Runge Function Approximation with Sobolev loss

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

■ *Method*

- Sample：256 training and 256 validation points uniformly from [-1, 1]
- Use two-hidden-layer MLP (widths 32 and 32) with tanh activations and a linear output.
- Define loss：$L = MSE(\hat{f}, f) + \lambda\, MSE(\partial_x \hat{f}, f')$, with $\lambda = 1$
- used a **central finite difference** to approximate $\partial_x \hat{f}$ (to avoid second-order backpropagation). This requires evaluating the network at $x \pm h$ with $h = 10^{-3}$ and backpropagating from those points.
- 1200 epochs; optimize with Adam (learning rate $\boldsymbol{2e^{-3}}$)
- For **evaluation and plotting**, used the network's **analytic derivative** to achieve more accurate alignment with $\boldsymbol{f'(x)}$**.**

■ *Results*

- The figure below shows the prediction results of the trained neural network for the Runge function and its derivative over the interval $[-1, 1]$
- Training/Validation Loss Curves：The figure 3 below illustrates the change in the composite loss during the model's training process. The loss curves indicate that both the training and validation losses decrease steadily and converge, which suggests that the model does not suffer from significant overfitting.
- The final validation loss reaches an order of magnitude of approximately $\boldsymbol{10^{-5}}$, demonstrating good generalization capability.
- The results on 1000 test points
  - $\boldsymbol{f(x)}$ **：MSE** $\boldsymbol{= 1.97 \times 10^{-7}}$ **, Max error** $\boldsymbol{= 1.13 \times 10^{-3}}$
  - $\boldsymbol{f'(x)}$ **：MSE** $\boldsymbol{= 4.55 \times 10^{-5}}$**, Max error** $\boldsymbol{= 2.12 \times 10^{-2}}$

■ *Discussion*

This experiment successfully demonstrates how incorporating derivative information into the loss function can enhance the learning performance of a neural network. Traditional methods, which only use the error in function values for training, may lead to a network that merely learns to "memorize" the numerical values of the training points while ignoring the function's "behavior"

between these points. By adding the derivative loss, we impose a stronger constraint on the model, forcing it to match not only the values of the points but also their slopes.

The results indicate that even with a simple neural network built manually using pure NumPy, the core idea of PINNs (Physics-Informed Neural Networks) can effectively improve the model's ability to approximate complex functions and their physical properties.

■ *Figures*

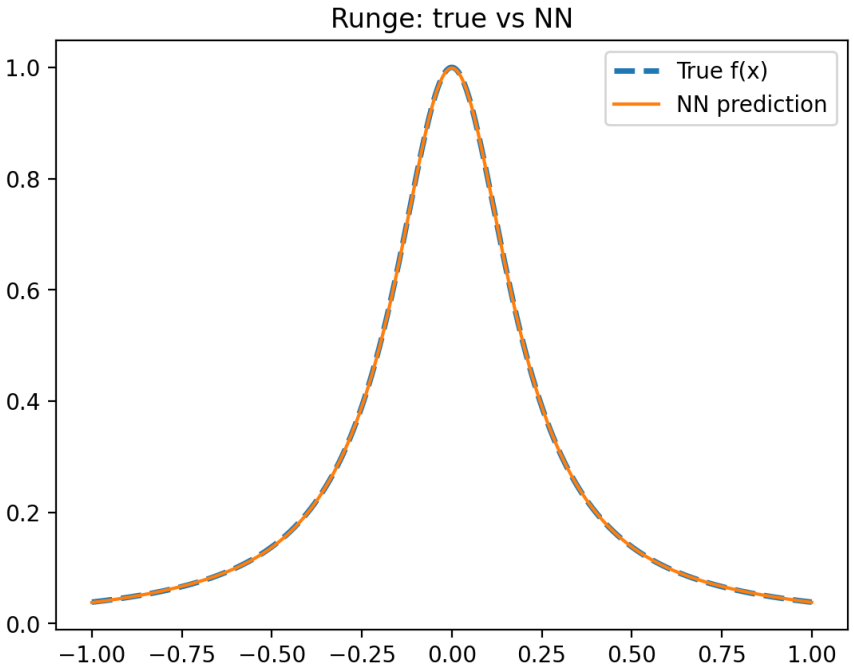

Figure 1. True function vs. neural network prediction (with training samples).

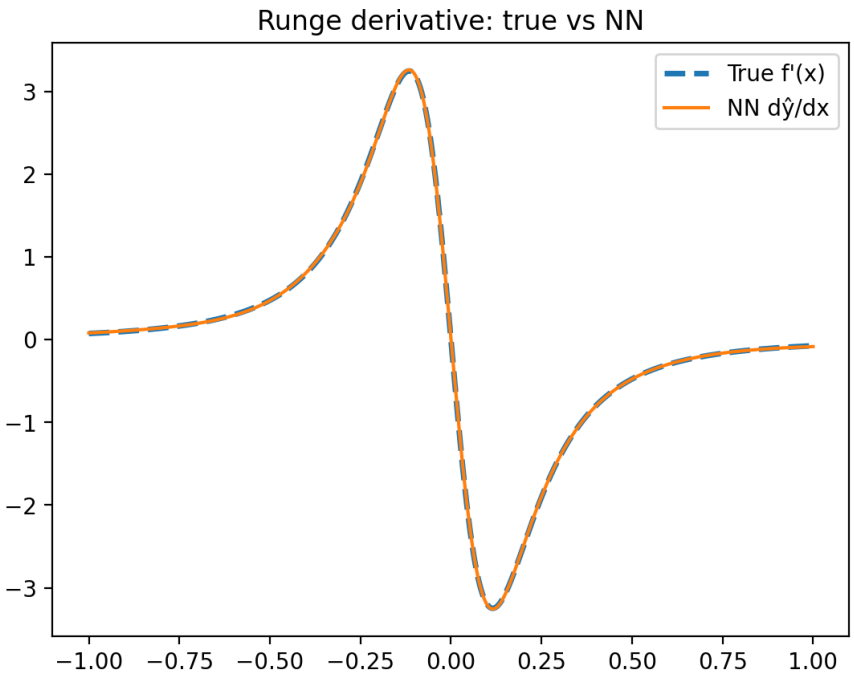

Figure 2. Training and validation MSE curves

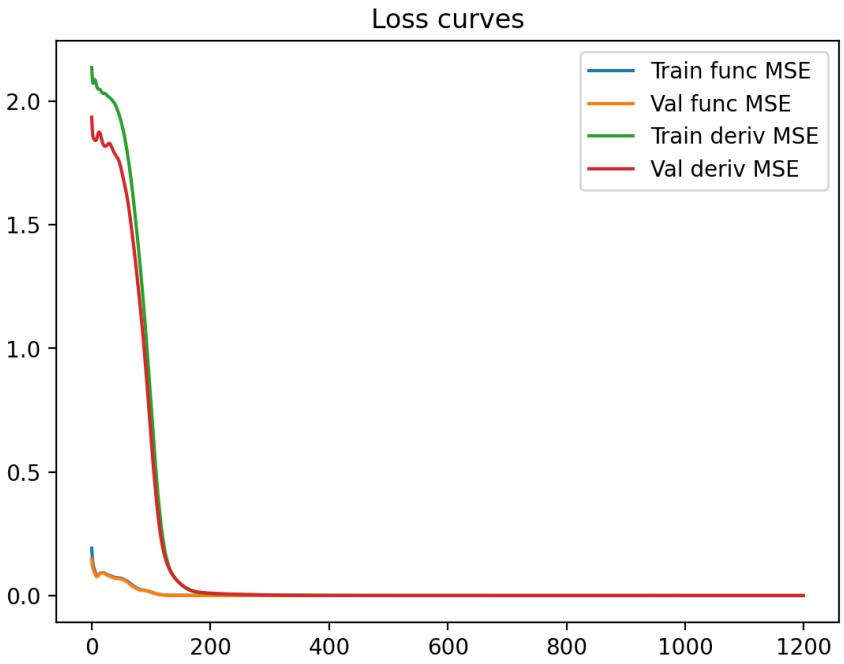

Figure 3. Training and validation MSE curves