Approximating the Runge Function with a Small Neural Network

- Target

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

- Method

Model One–hidden-layer MLP (width H=64) with a leaky-ReLU hidden activation and linear output.

Training objective Mean–squared error (MSE) between prediction $\hat{f}(x)$ and $f(x)$.

Data 256 training points on a uniform grid in [−1,1]; validation uses 256 **midpoints** to avoid overlap.

Optimization Full-batch gradient descent, learning rate 0.010.010.01 with step decays; 3000 epochs.

Implementation Pure-Python (no external libraries), forward & back-prop coded by hand.

- Loss definitions

Function loss (MSE)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{f}(x_i) - f(x_i)\right)^2$$

Max error (on a dense grid)

$$\max_i \left|\hat{f}(x_i) - f(x_i)\right|$$

- Results

Learning curves (numbers from your logs)

Epoch	Train MSE	Val MSE
200	0.017319	0.017174
1000	0.010072	0.010022
2000	0.003475	0.003463
3000	**0.001768**	**0.001763**

Both curves decrease steadily and remain almost identical → no overfitting.

Final evaluation (dense grid of 1000 points)

MSE: 0.00176431

Max error: 0.07533909 at x≈0.0771

These errors are small relative to the function scale [0.0385,1], showing a solid fit.

Qualitative behavior

The network captures the central peak near x=0 and the flat tails toward ±1.

The largest deviation occurs around a moderate x (here x≈0.077), which matches where the curvature changes fastest.

- Discussion

What helped

Using midpoints for validation avoids train/val leakage on a grid.

A simple learning-rate decay after plateaus.

Even a single hidden layer is sufficient for this 1D target.

Limitations / possible improvements

Increase width (e.g., H=128) or train longer to push MSE lower.

Use Chebyshev-biased sampling (denser near ±1) to reduce edge errors for Runge-type shapes.

A smooth activation (tanh-like) can further stabilize training if you later add a derivative term to the loss.