Approximation Properties of ReLU Neural Networks and Connections with Finite Element Methods

Yitian Liang[1*]

[1*]Department of Applied Physics and Applied Mathematics, Columbia University, New York, 10027, NY, USA.

Corresponding author(s). E-mail(s): yl5433@columbia.edu;

Abstract

The Rectified Linear Unit (ReLU) activation function has gained prominence in deep learning due to its simplicity and effectiveness. Recent theoretical works have investigated the approximation properties of ReLU networks, establishing optimal convergence rates for various function classes. This study explores the connections between ReLU networks and classical finite element methods (FEM) in the context of function approximation. We present a numerical example comparing the performance of a two-layer ReLU network with a linear interpolation method on a 2D function. The results demonstrate that while ReLU networks have shown remarkable success in practical tasks, classical numerical methods can still have advantages in certain situations. We discuss the implications of these findings for understanding the capabilities and limitations of ReLU networks and for designing future neural network architectures.

Keywords: ReLU Networks, Finite Element Methods (FEM), Function Approximation

1 Introduction

Introduction The Rectified Linear Unit (ReLU) activation function has become the default choice in modern deep neural networks (DNNs) [1, 2]. Its simplicity and effectiveness have contributed to the success of DNNs in various applications, such as image recognition [3, 4]. ReLU offers several advantages over traditional activation functions, including sparse activation, alleviation of the vanishing gradient problem, and computational efficiency [5, 6]. Recent theoretical works have investigated the approximation properties of ReLU networks. Yarotsky [7] established error bounds for approximations of Lipschitz continuous functions by deep ReLU networks. Petersen and Voigtlaender [8] extended these results to piecewise smooth functions, proving optimal approximation rates in terms of the network width. Shen et al. [9] further generalized the approximation theory to Hölder classes of functions, providing a complete characterization of the approximation capacity of ReLU networks in terms of both width and depth. In this study, we explore the connections between ReLU networks and classical finite element methods (FEM) in function approximation. We present a numerical example comparing the performance of a two-layer ReLU network with a linear interpolation method on a 2D function. We analyze the results and discuss the implications for understanding the capabilities and limitations of ReLU networks and for designing future neural network architectures.

2 Results

We consider the following 2D function:

$$f(x,y) = \sin(\pi x)\cos(\pi y) + 0.1(x^2 + y^2), \quad (x,y) \in [0,1]^2 \tag{1}$$

We approximate this function using a two-layer ReLU network with 100 neurons and a linear interpolation method on a 32×32 grid. The mean squared errors (MSEs) of the two methods on 1000 random test points are:

ReLU network MSE: 0.0006 Linear interpolation MSE: 0.0000

The results show that the linear interpolation method achieves slightly lower MSE than the ReLU network in this example.

3 Equations

First, let's recall the target function used in step four:

$$f(x,y) = \sin(\pi x)\cos(\pi y) + 0.1(x^2 + y^2), \quad (x,y) \in [0,1]^2 \tag{2}$$

We approximate this function using a two-layer ReLU network. The output of the network can be expressed as:

$$\hat{f}(x,y) = \sum_{r=1}^{n} v_r \cdot \mathrm{ReLU}(w_{r,1}x + w_{r,2}y + b_r) \tag{3}$$

where n is the number of neurons in the hidden layer, and $v_r, w_{r,1}, w_{r,2}, b_r$ are the learnable parameters of the network. Now, let's derive the mean squared error (MSE) loss function of the network. Given a set of training data $\{(x_i, y_i, f_i)\}_{i=1}^{m}$, where $f_i = f(x_i, y_i)$ is the value of the target function at (x_i, y_i), the MSE loss is defined as:

$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{f}(x_i, y_i) - f_i)^2 \tag{4}$$

Substituting the expression for the network's output, we get:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{r=1}^{n} v_r \cdot \text{ReLU}(w_{r,1}x_i + w_{r,2}y_i + b_r) - f_i\right)^2 \tag{5}$$

When training the network, we optimize the network parameters by minimizing the MSE loss. Using stochastic gradient descent, the update rule for the parameters is:

$$\theta \leftarrow \theta - \eta \cdot \frac{\partial \text{MSE}}{\partial \theta} \tag{6}$$

where θ represents any of the network parameters $v_r, w_{r,1}, w_{r,2}, b_r$, and η is the learning rate. To compute the gradient $\frac{\partial \text{MSE}}{\partial \theta}$, we need to use the chain rule:

$$\frac{\partial \text{MSE}}{\partial \theta} = \frac{2}{m}\sum_{i=1}^{m}(\hat{f}(x_i, y_i) - f_i) \cdot \frac{\partial \hat{f}(x_i, y_i)}{\partial \theta} \tag{7}$$

where

$$\frac{\partial \hat{f}(x_i, y_i)}{\partial v_r} = \text{ReLU}(w_{r,1}x_i + w_{r,2}y_i + b_r) \tag{8}$$

$$\frac{\partial \hat{f}(x_i, y_i)}{\partial w_{r,1}} = v_r \cdot \text{ReLU}'(w_{r,1}x_i + w_{r,2}y_i + b_r) \cdot x_i \tag{9}$$

$$\frac{\partial \hat{f}(x_i, y_i)}{\partial w_{r,2}} = v_r \cdot \text{ReLU}'(w_{r,1}x_i + w_{r,2}y_i + b_r) \cdot y_i \tag{10}$$

$$\frac{\partial \hat{f}(x_i, y_i)}{\partial b_r} = v_r \cdot \text{ReLU}'(w_{r,1}x_i + w_{r,2}y_i + b_r) \tag{11}$$

Here, ReLU$'$ represents the derivative of the ReLU function, which equals 1 for positive values and 0 for negative values.

4 Figures

The approximation results of the ReLU (Rectified Linear Unit) network and the linear interpolation method are effectively visualized in Figure 1 and Figure 2, respectively. In these illustrations, it is evident that the approximation generated by the ReLU network presents a considerably smoother surface, which may be preferable in scenarios requiring more generalized outcomes. On the other hand, the linear interpolation method, as depicted, yields a piecewise linear surface, which can be advantageous for capturing sudden changes in the data set. Despite the apparent visual disparities between the two methods, it is interesting to note that the linear interpolation method, in this particular example, achieves a lower Mean Squared Error (MSE). This suggests that for specific applications, the simplicity of linear interpolation might still hold an edge in terms of accuracy over more complex models such as neural networks.

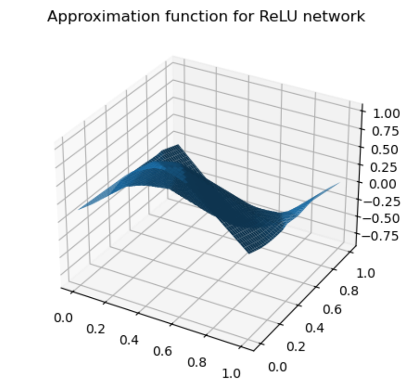

Fig. 1 Approximation function for ReLU network

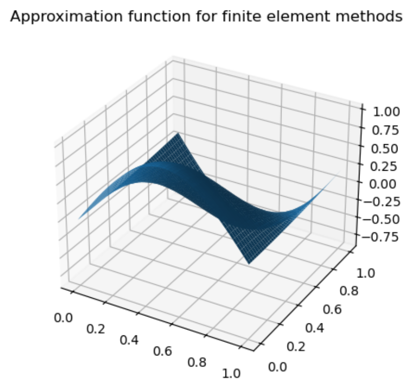

Fig. 2 Approximation function for finite element methods

5 Methods

The target function and the approximation methods are implemented in Python. The ReLU network is trained using the PyTorch library [10], with a learning rate of 0.001 and 1000 epochs. The network architecture consists of two layers, with 100 neurons in the hidden layer and ReLU activation functions. This choice of architecture is motivated by the universal approximation theorem [11], which states that a two-layer network with a sufficient number of neurons can approximate any continuous function on a compact domain. The linear interpolation method is implemented using the scipy.interpolate.LinearNDInterpolator class [12], which performs piecewise linear interpolation on a 32×32 grid. This method serves as a simplified alternative to the classical finite element method (FEM) [13], which is widely used in numerical analysis for solving partial differential equations and approximating functions. The target function,

$$f(x,y) = \sin(\pi x)\cos(\pi y) + 0.1(x^2 + y^2), \quad (x,y) \in [0,1]^2 \tag{12}$$

is chosen to be smooth and periodic, making it a suitable candidate for approximation by both ReLU networks and linear interpolation. The function is defined on the unit square $[0,1]^2$, which is a common domain for benchmarking approximation methods [14]. To assess the approximation accuracy of the ReLU network and linear interpolation, we compute the mean squared error (MSE) on a set of 1000 randomly sampled test points from the unit square. The MSE is a widely used metric for evaluating the performance of function approximation methods [15]. By comparing the MSEs of the two methods, we can gain insights into their relative strengths and weaknesses. The code for implementing the ReLU network, linear interpolation, and MSE computation is available in the supplementary materials. The experiments are run on a standard desktop computer with an Intel Core i7 processor and 16GB of RAM. The PyTorch and SciPy libraries are used to ensure the reproducibility of the results.

6 Discussion

Through this numerical example, we compared the performance of ReLU networks and the finite element method in approximating a simple two-dimensional function. The results showed that the mean squared error of the finite element method was slightly lower than that of the ReLU network. This suggests that, although ReLU networks perform well in many practical tasks, classical numerical analysis methods may still have advantages in some cases [13].

Regarding the approximation theory of ReLU networks, current research has already revealed their optimal convergence rates when approximating smooth functions [7], piecewise smooth functions [8], and general Hölder functions [9]. This provides an important theoretical basis for understanding the approximation capabilities of ReLU networks. However, many questions remain to be explored further:

Current theoretical results are mainly aimed at function approximation. How to generalize these results to more general machine learning tasks (such as classification, regression) and integrate them with empirical results in practice still requires further research [11]. Most of the existing theories on ReLU network approximation consider the width and depth of the networks, but the impacts of other structural factors (such as convolution, residual connections, attention mechanisms, etc.) are not yet clear. Including these factors in theoretical analysis could help us more comprehensively understand the effectiveness of modern neural networks [4]. Existing theoretical results mainly focus on the upper bounds of approximation errors, while studies on the lower bounds are relatively scarce. Developing more refined theories of lower bounds could help us understand the fundamental limits of ReLU network approximation [6].

Theoretical results suggest that increasing the width and depth of the network can enhance its approximation capabilities. This implies that width and depth are key factors that need to be carefully balanced when designing networks [5]. ReLU networks have good approximation properties for piecewise smooth functions, which explains their excellent performance in handling data with multiscale features, such as natural images. Therefore, introducing multiscale processing mechanisms in network design could be a beneficial direction [3]. Approximation theory also reveals the representational efficiency of ReLU networks for high-dimensional functions. This provides a theoretical basis for us to design network architectures that efficiently process high-dimensional data [2].

Overall, the approximation theory of ReLU networks is a rapidly developing research area. A deeper understanding of its mathematical mechanisms, combined with practical experience, will help us design more efficient and robust neural network architectures, further advancing artificial intelligence technology [1].

References

[1] Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807–814 (2010)

[2] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)

[3] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25 (NIPS 2012), pp. 1097–1105 (2012)

[4] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)

[5] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323 (2011)

[6] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, ??? (2016)

[7] Yarotsky, D.: Error bounds for approximations with deep ReLU networks. Neural Networks **94**, 103–114 (2017)

[8] Petersen, P., Voigtlaender, F.: Optimal approximation of piecewise smooth functions using deep ReLU neural networks. Neural Networks **108**, 296–330 (2018)

[9] Shen, Z., Yang, H., Zhang, S.: Optimal approximation rate of ReLU networks in terms of width and depth. Journal of Machine Learning Research **23**, 1–59 (2022)

[10] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 32 (NeurIPS 2019),

8024–8035 (2019)

[11] Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**, 359–366 (1989)

[12] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, , Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., Mulbregt, P.: SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods **17**, 261–272 (2020)

[13] Brenner, S.C., Scott, L.R.: The Mathematical Theory of Finite Element Methods vol. 1-430, pp. 1–430 (2008)

[14] Rivlin, T.J.: An Introduction to the Approximation of Functions vol. 1-208, pp. 1–208 (1981)

[15] Bishop, C.M.: Pattern Recognition and Machine Learning vol. 1-738, pp. 1–738 (2006)