Assignment 3

September 2025

1 Background

Neural network:
A neural network consists of input layer, hidden layers, and output layer.
Each layer has neurons and they are connected between layers. And the connections have weights on them.
The input layer's neuron usually only reads the input data.(In the following, it takes in $x, x \in R^1$)
The hidden layer has multiple neurons, each neuron has an activation function(In the following NN, it's tanh) which takes in the value that previous neurons connected to it and bias.
Last, the output layer outputs the result. Taking the following NN as example, the output layer has s-neurons and there are s results.
So it can be seen as a function that input layer defines the domain, hidden layers determine the form of the function and output layer is the range.

2 Lemma 3.1

Statement:
There exists a shallow tanh neural network(a neural network with only one hidden layer and the neurons' activation function is tanh); it can be used to approximate moonomials(e.g. x^3) with odd degrees.
And for an odd number s, the hidden layer only needs $\frac{s+1}{2}$ neurons to approximate x^p for $p \leq s$ and p is odd at the same time as accurate as we want.
Moreover, this lemma gives scales for the weights of the network, which is $O(\epsilon^{-s/2}(2(s+2)\sqrt{2M})^{s(s+3)})$, M stands for the domain of the neural network with one neuron in the input layer.(If we want more accuracy, the bigger the weights.)
For example, according to this lemma, we can approximate x, x^3, x^5, x^7 at the same time with a NN that has only one 4-neuron hidden layer.

Idea:
An intuitive explanation of the idea of this lemma is to use the properties of tanh that it is an odd function, which makes it good to approximate the odd

function, and $tanh(x) \approx x$ near the origin, which ensures that we can approximate x.
Look at the Taylor expansion of tanh(x):
$tanh(x) = x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - ...$
For the other odd-degree monomials which are odd functions, we can approximate them by several tanh functions which are scaled and combined.
As for the number of neurons that the hidden layer needs, it can be intuitively seen as the number of odd-degree monomials, since each neuron helps isolate these terms.

3 Lemma 3.2

Statement:
There exists a shallow tanh neural network that can be used to approximate moonomials with not only odd but also even degrees.
And for an odd number s, the hidden layer only needs $\frac{3(s+1)}{2}$ neurons to approximate x^p for $p \leq s$ at the same time as accurate as we want.
Moreover, this lemma gives scales for the weights of the network which is $O(\epsilon^{-s/2}(\sqrt{M}(s+2)^{\frac{3s(s+3)}{2}})$, M stands for the domain of the neural network with one neuron in the input layer.

Idea:
The intuitive explanation is to use the odd-degree monomials to construct the even-degree monomials.
Since Lemma 3.1 already gives us odd-degree monomials, then combine these terms, we have even-degree monomials.
For example, if we want x^4, we can combine x and x^3 to get it.
From the example, we can see that in order to approximate even-degree monomials, we need extra 2 neurons that do the odd-degree approximation to do it.
So we can intuitively get that the number of neurons in the hidden layer is $\frac{3(s+1)}{2}$.

4 Question

For a neural network to approximate a function f and its derivative f' at the same time, I find it's hard to train.
I first choose the loss to be MSE loss of f + MSE loss of f' and the result didn't go well.
Then I think that if the MSE loss of f' is weighted then it will be better, and it did be better, but it's not good enough.
I don't know if I should choose another weight or there is other way to make it better?

5 Progamming report

1. The MSE error of the derivative of the given function is 0.0033007229659670527. Note that I choose the training set randomly in the domain, so the error might be slightly different.

2. First, define the Runge function f and f', then generate the training set by choosing 200 points in [-1,1] that are equally spaced and is denoted by x_{train}.

Then I choose 100 points in [-1,1] which are equally spaced and denoted by x_{test}.

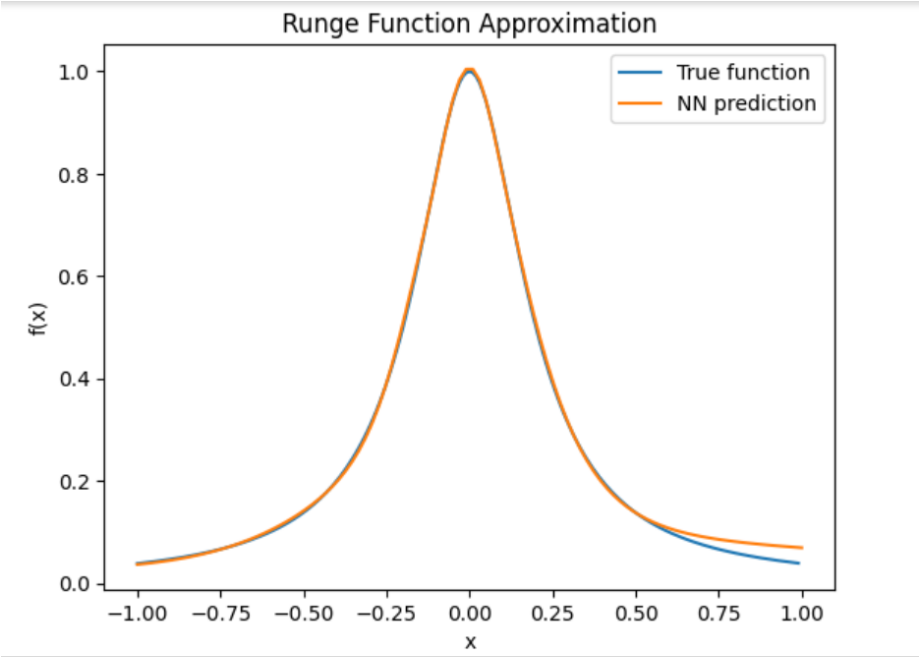
Then define $y_{train} = (y1, y2)$ and $y_{test} = (f(x_{test}), f'(x_{test}))$, where $y1 = f(x_{train})$ and $y2 = f'(x_{train})$.

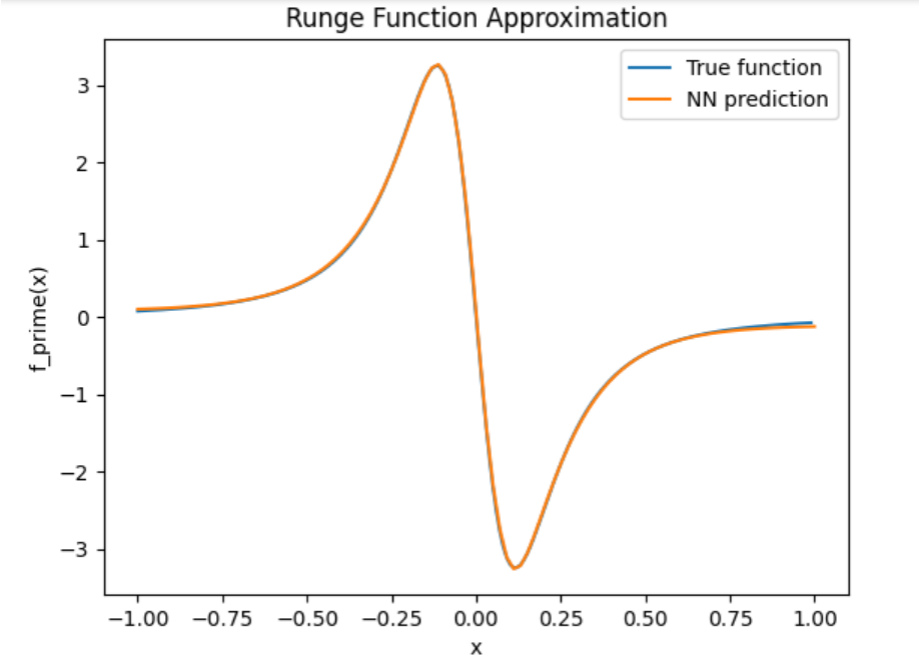
Then I design a neural network with 1-neuron which takkes in x in the input layer, 500-neuron with the tanh activation function in the first and second hidden layer, then the output layer has 2-neuron outputs (f, f') as the hypothesis function.

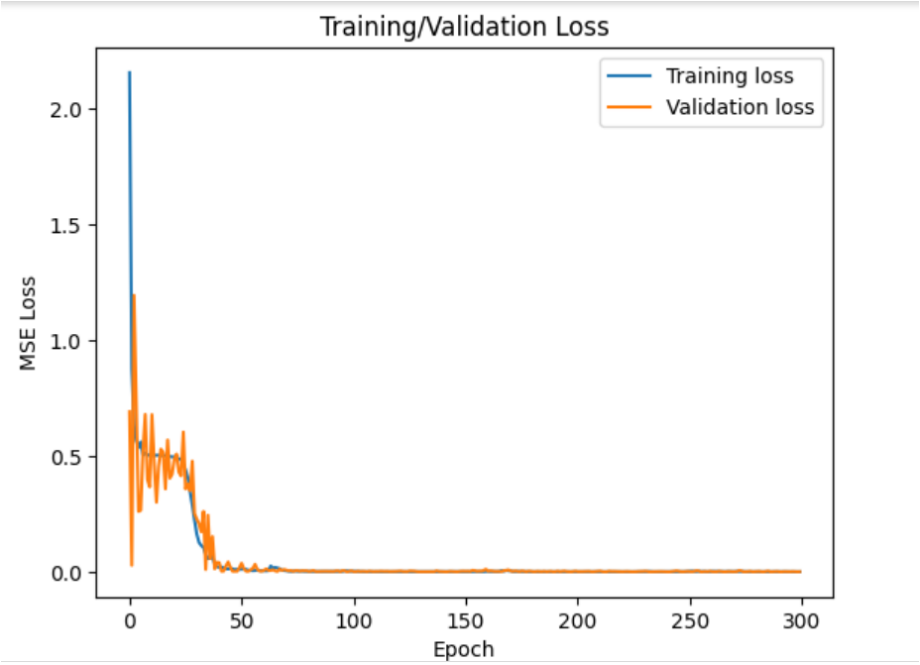
I choose the loss function by combining 'MSE loss of f' and '0.2*MSE loss of f".

Then to train the neural network, it will do a forward and back propagation for 300 epochs while using the mini-batch gradient descent for 32 data each time and Adaptive moment estimation as optimizer.
And the validation set is constructed by choosing 20 percent data in the training set.

Next, we have the following result: MSE error: 0.00028547543815205103


Runge Function Approximation
True function
NN prediction
f(x)
x
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


Runge Function Approximation
True function
NN prediction
f_prime(x)
x
3
2
1
0
−1
−2
−3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


Training/Validation Loss
Training loss
Validation loss
MSE Loss
2.0
1.5
1.0
0.5
0.0
0
50
100
150
200
250
300
Epoch