Neural Network-Based Function Upscaling: The Role of Activation Functions

Jen-Hsuan Li[1]

[1]Department of Electrophysics, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan,
(March 19, 2025)

Abstract

In this experiment, we examine the impact of deep neural network architecture on training performance. Our primary objective is to identify an optimal model configuration that accurately approximates the ground truth. Additionally, we analyze the effects of excessively increasing the number of layers or neurons per layer, which can degrade performance due to overfitting or optimization challenges. Finally, we conduct a preliminary investigation of multi-input DNN models and demonstrate the effectiveness of the Chebyshev nodes method in stabilizing function interpolation.

1 Introduction

Deep neural networks have recently become a widely used tool across various fields, including biomedical research and semiconductor engineering. When addressing engineering problems, it is often necessary to plot curves and investigate the relationships between different variables. In this study, we employ DNNs to learn nonlinear functions.

In our experiment, we use a combination of sinusoidal functions as the ground truth. Rather than directly using these functions, we sample points from them and apply interpolation to generate a new function.

2 Theory

2.1 Evaluation of Activation function

Different activation functions serve distinct purposes in neural networks. The primary role of an activation function is to transform linear inputs into nonlinear outputs, which is fundamental to the functionality of neural networks.

For instance, the Rectified Linear Unit (ReLU) helps mitigate the vanishing gradient problem, facilitating efficient training in deep networks. The Sigmoid function is commonly used for probability-based outputs and binary classification tasks. The Hyperbolic Tangent (Tanh) function, compared to Sigmoid, is more effective for balanced gradient updates.

Name	Equation	Derivative
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(1+e^{-x})^2}$
Hyperbolic Tangent	$\frac{2}{1+e^{-2x}} - 1$	$\frac{1}{\cosh^2 x}$
ReLU [6] [5]	$\max(0, x)$	$\begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$
Leaky ReLU [6]	$\max(0, x) + \alpha \min(0, x)$	$\begin{cases} 1, & \text{if } x \geq 0 \\ \alpha, & \text{otherwise} \end{cases}$
PReLU [6]	$\max(0, x) + \alpha \times \min(0, x)$	$\begin{cases} 1, & \text{if } x \geq 0 \\ \alpha, & \text{otherwise} \end{cases}$
ReLU6 [12]	$\min(\max(0, x), 6)$	$\begin{cases} 0, & \text{if } x < 0 \text{ or } x > 6 \\ 1, & \text{if } 6 \geq x \geq 0 \end{cases}$
ELU [7]	$\max(0, x) + \min(0, \alpha(e^x - 1))$	$\begin{cases} \alpha e^x, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$
SELU [13] [1]	$\gamma(\max(0, x) + \min(0, \alpha(e^x - 1)))$	$\gamma \begin{cases} \alpha e^x, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases}$
Swish [10]	$\frac{x}{1+e^{-x}}$	$\frac{e^x(x+e^x+1)}{(e^x+1)^2}$
Mish [11]	$x \tanh(\log(1 + e^x))$	$\frac{e^x(4e^x x+4x+6e^x+4e^{2x}+e^{3x}+4)}{(2e^x+e^{2x}+2)^2}$

Figure 1: Summary of activation function [1]

2.2 Impact of Hidden Layer Width and Depth

Shallow layers are capable of learning simple features of a function. As the number of layers increases, a deep neural network (DNN) can progressively capture more complex features. However, this improvement is not limitless. If a network becomes excessively deep, gradients may either

diminish (vanishing gradient problem) or grow uncontrollably (exploding gradient problem) during backpropagation, hindering effective training.

The width of hidden layers can be as influential as depth in enhancing network performance. A DNN with wider layers possesses greater capacity to model complex functions. However, excessively wide hidden layers can lead to overfitting, where the network memorizes the training data rather than learning generalizable patterns.

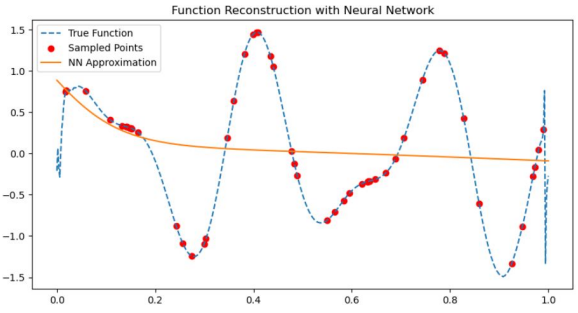

Figure 2: Fitting with Sigmoid

3 Experiment Result

In this section, we present our experimental results. The ground truth functions are generated using Barycentric interpolation. First, we evaluate the performance of a single-input deep neural network (DNN) model by varying the activation functions, the number of sampled points, and the width and depth of the hidden layers.

In the second part, we examine a multi-input model designed to train a family of random functions. This approach allows the model to generalize across different amplitudes and frequencies of sinusoidal functions, requiring only a single training process.

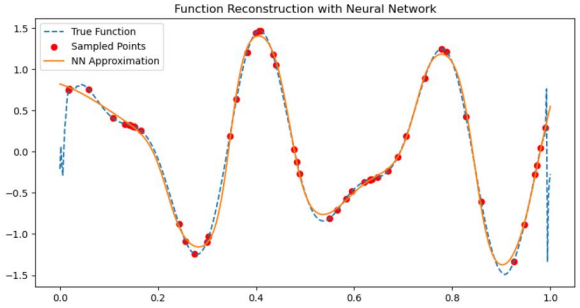

Figure 3: Fitting with Hyperbolic Tangent

3.1 Variation via different activation function

Using the Sigmoid activation function requires a longer training time due to the slow convergence of gradients. Additionally, since the model output includes negative values, a positive activation function may be more suitable. The training times for the Sigmoid and Tanh activation functions are 0.91 seconds and 0.78 seconds, respectively.

Table 1: Activation Function and Training time

Activation Function	Training Time (sec)
Sigmoid	0.91
Hyperbolic Tangent	0.78

Table 2: Activation Function and Training Point MSE

Activation Function	Training Points MSE
Sigmoid	0.595198
Hyperbolic Tangent	0.026820

3.2 Variation via different width of hidden layer

Increasing the width of a hidden layer generally enhances accuracy, as a wider layer can better represent complex functions. Additionally, the Rectified Linear Unit (ReLU) activation function is used instead of Tanh to mitigate the vanishing gradient problem.

However, excessively increasing the width to 1000 neurons per layer leads to performance degradation due to overfitting. Moreover, the training process slows down as the optimizer must update a significantly larger number of parameters. Therefore, it is crucial to identify an optimal width, or "sweet spot." In our experiment, the optimal width ranges from 256 to 512 neurons per layer.

Table 3: Width and Training time

Neuron per layer	Training Time (sec)
100	0.85
256	1.24

Table 4: Width and Training Points MSE

Neuron per layer	Training Points MSE
100	0.031289
256	0.028993

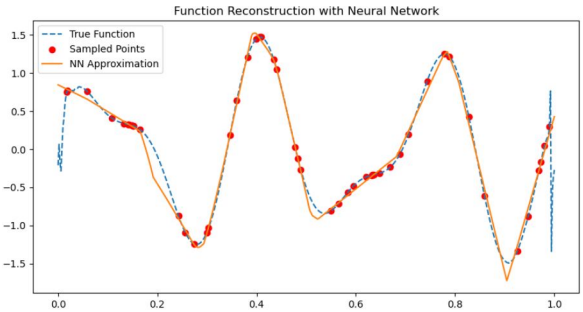

Figure 4: Fitting with 100 neurons per layer

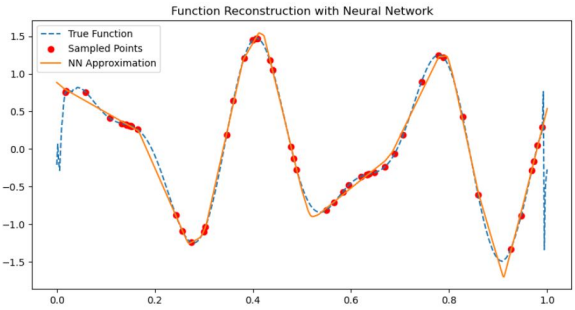

Figure 5: Fitting with 256 neurons per layer

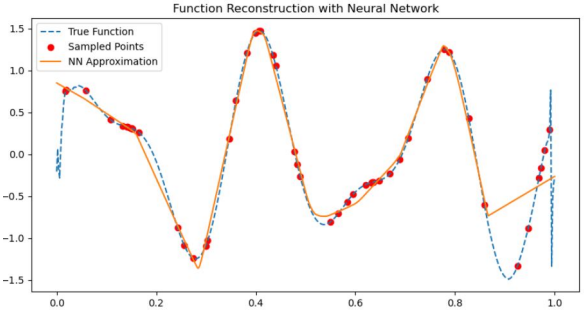

Figure 6: Fitting with 1000 neurons per layer

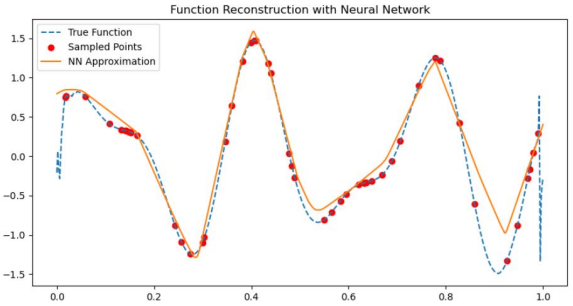

Figure 7: Fitting with 2000 neurons per layer

3.3 Variation via different depth of hidden layer

Deep networks can approximate certain functions more efficiently using fewer neurons. In our case, the target function exhibits a high level of complexity, making it essential to examine the impact of increasing the depth of the network.

However, when the depth is increased to 11 layers, performance deteriorates due to overfitting. Additionally, excessively deep networks may suffer from gradient-related issues, where gradients either become too small (vanishing gradient problem) or excessively large (exploding gradient problem) during backpropagation. To mitigate the vanishing gradient issue, we employ the Rectified Linear Unit (ReLU) activation function.

Table 5: Number of layers and Training time

Number of layers	Training Time (sec)
4	1.25
8	1.98

Table 6: Number of layers and Training time

Number of layers	Training Points MSE)
4	0.023119
8	0.024880

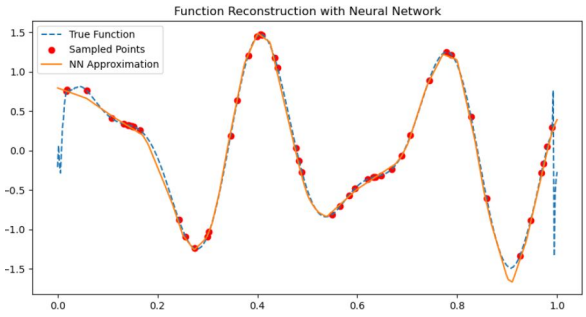

Figure 8: Fitting with 4 layers

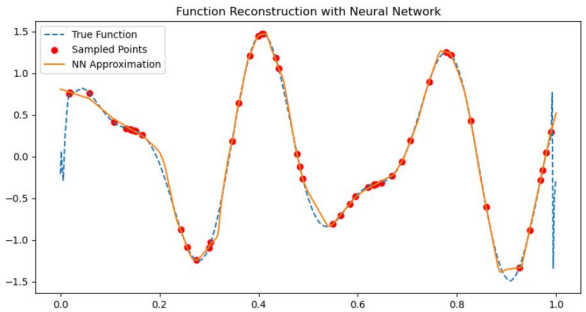

Figure 9: Fitting with 8 layers

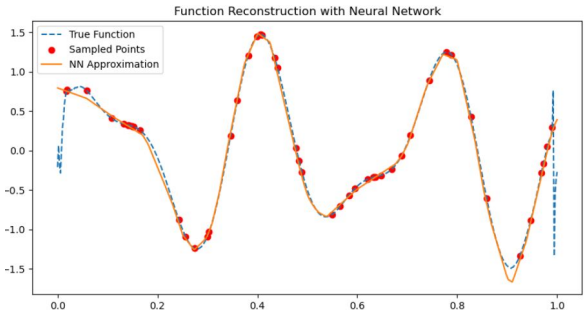

Figure 10: Fitting with 11 layers

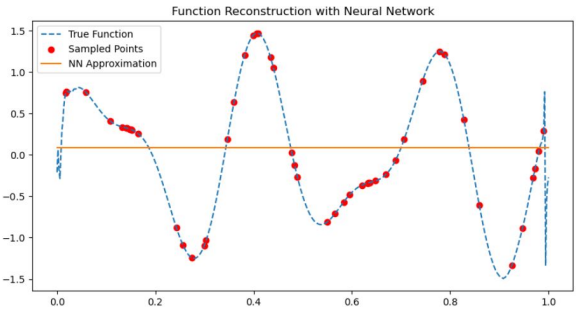

Figure 11: Fitting with 16 layers

3.4 Variation via different size of sample

Our original model utilizes 50 sample points, which appears to be an optimal sample size. To examine the effect of sample size, we trained the network with 25 sample points and observed that the model struggles to learn the function effectively due to insufficient information.

Conversely, when increasing the sample size to 75 points, the generated function becomes more intricate and unstable, containing excessive details. Accurately fitting such a function requires a higher-capacity model with greater complexity.

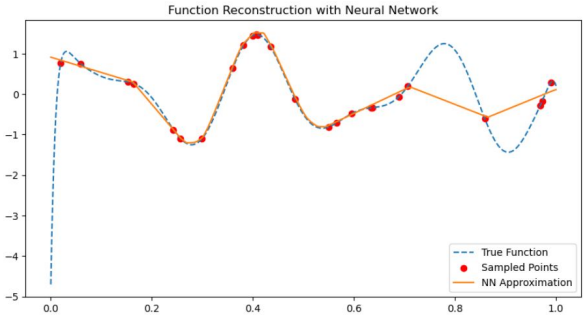

Figure 12: Fitting with 25 sample points

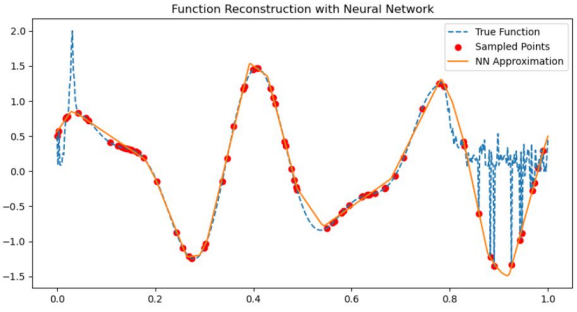

Figure 13: Fitting with 75 sample points

3.5 Generating Class of Barycentric-Interpolated Functions

In this section, we construct a multi-input deep neural network model capable of processing a size-100 input vector and producing a size-512 output. Additionally, we generate random functions within the $(0, 1)$ interval by varying both frequency and amplitude to ensure uniqueness among the selected functions.

To distribute the sample points, we employ Chebyshev nodes, which are not uniformly spaced

but are denser near the boundaries. This non-uniform distribution helps mitigate interpolation errors and improves stability.

As observed in Task 1, increasing the number of sample points causes the generated function to exhibit more pronounced oscillations. By concentrating more points near the edges, Chebyshev nodes help prevent excessive oscillations and maintain interpolation stability. Since Chebyshev nodes minimize interpolation error, fewer points are required to achieve an accurate function approximation.

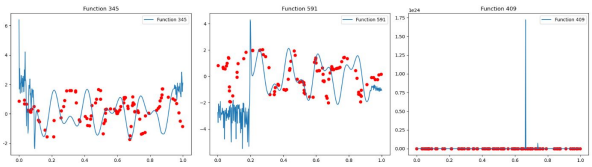

Figure 14: Generating random functions with uniform nodes

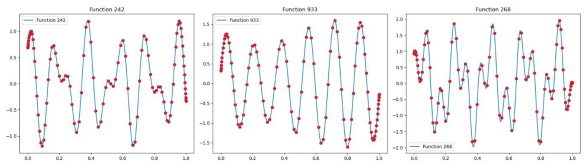

Figure 15: Generating random functions with Chebyshev nodes

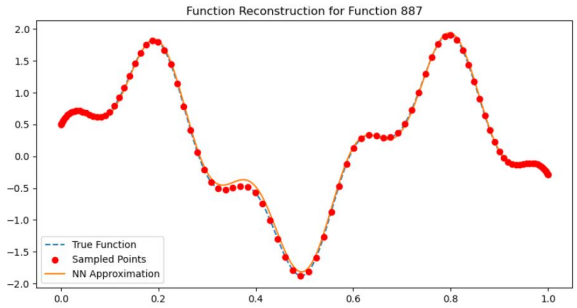

Figure 16: Fitting random function 887

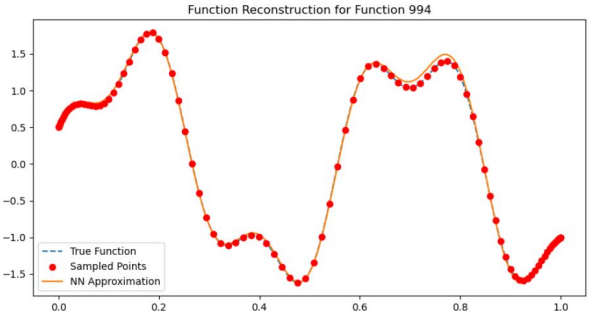

Figure 17: Fitting random function 994

We observe that both training and test accuracy are consistently high.

From the fitting results, the test and training functions closely resemble the ground truth, which is defined by the Barycentric-interpolated random function. Our strategy of utilizing Chebyshev nodes effectively stabilizes the boundaries of the interpolated function.

In contrast, in previous experiment, where Chebyshev nodes were not used, the generated function became unstable when employing 75 sample points. However, in this experiment, even with 100 sample points, the function remains stable, demonstrating the effectiveness of this approach.

4 Conclusion

In this experiment, we employ barycentric interpolation to generate functions, treating them as ground truth. However, this method tends to exhibit instability as the number of sample points increases. To address this issue, we utilize Chebyshev nodes, which help mitigate the instability. Additionally, we investigate the impact of network architecture, specifically the width, depth, and activation function, on the fitting performance of a single-input deep neural network model.

5 Reference

[1] Rasamoelina, Andrinandrasana David, Fouzia Adjailia, and Peter Sinčák. "A review of activation function for artificial neural network." In 2020 IEEE 18th world symposium on applied machine intelligence and informatics (SAMI), pp. 281-286. IEEE, 2020.

[2] Touretzky, David S., and Dean A. Pomerleau. "What's hidden in the hidden layers." Byte 14, no. 8 (1989): 227-233.