ML HW2

Date: 09/10/2025
Week: 2
Author: Alvin B. Lin
Student ID: 112652040

Problem 1: Algorithm Construction

1. Read Deep Learning: An Introduction for Applied Mathematicians. Consider a network as defined in (3.1) and (3.2).

$$a^{[1]} = x \in \mathbb{R}^{n_1}, \tag{3.1}$$

$$a^{[l]} = \sigma\left(W^{[l]}a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \qquad \text{for } l = 2,\ 3, \dots,\ L. \tag{3.2}$$

Assume that $n_L = 1$, find an algorithm to calculate $\nabla a^{[L]}(x)$.

Sol: Recall that we have a powerful tool ——**Chain Rule**.

Theorem 1: Recurrence Formula on δ Function

$$\delta^{[k]} = \frac{\partial C}{\partial z^{[k]}} = \frac{\partial C}{\partial z^{[k+1]}} \frac{\partial z^{[k+1]}}{\partial z^{[k]}} = \delta^{[k+1]} \frac{\partial z^{[k+1]}}{\partial z^{[k]}} = \delta^{[l]} \frac{\partial z^{[l]}}{\partial z^{[l-1]}} \cdots \frac{\partial z^{[k+1]}}{\partial z^{[k]}}$$

where $2 \le k < l \le L$.

Since $n_L = 1$, $a^{[L]} \in \mathbb{R}$. Hence we can replace the role of C by $a^{[L]}$. We get:

$$\xi^{[k]} := \frac{\partial a^{[L]}}{\partial z^{[k]}} = \frac{\partial a^{[L]}}{\partial z^{[k+1]}} \frac{\partial z^{[k+1]}}{\partial z^{[k]}} = \xi^{[k+1]} \frac{\partial z^{[k+1]}}{\partial z^{[k]}}, \quad 2 \le k < L; \qquad \xi^{[L]} := \sigma'(z^{[L]}),$$

where $2 \le k < L$. Note that

$$\frac{\partial z^{[k+1]}}{\partial z^{[k]}} = \frac{\partial z^{[k+1]}}{\partial a^{[k]}} \frac{\partial a^{[k]}}{\partial z^{[k]}} = W^{[k+1]} \sigma'(z^{[k]}), \quad 2 \le k < L;$$

therefore

$$\xi^{[k]} = \sigma'(z^{[k]}) \circ \left[(W^{[k+1]})^\top \xi^{[k+1]}\right], \quad 2 \le k < L.$$

Lastly, $\dfrac{\partial z^{[2]}}{\partial x} = W^{[2]}$, the last puzzle of the algorithm is finally done.

The algorithm is as follow:

Step 1: Calculate $\xi^{[L]} = \sigma'(z^{[L]})$

Step 2: Use recurrence formula:

$$\xi^{[k]} = \sigma'(z^{[k]}) \circ \left[(W^{[k+1]})^\top \xi^{[k+1]}\right], \quad 2 \le k < L,$$

to compute $\xi^{[2]}$.

Step 3: $\nabla a^{[L]} = \xi^{[2]} \dfrac{\partial z^{[2]}}{\partial x} = (W^{[2]})^\top \xi^{[2]}$. □

Problem 3: Runge Phenomena

1. Use a neural network to approximate the Runge function

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

Write a short report (1–2 pages) explaining method, results, and discussion including

- Plot the true function and the neural network prediction together.
- Show the training/validation loss curves.
- Compute and report errors (MSE or max error).

Sol: The full `Jupyter` file is here.

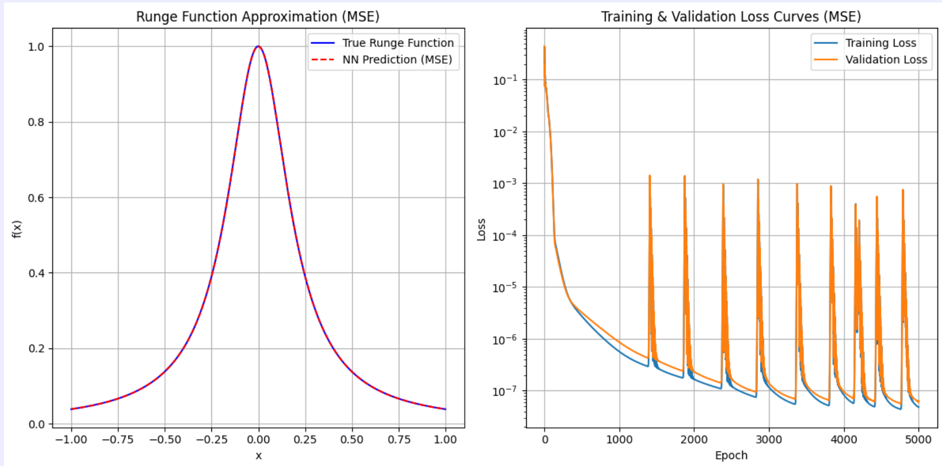
- **Basic Setup**
  1. The training set and validation/training set are sampled 100 and 20 uniformly in the interval $[-1,\ 1]$ respectively.
  2. The neural network has 2 layers and each has 50 neurons.
  3. Activation function is chosen as $\tanh x$.
  4. Two optimisers are tested for the comparison:
     - Regular **MSE**
     - **Sup-norm** (also **p-norm** for $p = 20$)
  5. Epoch is set 5000.
- **Result Presentation**
  1. Loss of **MSE**.

```
--- Training with MSE Loss ---
Epoch [500/5000], Train Loss: 0.000002, Val Loss: 0.000002
Epoch [1000/5000], Train Loss: 0.000000, Val Loss: 0.000001
Epoch [1500/5000], Train Loss: 0.001174, Val Loss: 0.000867
Epoch [2000/5000], Train Loss: 0.000000, Val Loss: 0.000000
Epoch [2500/5000], Train Loss: 0.000000, Val Loss: 0.000000
Epoch [3000/5000], Train Loss: 0.000000, Val Loss: 0.000000
Epoch [3500/5000], Train Loss: 0.000000, Val Loss: 0.000000
Epoch [4000/5000], Train Loss: 0.000000, Val Loss: 0.000000
Epoch [4500/5000], Train Loss: 0.000000, Val Loss: 0.000000
Epoch [5000/5000], Train Loss: 0.000000, Val Loss: 0.000001
```

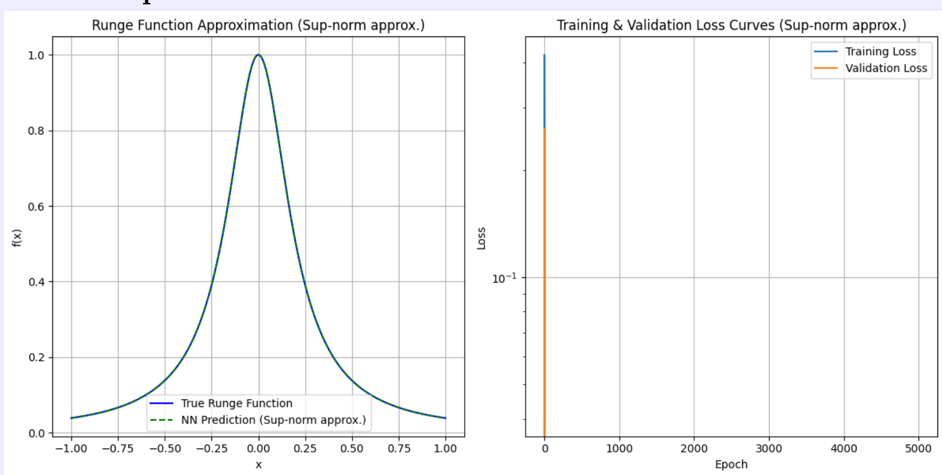
  2. Loss of **sup-norm**.

```
--- Training with Sup-norm Approximation Loss (Corrected) ---
Epoch [500/5000], Train Loss: -2.728260, Val Loss: -3.100614
Epoch [1000/5000], Train Loss: -3.279954, Val Loss: -3.616254
Epoch [1500/5000], Train Loss: -3.972852, Val Loss: -4.871111
Epoch [2000/5000], Train Loss: -4.939495, Val Loss: -5.529874
Epoch [2500/5000], Train Loss: -5.252147, Val Loss: -5.487857
Epoch [3000/5000], Train Loss: -5.954085, Val Loss: -6.249280
Epoch [3500/5000], Train Loss: -6.223094, Val Loss: -6.334863
Epoch [4000/5000], Train Loss: -7.161107, Val Loss: -7.228351
Epoch [4500/5000], Train Loss: -7.462693, Val Loss: -7.674273
Epoch [5000/5000], Train Loss: -7.557062, Val Loss: -7.776904
```

- Approximation Result
  1. Result of **MSE**:



  2. Result of **sup-norm**:



- **Validation Loss & Conclusion**

```
--- Final Results Comparison ---
-----------------------------------------------------------------
         Metric          |    MSE Method    | Sup-Norm Method
-----------------------------------------------------------------
Final MSE                | 0.000001         |  0.000000
Final Max Abs Error      | 0.002417         |  0.000431
-----------------------------------------------------------------

Conclusion: The Sup-Norm method achieved a lower maximum absolute
error, making it the better method for this specific objective.
```

- **Explanation**
  Although the loss of the **sup-norm** method may seem high relative to the **MSE** method, the actual loss under the same metric has the opposite result. Not only the runtime but the final loss has shown that **sup-norm** is overall the better option of the two.

My Question 1: Why Is The Approximation Work During The Course?

For the approximation theory mentioned during the class, we know that for continuous function $f(x)$ on a closed interval $[a,\ b]$, the following must hold:

$$\forall \varepsilon \in \mathbb{R}^+,\ \exists n \in \mathbb{Z}^+,\ p(x) \in \mathbb{P}_n,\ s.t.\ \|f(x) - p(x)\|_{\mathcal{C}^0([a,\ b])} < \frac{\varepsilon}{2}. \tag{1}$$

And the professor said we aim to use machine learning method to use $\tanh x$ to approximate $p(x)$ such that

$$\forall \varepsilon \in \mathbb{R}^+, \exists N \in \mathbb{Z}^+,\ h(x) = \sum_{k=1}^{N} a_k \sigma(khx) + b,\ s.t.\ \|h(x) - p(x)\|_{\mathcal{C}^0([a,\ b])} < \frac{\varepsilon}{2} \tag{2}$$

Thus combine (1) and (2), we get:

$$\forall \varepsilon \in \mathbb{R}^+, \exists N \in \mathbb{Z}^+,\ h(x) = \sum_{k=1}^{N} a_k \sigma(khx) + b,\ s.t.\ \|h(x) - f(x)\|_{\mathcal{C}^0([a,\ b])} < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon \tag{3}$$

However the process seems legit, I find it a flaw occurs in the equation (2).
The function $p(x)$ in equation (1) is actually $p(x;\varepsilon)$, thus in equation (2), $h(x)$ is $h(x;\varepsilon)$. Meaning that $\forall \varepsilon \in \mathbb{R}^+$, we have a varying, unique objective polynomial $p(x;\varepsilon)$ to approximate, thus the structure of each $h(x;\varepsilon)$ may look very different. It is not just a simple step size h changing process, isn't it?

The Answer Got from AI for Problem 1

You are correct: the polynomial $p(x)$ and the approximating neural network are not fixed. They change depending on the value of ε you choose.

Your observation is exactly right. If you want a more accurate approximation (you choose a smaller ε), the Weierstrass theorem might require a different, possibly higher-degree, polynomial $p'(x)$; then, the equation would require a different, likely larger, neural network to approximate that new polynomial $p'(x)$.

The theorem's power lies in this **guarantee of existence**. It states that for any continuous function $f(x)$ you give me and for any desired level of accuracy ε you specify, I can find a neural network (of sufficient size) that will do the job.

The proof doesn't tell you **how to find that specific polynomial** or that specific network. That's what the training process in machine learning is for. The theorem simply provides the **theoretical foundation**, assuring us that a sufficiently large neural network has the capacity to represent the function we want to learn.

——— (Answer from `Gemini`)

My Question 2: Can An Activation Function Be Linear or Identity Function?

Is a non-linear activation function an absolute necessity for a neural network to be a useful model, and what happens if all activation functions are linear? (This question is refined by `Gemini`.)

The Answer Got from AI for Problem 2

Yes, a non-linear activation function is **absolutely necessary** for a neural network to be a useful and powerful tool for solving complex problems. Without it, the network **cannot learn anything** more complex than a simple **linear** relationship, regardless of how many layers it has.

The Role of Non-Linearity

- The entire purpose of a deep neural network is to learn complex, non-linear patterns in data—the kind of patterns found in images, speech, and text. These patterns cannot be captured by a straight line or a simple plane. Activation functions are the key to introducing this essential non-linearity.

What Happens with Linear Activation Functions?

- If you were to use only linear activation functions (like the identity function) in all hidden layers, the network would **collapse** into a single-layer linear model. The mathematical reason is that the composition of any number of linear functions is always **another linear function**.

In conclusion, a linear activation function in a hidden layer is a **dead-end**, as it prevents the network from **learning the complex representations** that make it powerful. Non-linear activation functions are the fundamental building blocks that unlock the potential of deep learning.

——— (Answer from `Gemini`)

What I Want to Propose 1: Importance of One-Hot Encoding

I think the importance of **one-hot encoding** or **dummy variable** can be manifested in the scenario that a variable has 3 or more discrete states. In my current project focusing on the **fraud detection** with machine learning, this occurs quite often; for instance, the transactions can be simply categorised into 5 continents, it is unreasonable to label *Asia* to be 1, *Europe* to be 2, and so on. The solution we come up with is to use dummy variables like `is_asia`, `is_europe`,... functions that give either 0 or 1. This can be seen that a problem under lower-dimensional space cannot find a smooth curve to accommodate, but with higher dimensions, it is feasible. This is my own perspective to the **one-hot encoding** on the implementation.