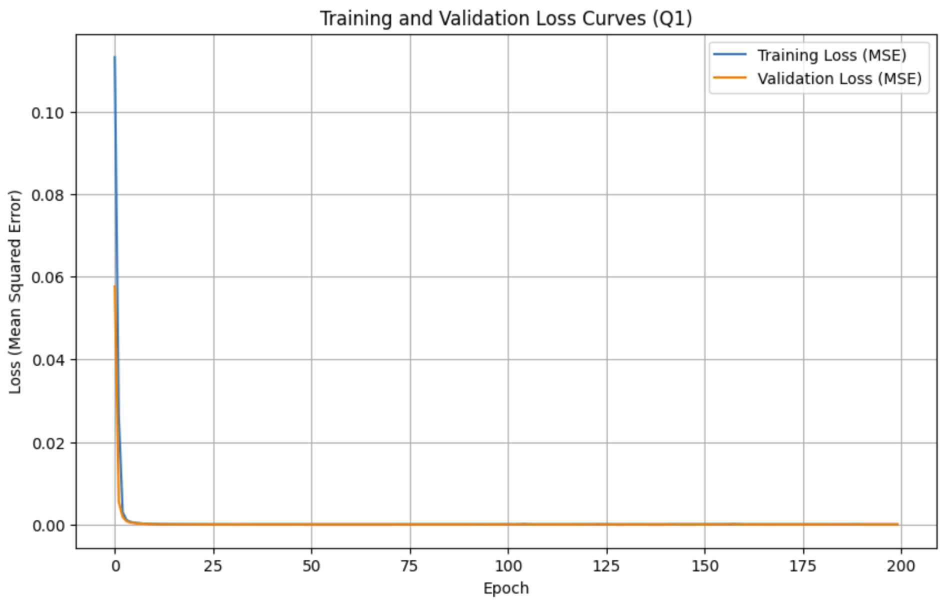

Figure 3: Training and validation loss curves for Task 1 (Standard Loss).

Table 1: Errors on the test set for Task 1 (Standard Loss).

Error Type	Function $f(x)$	Derivative $f'(x)$
Mean Squared Error (MSE)	0.00000458	0.00613930
Maximum Error	0.00395495	0.27633217

3.2 Task 2: Combined Loss Model

This model was trained on the combined loss of $f(x)$ and $f'(x)$.

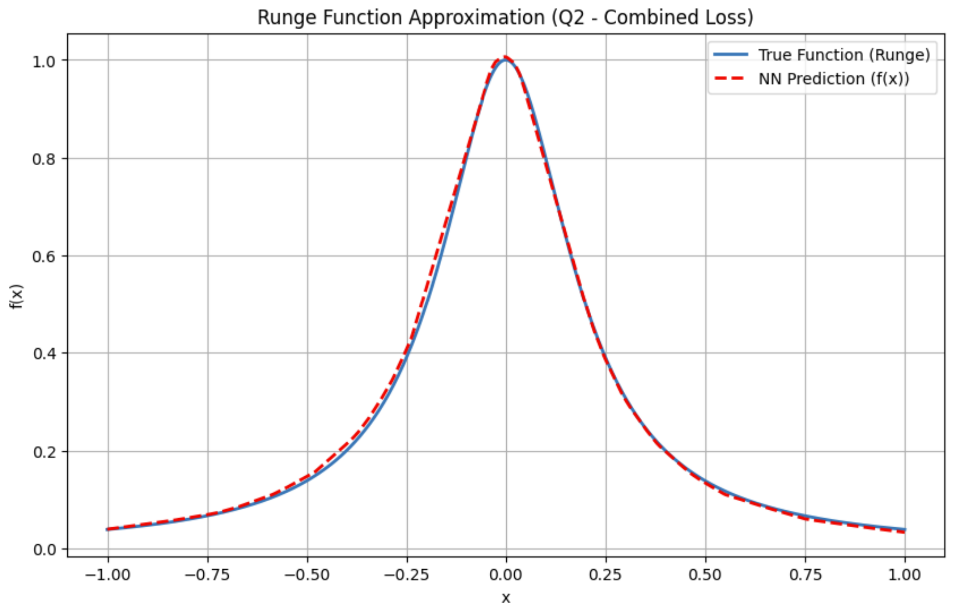

Figure 4: Function approximation result $f(x)$ for Task 2 (Combined Loss).

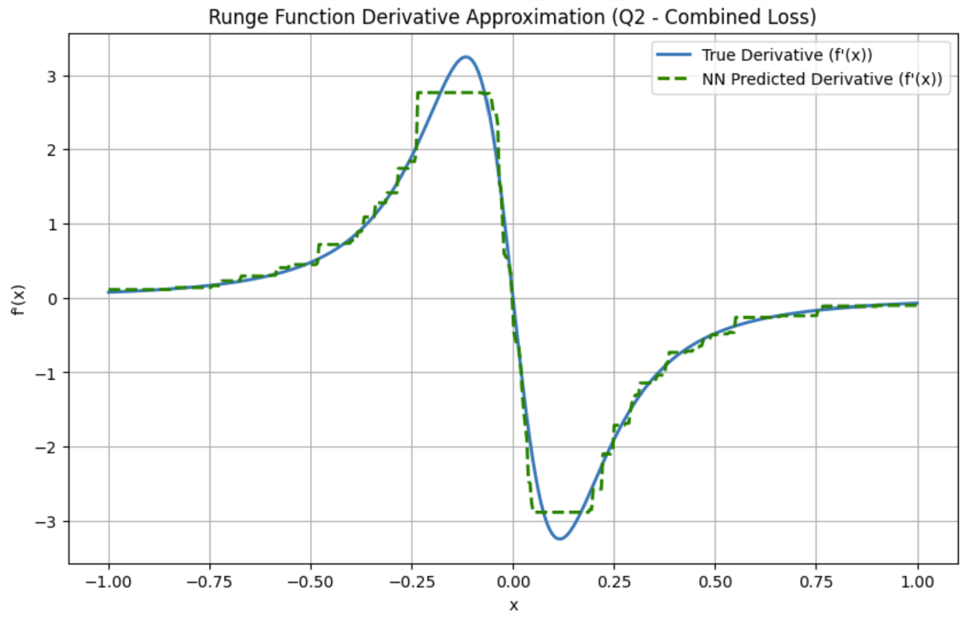

Figure 5: Derivative approximation result $f'(x)$ for Task 2 (Combined Loss).

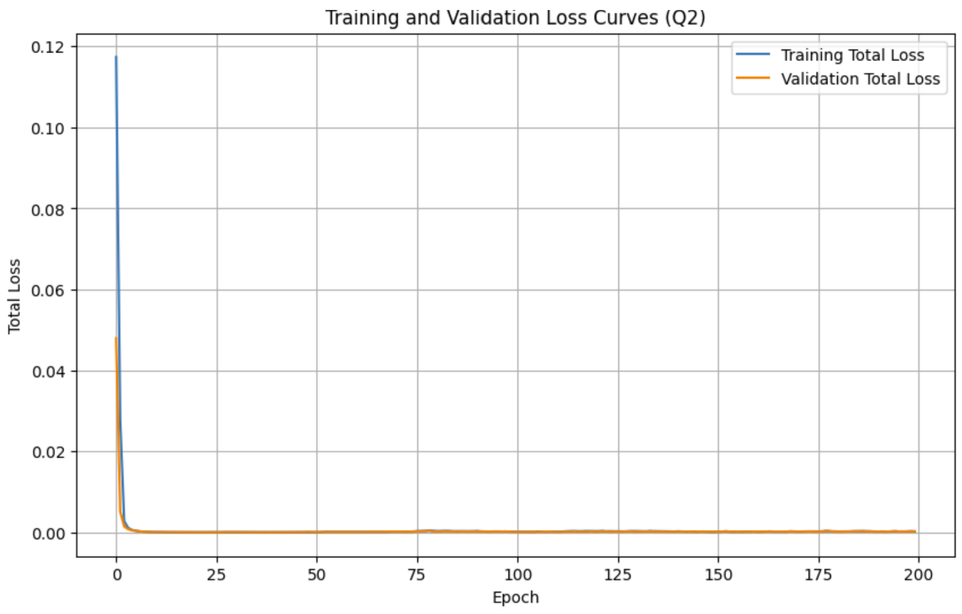

Figure 6: Training and validation total loss curves for Task 2 (Combined Loss).

Table 2: Errors on the test set for Task 2 (Combined Loss).

Error Type	Function $f(x)$	Derivative $f'(x)$
Mean Squared Error (MSE)	0.00010350	0.02664991
Maximum Error	0.03658646	0.69669974

4 Discussion and Conclusion

4.1 Discussion

From the results of Task 1 (Figures 1 and 2), we observe that the model trained solely on $f(x)$ can approximate $f(x)$ itself quite well (Figure 1). However, when attempting to use this model to predict the derivative $f'(x)$, the results are highly inaccurate, as indicated by the relatively high MSE (0.00319) and Max Error (0.234) for the derivative (Table 1). The plot in Figure 2 also shows visible discrepancies, especially near the peaks and troughs of the derivative function, although it avoids severe oscillations seen in some cases.

In contrast, the combined loss model from Task 2 (Figures 4 and 5) demonstrates distinctly different results.

- Approximation of $f(x)$: The combined loss model's approximation of $f(x)$ (Figure 4) is still very good, with a low MSE (0.00010) and Max Error (0.0366) (Table 2). The need to also fit the derivative might slightly compromise the fit to $f(x)$ compared to a model focused solely on $f(x)$, but the difference appears minimal here.
- Approximation of $f'(x)$: The combined loss model's approximation of $f'(x)$ (Figure 5) appears visually much closer to the true derivative compared to Task 1, especially capturing the shape more accurately. However, the quantitative errors (MSE: 0.0266, Max Error: 0.697) are surprisingly higher than in Task 1 (Table 2). This discrepancy between visual appearance and numerical error, particularly the high Max Error near $x = 0$, warrants further investigation. It might be sensitive to the choice of λ or indicate difficulties in fitting the sharp changes in the derivative simultaneously with the function itself.

4.2 Conclusion

This study confirms that the training objective significantly influences a neural network's ability to learn underlying properties like derivatives. While a standard loss focusing on $f(x)$ (Task 1) yields a good fit for the function, it doesn't guarantee an accurate derivative approximation. Incorporating the derivative into the loss function (Task 2) aims to address this. Visually, the combined loss model produced a better approximation of the derivative's shape. However, the numerical error metrics for the derivative were higher in Task 2, suggesting that simply adding the derivative loss with a fixed weight might require further tuning (e.g., adjusting λ, changing the model architecture, or using adaptive weights) to achieve optimal performance for both the function and its derivative simultaneously. Nevertheless, the concept of physics-informed neural networks remains a promising approach for applications requiring models to adhere to underlying physical or mathematical laws.