Machine Learning

313704071 陳安定 Assignment 3

Lemma 3.1. Let $k \in \mathbb{N}_0$ *and* $s \in 2\mathbb{N}-1$

$$\max_{\substack{p \le s \\ p\ odd}} \left\| f_p - (\Psi_{s,\epsilon})_{\frac{p+1}{2}} \right\|_{W^{k,\infty}} \le \epsilon$$

This lemma shows how to approximate all odd degree monomials y^p with tanh neural network, and the error from the actual value is approached to an arbitrarily small error ϵ in $[-M,M]$ and under norm $W^{k,\infty}$.

The author first define the pth order central finite difference (中央有限差分) δ_h^p, which is used to exlicit the pth derivatives of tanh. Let $p \le s$.

$$\delta_h^p[f](x) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} f\left(x + \left(\frac{p}{2} - i\right) h\right)$$

Use $\hat{f}_{q,h}(y)$ as an approximator, the purpose is to approach y^p, for any *p* is odd and lower than *s*. The proof will be described step by step.

$$\hat{f}_{q,h}(y) := \frac{\delta_{hy}^q[\sigma](0)}{\sigma^{(q)}(0)h^q}$$

With $0 \le m \le \min\{k, p+1\}$, by chain rule, the mth dirivitive of $\delta_{hx}^q[\sigma](0)$ is:

$$\frac{d^m}{dx^m}\delta_{hx}^p[\sigma](0) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} \left(\frac{p}{2} - i\right)^m h^m \cdot \sigma^{(m)}\left(\left(\frac{p}{2} - i\right) hx\right)$$

Then, do Taylor expansion (for part of $\sigma^{(m)}$), and the main body is the finite sum from $l = m\ to\ p+1$ plus a remainder containing $\sigma^{(p+2)}(\xi_{x,i})$.

$$\frac{d^m}{dx^m}\delta_{hx}^p[\sigma](0) = \sum_{i=0}^{p} (-1)^i \binom{p}{i} \left(\frac{p}{2} - i\right)^m h^m \left(\sum_{l=m}^{p+1} \frac{\sigma^{(l)}(0)}{(l-m)!} \left(\frac{p}{2} - i\right)^{l-m} (hx)^{l-m} \right)$$

$$+ \sum_{i=0}^{p} (-1)^i \binom{p}{i} \left(\frac{p}{2} - i\right)^m h^m \left(\frac{\sigma^{(p+2)}(\xi_{x,i})}{(p+2-m)!} \left(\frac{p}{2} - i\right)^{p+2-m} (hx)^{p+2-m} \right)$$

By key Identity (Katsuura 2009):

$$\sum_{i=0}^{p}(-1)^i\binom{p}{i}\left(\frac{p}{2}-i\right)^l = p!\,\delta(1-p) = \begin{cases} p!\,,(\boldsymbol{l}=\boldsymbol{p}), \\ 0, \quad (l \neq p). \end{cases}$$

We can do some technical algebraic operations and rewrite blue part

$$\sum_{i=0}^{p}(-1)^i\binom{p}{i}\left(\frac{p}{2}-i\right)^m h^m\left(\sum_{l=m}^{p+1}\frac{\sigma^{(l)}(0)}{(l-m)!}\left(\frac{p}{2}-i\right)^{l-m}(hx)^{l-m}\right)$$

$$= h^m\sum_{l=m}^{p+1}\frac{\sigma^{(l)}(0)}{(l-m)!}(hx)^{l-m}p!$$

$$= h^m\frac{\sigma^{(p)}(0)}{(p-m)!}(hx)^{p-m}p!$$

$$= h^p\sigma^{(p)}(0)\boldsymbol{f}_{\boldsymbol{p}}^{(\boldsymbol{m})}(\boldsymbol{x}), \qquad (0 \leq m \leq p)$$

Now we know $\boldsymbol{f}_{\boldsymbol{p}}^{(\boldsymbol{m})}(\boldsymbol{x})$.

Then, we calculate $\hat{f}_{p,h}^{(m)}(x) - \boldsymbol{f}_{\boldsymbol{p}}^{(\boldsymbol{m})}(\boldsymbol{x})$

$$= \frac{1}{h^p\sigma^{(p)}(0)}(\textit{red part})$$

$$= \sum_{i=0}^{p}(-1)^i\binom{p}{i}\frac{1}{(p+2-m)!}\frac{\sigma^{(p+2)}(\xi_{x,i})}{\sigma^{(p)}(0)}\left(\frac{p}{2}-i\right)^{p+2}h^2x^{p+2-m}$$

With the lower and upper bound on the derivatives of σ from Lemma A.1 and A.4, which yield for $m \leq \min(k, p+1)$, and we should only know that error will be bound by an arbitrary number $(k \in \mathbb{N})$. And $h > 0$, such that for all $p < s$.

$$\|f_p - \hat{f}_{p,h}\|_{W^{m,\infty}} \leq ((2(p+2)pM)^{p+3} + (2ph)^{k+1}))h^2 =: \boldsymbol{\epsilon}$$

The shallow tanh neural network $\Psi_{s,\epsilon}$, by $\left(\Psi_{s,\epsilon}\right)_p = \hat{f}_{p,h}$ such that it only has $\frac{s+1}{2}$ neurons in hidden layer. The width follows directly from its definition and the fact that σ is an odd function.

Lemma 3.2.

In Lemma 3.1., we made a monomial approximator with odd degree $\hat{f}_{p,h}$, and control $W^{k,\infty}$ error. The even power y^{2n} is not derived directly from difference but is expressed as **"The difference of two odd powers minus the linear combination of lower-order even powers"** using an algebraic equation. In this way, the higher-order even power can be constructed. To any $n \in \mathbb{N}, \alpha > 0$, it holds that

$$y^{2n} = \frac{1}{2\alpha(2n+1)}\left((y+\alpha)^{2n+1} - (y-\alpha)^{2n+1} - 2\sum_{k=0}^{n-1}\binom{2n+1}{2k}\alpha^{2(n-k)+1}y^{2k}\right)$$

Define $(\Psi_{s,\epsilon}(y))_{2n}$ as approximator of y^{2n}, and when $p = odd$, we can use Lemma3.1., so the above formula can be rewritten as:

$$(\Psi_{s,\epsilon}(y))_{2n} = \frac{1}{2\alpha(2n+1)}\Bigg(\hat{f}_{2n+1,h}(y+\alpha) - \hat{f}_{2n+1,h}(y-\alpha)$$

$$- 2\sum_{k=0}^{n-1}\binom{2n+1}{2k}\alpha^{2(n-k)+1}(\Psi_{s,\epsilon}(y))_{2k}\Bigg)$$

Introducing the notation $E_p = \left\|f_p - \left(\Psi_{s,\epsilon}\right)_p\right\|_{W^{m,\infty}}$, then

$$E_p \leq E_p^* := \frac{2^{p/2}(1+\alpha)^{(p^2+p)/2}}{\alpha^{p/2}} \cdot \epsilon$$

Note that choosing h as in Lemma 3.1. implies that $\max\limits_{\substack{p\leq s \\ p\ odd}} E_p \leq \epsilon$

The author uses induction steps to obtain that $E_{2n} \leq E_{2n}^*$ for all $p \leq s$.

By Lemma A.2, it proves that the optimal choice of α is $1/s$. It concludes that for any $\epsilon \geq 0$, there exist a shallow tanh neural network $\Psi_{s,\epsilon}$, such that

$$\max_{p\leq s}\left\|f_p - \left(\Psi_{s,\epsilon}\right)_p\right\|_{W^{k,\infty}} \leq \epsilon$$

Since it needs to use three inputs $(y-\alpha), y, (y+\alpha)$, the hidden layer width is $\frac{3(s+1)}{2}$ (three per position). The entire structure is still a single hidden layer, but it can compress all $0 \leq p < s$ monomials to the specified accuracy at once.

Programming assignment

Review: Approximation of Runge function

Data:

In previous assignment, the sample data (training points) are distributed using **Chebyshev samples**. This sampling method will make the data points denser at the end points (x≈±1) and sparser in the middle (x≈0). It can reduce the error caused by Runge phenomenon (endpoint oscillation). However, since this method is ChatGPT teaches us, I skipped trying to use uniform-sampling. I found that because we use MSE regression training, the model is not forced to "step through" the training samples. Therefore, we don't need to use Chebyshev samples.

Hypothesis:

# of hidden layer: **2**

# of neurons in each hidden layer: **64**

Activation function: **tanh**

Two hidden layers can express the required curvature and nonlinear relationship, and about number of neurons, the scale (1-64-64-1) has achieved low RMSE and good L_∞ performance in a one-dimensional smooth approximation. As for the activation function, tanh is a smooth and oddly symmetric nonlinearity that can approximate the curvature with fewer nodes.

Loss function: MSE

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$$

The reason why use MSE is that the model output is smooth, differentiable everywhere, and has simple gradients. Also, it is convenient for back propagation and learning rate setting. By the way, we report RMSE and L_∞ to intuitively interpret the average error size and avoid ignoring large local errors.

1. ***Use the same code to calculate the error in approximating the derivative of the given function.***

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

Also use MSE to evaluate error between $f'(x)$ and $\hat{f}'(x)$, using PyTorch automatic differentiation to find the derivative of the model output with respect to the input.

```
dhat_dx = torch.autograd.grad(yhat.sum(), xs, create_graph=False)[0]
```

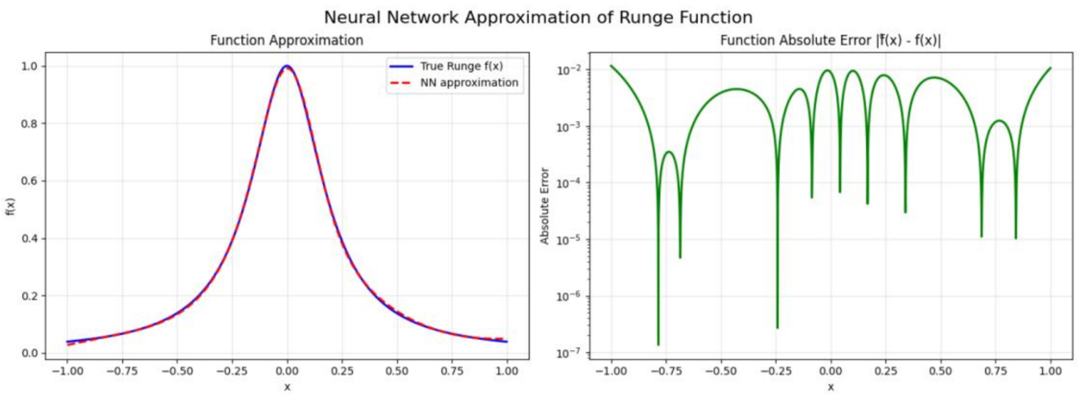
yhat.sum() is a n by 1 matrix, **xs** are independent variables, and **autograd.grad** means automatically differentiating each sample.

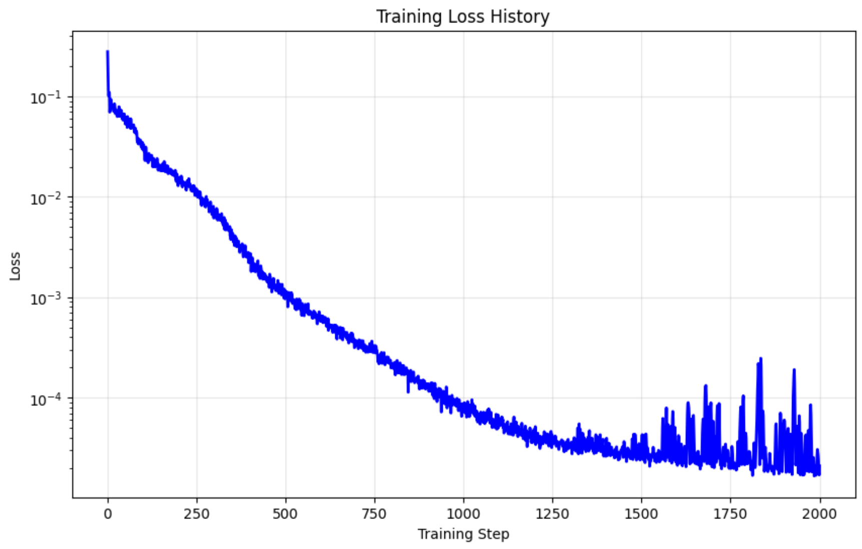
```
Mean absolute error: 5.596297e-02
Max absolute error: 2.636853e-01
Min absolute error: 1.087785e-05
Std of absolute error: 5.460906e-02
Median absolute error: 3.865539e-02
```

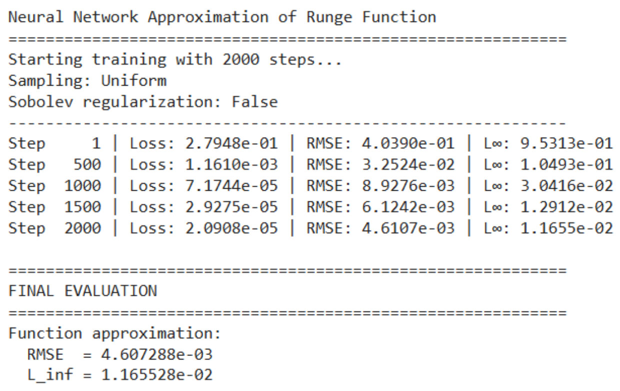
2. Use a neural network to approximate both the Runge function and its derivative.

Loss function of the function and the derivative function: both are **MSE**.

(1) Function



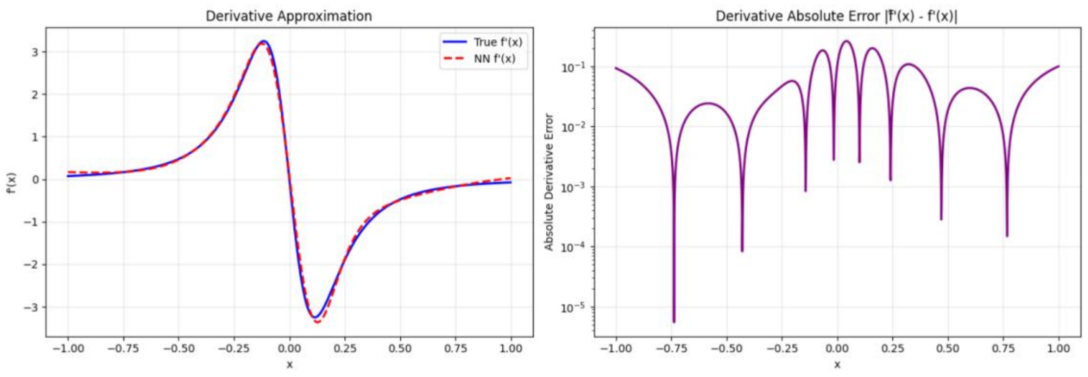


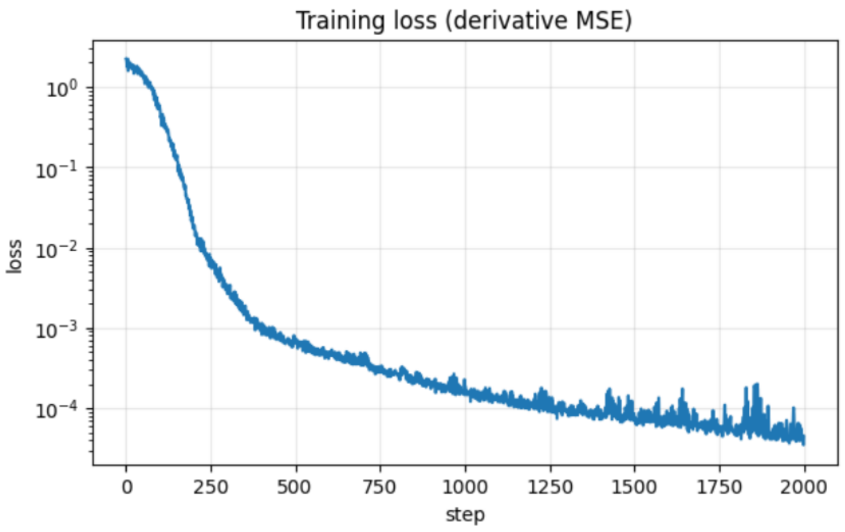


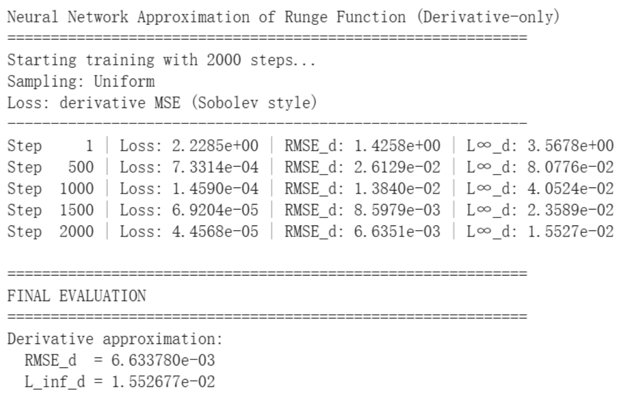
```
Neural Network Approximation of Runge Function
============================================================
Starting training with 2000 steps...
Sampling: Uniform
Sobolev regularization: False
------------------------------------------------------------
Step     1 | Loss: 2.7948e-01 | RMSE: 4.0390e-01 | L∞: 9.5313e-01
Step   500 | Loss: 1.1610e-03 | RMSE: 3.2524e-02 | L∞: 1.0493e-01
Step  1000 | Loss: 7.1744e-05 | RMSE: 8.9276e-03 | L∞: 3.0416e-02
Step  1500 | Loss: 2.9275e-05 | RMSE: 6.1242e-03 | L∞: 1.2912e-02
Step  2000 | Loss: 2.0908e-05 | RMSE: 4.6107e-03 | L∞: 1.1655e-02

============================================================
FINAL EVALUATION
============================================================
Function approximation:
  RMSE  = 4.607288e-03
  L_inf = 1.165528e-02
```

(2) Derivative function







```
Neural Network Approximation of Runge Function (Derivative-only)
============================================================
Starting training with 2000 steps...
Sampling: Uniform
Loss: derivative MSE (Sobolev style)
------------------------------------------------------------
Step     1 | Loss: 2.2285e+00 | RMSE_d: 1.4258e+00 | L∞_d: 3.5678e+00
Step   500 | Loss: 7.3314e-04 | RMSE_d: 2.6129e-02 | L∞_d: 8.0776e-02
Step  1000 | Loss: 1.4590e-04 | RMSE_d: 1.3840e-02 | L∞_d: 4.0524e-02
Step  1500 | Loss: 6.9204e-05 | RMSE_d: 8.5979e-03 | L∞_d: 2.3589e-02
Step  2000 | Loss: 4.4568e-05 | RMSE_d: 6.6351e-03 | L∞_d: 1.5527e-02

============================================================
FINAL EVALUATION
============================================================
Derivative approximation:
  RMSE_d  = 6.633780e-03
  L_inf_d = 1.552677e-02
```