314652021 羅瑋翔

Assignment 2

1. Read Deep Learning: An Introduction for Applied Mathematicians. Consider a network as defined in (3.1) and (3.2). Assume that $n_L = 1$, find an algorithm to calculate $\nabla a^{[L]}(x)$.

2. There are unanswered questions during the lecture, and there are likely more questions we haven't covered. Take a moment to think about them and write them down here.

(3.1) $\quad a^{[1]} = x \in \mathbb{R}^{n_1}$

(3.2) $\quad a^{[l]} = \sigma(W^{[l]} a^{[l-1]} + b^{[l]}) \in \mathbb{R}^{i}, \; i = 1, 2, \dots, L$

1. Define $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$, $a^{[l]} = \sigma(z^{[l]})$.

Then, $\dfrac{\partial a^{[l]}}{\partial a^{[l-1]}} = \dfrac{\partial a^{[l]}}{\partial z^{[l]}} \cdot \dfrac{\partial z^{[l]}}{\partial a^{[l-1]}} = \sigma'(z^{[l]}) \cdot W^{[l]}$

and

$$\sigma'(z^{[l]}) = \begin{bmatrix} \sigma'(z_1^{[l]}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma'(z_{n_l}^{[l]}) \end{bmatrix} \in \mathbb{R}^{n_l \times n_l}$$

$\because \sigma(z_i^{[l]})$ relies on i-th input, and $W \in \mathbb{R}^{n_l \times n_{l-1}}$

$\Rightarrow$ Hence, we have

$$\frac{\partial a^{[L]}}{\partial a^{[1]}} = \frac{\partial a^{[L]}}{\partial a^{[L-1]}} \cdot \frac{\partial a^{[L-1]}}{\partial a^{[L-2]}} \cdot \cdots \cdot \frac{\partial a^{[2]}}{\partial a^{[1]}}$$

$$= (\sigma'(z^{[L]}) \cdot W^{[L]}) \cdot (\sigma'(z^{[L-1]}) \cdot W^{[L-1]}) \cdots (\sigma'(z^{[2]}) \cdot W^{[2]})$$

$$\in \mathbb{R}^{n_L \times n_1}$$

☆ Since we need to obtain $Z^{[l]}$ and $A^{[l]}$ to calculat $\nabla a^{[L]}(x)$, we can use "Forward passing and back propagation" to solve this

Algo:

$a^{[1]} = x$

for $i = 2$ to L

$\quad z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]}$

$\quad a^{[i]} = \sigma(z^{[i]})$ // σ is the activation function.

$\quad S^{[i]} = 1_{n_i}^{T} \cdot \operatorname{diag} \sigma'(z^{[i]})$ // saved this for back propagation

$p^{[L]} = a^{[L]}$

for $i = L$ to 2.

$\quad p^{[i-1]} = (p^{[i]} \odot S^{[i]}) W^{[i]}$ // $\odot$ is elementwise product

$\Rightarrow$ By this, we obtain $p^{[1]}$ $(\nabla a^{[L]}(x))$ #.

2. During the class, it says that using a tanh neural network with $3n+2$ neurons in the layer and $2n+1$ neurons in the output layers, we can approximate the functions x^p, $p = 0, \dots, 2n$ to any desired accuracy.

$\Rightarrow$ The question is, if the domain of x become larger, such as $x \in [-10, 10]$, is the lemma still holds?
And is tanh still an effective way to approximate x^p?
Is there exists any way to approximate x^p for $x \in (-\infty, \infty)$?

Assignment 2

The definition of the neural network
$f \in FC(W = 64, L = 2, d_{in} = 1, d_{out} = 1)$

- The choice of activation function is $tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. (Since it performs a good approximation of polynomials on the interval $x \in [-1, 1]$)
- I intialize W(the weight) with Glorot/Xavier initialization. (Maintain a moderate variace for the outputs of each layer.) And set the bias $b^{[l]}$ to $0_{1 \times n_l}$
.
- Then, I use **Adam optimizer** to maintain a good choice of learning rate α. (Since adam optimizer can choose suitable momentum and velocity overtime, it might better approaches minimum than SGD.)

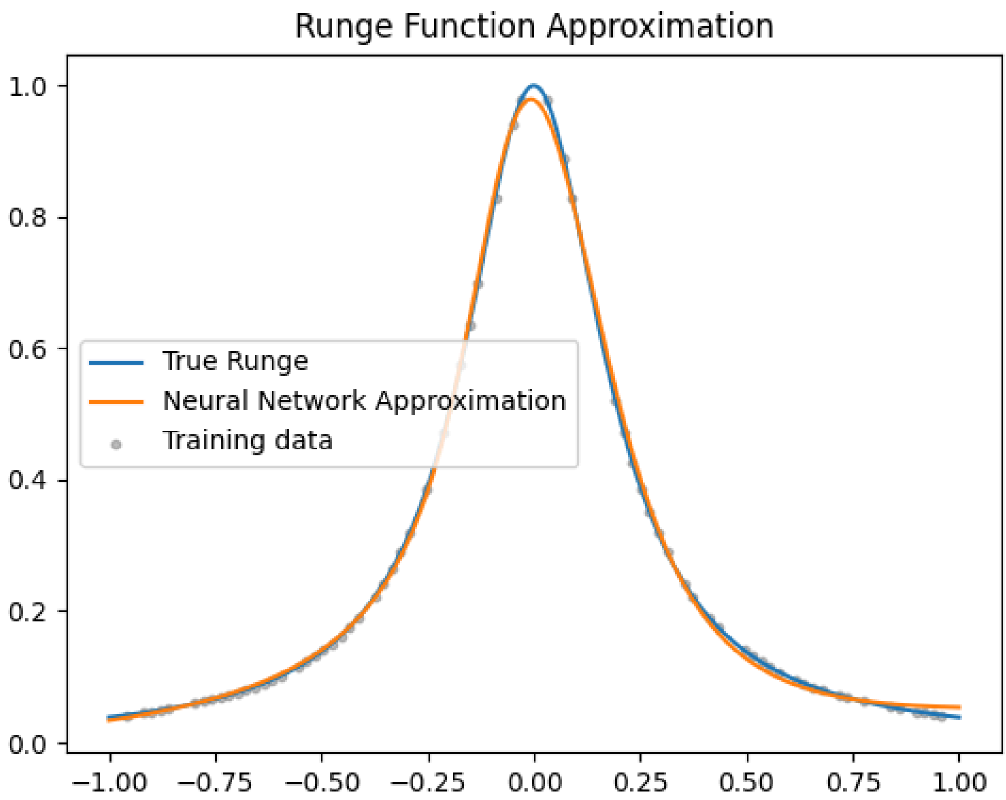

Figure 1. Runge Function Approximation with Neural Network

One can observe is that when x approaches 0, it deviates away.

- Xavier initialization would let most of the inputs not lies near 0
- And N might not be big enough to spread on the interval.

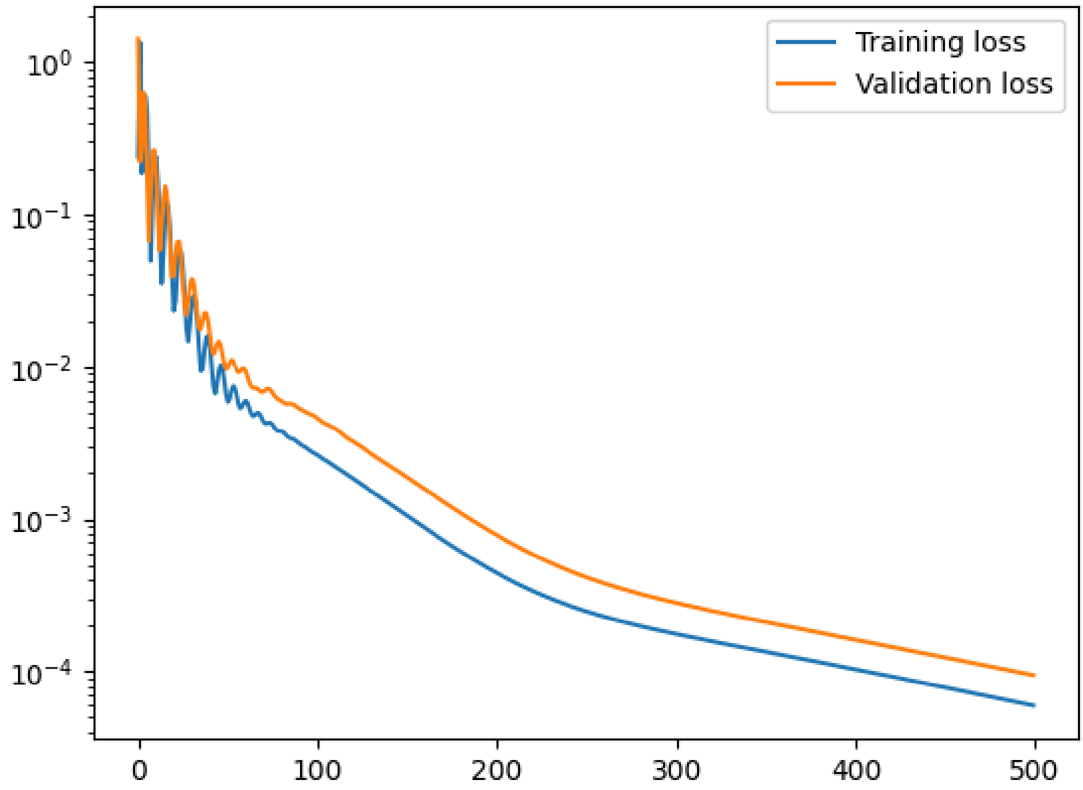

Figure 2. Traning and Validation Loss Curves vs. Epoch

At the start of traing, the loss is higher and oscillating. Since the model just start traing and not learned the charactistic of the function. As epoch grows, the model coverges gradually, and loss decent significantly. At around epoch 500, the validation loss becomes around 10^{-4}