Serdica Journal of Computing 17(1), 2023, pp. 1-16, 10.55630/sjc.2023.17.1-16
Published by Institute of Mathematics and Informatics, Bulgarian Academy of Sciences

Higher Order Orthogonal Polynomials as Activation Functions in Artificial Neural Networks

Burak Nebioglu[1], Alexander I. Iliev[2]

[1]School of Technology,
SRH Berlin University of Applied Sciences, Germany
buraknebioglu@gmail.com

[2]Institute of Mathematics and Informatics,
Bulgarian Academy of Sciences, Bulgaria
ailiev@berkeley.edu

Abstract

Activation functions are used in Artificial Neural Networks to provide non-linearity to the system. Several different activation functions in use are very well known by almost any AI practitioner however this is not the case for polynomial activation functions. Increasing attention to these valuable mathematical functions can encourage more research and help to fill the gap. During this work, Chebyshev and Hermite orthogonal polynomials were used as activation functions. Calculations were conducted on 3 different datasets with different hyperparameters. According to the results, calculations done by Chebyshev activation functions take less time, but Chebyshev can be more fragile depending on the solved problem. On the other hand, Hermite shows a more robust and generalized behavior, it is less dependent on the problem type, and it improves by necessary adjustments.

Keywords: Activation function, Chebyshev orthogonal polynomials, Hermite orthogonal polynomials, Artificial Neural Networks

ACM Computing Classification System 2012: Computing methodologies → Artificial intelligence

Mathematics Subject Classification 2020: 68T07

Received: April 26, 2023, *Accepted:* June 30, 2023, *Published:* July 6, 2023

Citation: Burak Nebioglu, Alexander I. Iliev, Higher Order Orthogonal Polynomials as Activation Functions in Artificial Neural Networks, Serdica Journal of Computing 17(1), 2023, pp. 1-16, https://doi.org/10.55630/sjc.2023.17.1-16

1 Introduction

The idea of artificial intelligence (AI) is not new, it has roots going back to 1950s with a bunch of enthusiastic scientists and the idea behind is simply making machines that can learn the rules from data instead of hard coding. The reason why it has become much more popular and widely used nowadays is, mainly due to the improvements done in hardware and availability to reach huge amount of data. Besides those reasons democratization of data science also has a huge impact on improvements. By saying democratization, we imply the reachability to data science tools and methods not only by academicians but also by any discipline. AI is the most general term which includes machine learning (ML) and deep learning (DL). Even it seems like a magic to the everyday user AI code basically runs code which makes calculations and due to the nature of calculations done in a digital environment by some approximations. We as scientists face some bottlenecks depending on the conditions. Subject itself is very broad and solutions can be very specific. As it is visible at Figure 1 AI is the universal set which includes ML and DL, which also represents the development of AI journey. Most of the time it is common to hear that if AI is the engine, data is the fuel of this engine because of this we use tools related with this description.

Let's examine hardware such as Central Processing Unit (CPU), Graphical Processing Unit (GPU) and Tensor Processing Unit (TPU). Numerical calculations are expensive and time consuming, so depending on the calculation power GPU or TPU are preferred instead of CPU for heavy calculations. When we consider Data part of this equation, ideas mainly from statistics opens the doors for us. At the end of the day, all we want is to get a satisfying result from our calculations. Sometimes we do achieve sometimes not, depending on our expectations or criteria. This result depends on multiple factors that we choose even before we start to run our calculations. First, we need to consider the nature of the problem and focus on the methods we will apply to see whether it is suitable or not. What kind of data is used? If it is a sequence of numbers such as daily stock prices and we want to predict next day's price a regression line can be useful on the other hand if we are dealing with colored image dataset which is represented by pixel values between (0, 255) for each channel (r, g, b), Convolutional Neural Network (CNN) will be used.

Regarding to CNN solution other questions arise such as how deep the network should be, which activation function should be used, dropout applied or not, optimizer type. Such questions must be considered at the very beginning of our work. There are already some ready to use approaches turned into best

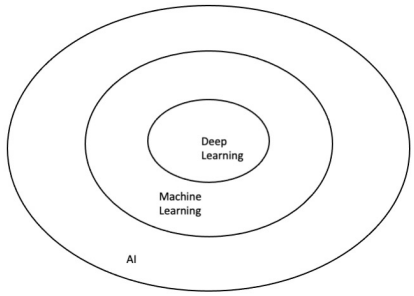

Fig. 1: AI, Machine Learning, Deep Learning relation

practices by the experiences of many experts. After all there is no magical formula for absolute success rate, results are dependent on the elements and their relation, in other words interaction between each other. Knowing and understanding the nature of the problem and related tools on our belt will improve the expected results.

2 Activation functions

Widely used activation functions can be divided into two major groups: gate like functions and non-saturating functions. In gate like functions input is mapped into limited ranges such as $[0, 1]$ or $[-1, 1]$ – Sigmoid, Tanh and Softsign are such functions. However, in non-saturating activation functions when input value grows it does not approximate to a constant – ReLU, LReLU, ELU and SELU are such functions [1]. As described in [2] another property of activation functions is their monotonic or non-monotonic behavior. When a function f is called monotonic that means it is either entirely increasing or entirely decreasing. This predictable behavior of activation function results in faster optimization. Non-monotonic activation functions can also be used in this case it may take more time to train the network.

2.1 Linear activation function

Linear activation functions simply output a result which is a multiplication of input with a constant. In Figure 3 we can see an example of a linear function in this case our k value is equal to 2. As this is a linear function, we will get a

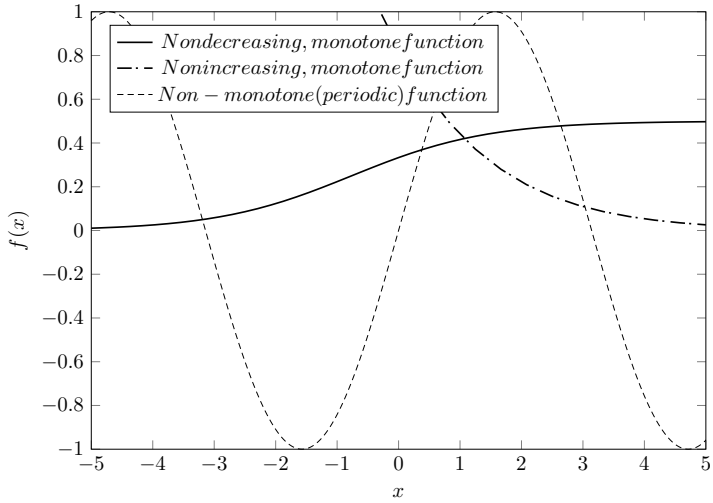

Fig. 2: Monotonic functions taken from [3]

constant value because of the derivative.

$$f(x_i) = kx_i \tag{1}$$

If all the activation functions used in the network are linear it does not matter how deep the network is, the result will be just like a single-layered network. The derivative of this function equals k which is a constant. Because of this it does not hold a relationship with the input, consequently error cannot be decreased by the gradient. Output can be any value between $(-\infty, \infty)$.

Table 1: Advantage and disadvantage of primary activation functions from [4].

AFs	Diminishing gradients	Limited non-linearity	Optimization difficulty	Lack of adaptability	Computational inefficiency
Sigmoid	Yes	No	Yes	Yes	Yes
Tanh	Yes	No	Partial	Yes	Yes
ReLU	Partial	Yes	Partial	Yes	No
ELU	No	Partial	No	Yes	Partial
APL	No	Partial	No	No	No
Swish	No	Partial	No	No	Partial

2.2 Sigmoid

This is an S-shaped monotonic function as seen in the below graph Figure 4. In sigmoid function values are bounded between upper and lower values which are 1 and 0. And because this is not a linear function at each point, we get different derivative results or slopes. As we can also see from the graph while y axes values are approaching to both limit values, derivatives tend to approach to zero. Because of this behavior, we start to face the problem called vanishing gradients. When vanishing gradients occurs network's learning will be impossible because signal will not be passing through neuron to the weights and to the data. Sigmoid function keeps its popularity in binary classification where the output is either 0 or 1. If the value is greater than 0.5 result will be 1 else 0.

$$f(x_i) = \frac{e^{x_i}}{1 + e^{x_i}} = \frac{1}{1 + e^{-x_i}} \tag{2}$$

$$f'(x_i) = \frac{e^{-x_i}}{(1 + e^{-x_i})^2} \tag{3}$$

2.3 Tanh

The main difference between sigmoid and tanh is in their bounding values while sigmoid had values between 0 and 1 tanh has values between -1 and 1 and unlike sigmoid its output is zero-centered, it can be obtained from sigmoid function. Just like sigmoid function tanh also suffers from vanishing gradients problem. Mean of the outputs are 0 or very close to zero because of the range $[-1, 1]$. Because learning is much easier ANN use it mostly in hidden layers.

$$\tanh(x_i) = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}} \tag{4}$$

$$\tanh'(x_i) = 1 - \tanh(x_i)^2 \tag{5}$$

2.4 ReLU

ReLU which means rectified linear unit is a piecewise function that has 2 possible outputs. If the input value is positive, we get the input value back else if the input is negative return value is always zero. This activation function is the mainly used activation function on many models due to its efficiency and results. The gradient value for negative values is 0 and for positive input, the value's gradient is 1. The negative side of this function is that values will be stuck in

an inactive state and die when the inputs are negative which is called Dying ReLU. It looks like a linear function but in fact it is nonlinear. In comparison to the sigmoid and tanh it is easier to compute this function. It is also a very popular choice in deep layers of CNN or DL.

$$f(x_i) = \max(0, x_i) = \begin{cases} x_i, x_i > 0 \\ 0, x_i < 0 \end{cases} \tag{6}$$

$$f'(x_i) = \begin{cases} 1, x_i > 0 \\ 0, x_i < 0 \end{cases} \tag{7}$$

2.5 Leaky ReLU

Main difference between ReLU and LReLU is how they react to negative values. While ReLU is returning zero for negative input values LReLU does not return zero. Instead of this it returns the input value multiplied with a constant. This is done for utilizing negative values against vanishing gradient problem. One difficulty in this method is finding the best slope value which can depend on problem and network type. Output range is between $(-\infty, \infty)$.

$$f(x_i) = \begin{cases} x_i, x_i > 0 \\ \alpha x_i, x_i < 0 \end{cases} \tag{8}$$

$$f'(x_i) = \begin{cases} 1, x_i > 0 \\ \alpha x_i, x_i < 0 \end{cases} \tag{9}$$

3 Polynomial activation functions in AI

Polynomials are heavily used in theory-based models for calculating numerical approximations and dynamic system modeling. Researchers started to use polynomial activation functions around the beginning of the 1990s. Recently interest in those valuable functions decreased because of their fragile behavior which we face while training them. One of its main difficulties is related to data flow explosion problem. Due to this reason polynomial activation functions were applied to single hidden layer networks [5–7]. Higher degree polynomials can be represented by a single hidden layer perceptron by providing more hidden units [8].

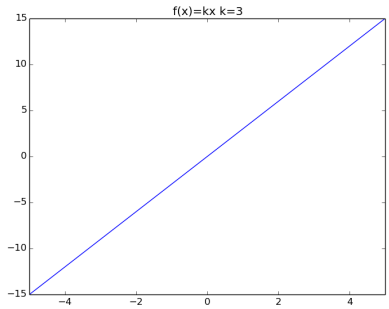

Fig. 3: Linear activation function

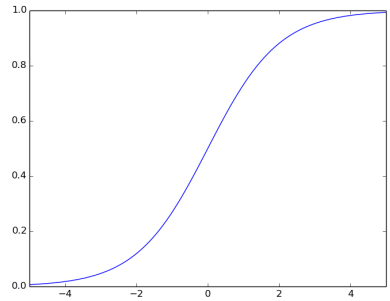

Fig. 4: Sigmoid activation function

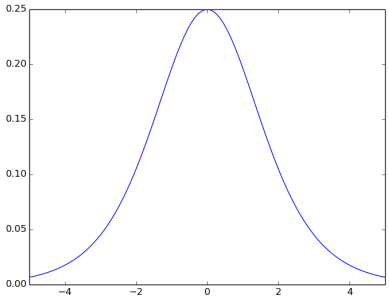

Fig. 5: Sigmoid activation function derivative

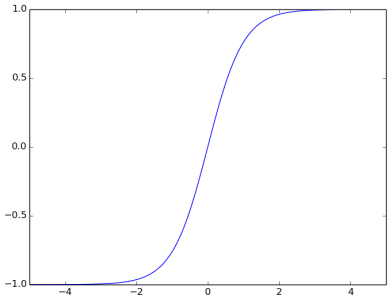

Fig. 6: Hyperbolic tangent activation function

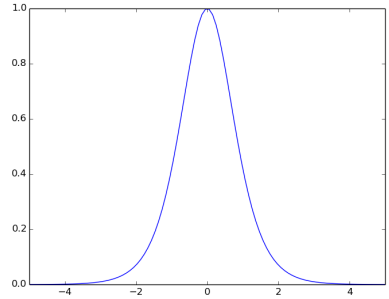

Fig. 7: Hyperbolic tangent activation function derivative

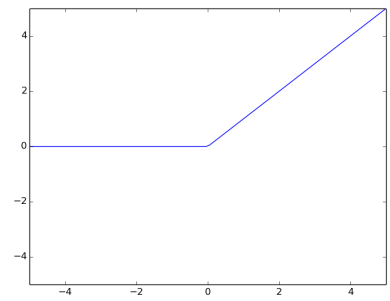

Fig. 8: Relu activation function

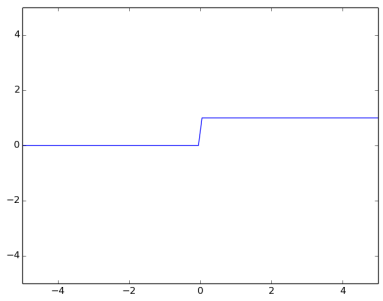

Fig. 9: Relu activation function derivative

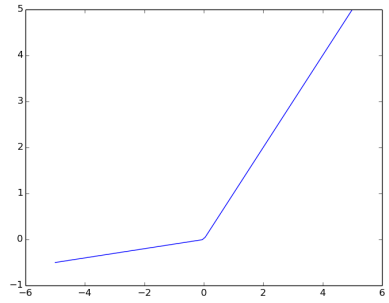

Fig. 10: Leaky ReLU activation function

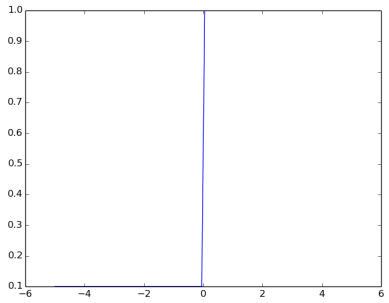

Fig. 11: Leaky ReLU activation function derivative

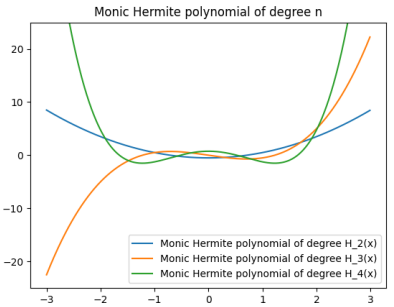

Fig. 12: Hermite polynomial graph adapted from scipy.special.hermite

3.1 Orthogonal polynomial functions

Orthogonal polynomial functions are a special set of polynomials that have the property of being orthogonal to each other over a certain interval. Specifically, let's say we have a weight function $w(x)$ and we want to find a set of polynomials $P_n(x)$ that are orthogonal with respect to this weight function over an interval $[a, b]$. That means we want to find a set of polynomials that satisfy the following equation:

$$\int_a^b P_m(x)P_n(x)w(x)dx = 0, \quad m \neq n. \tag{10}$$

This means that the inner product of any two different orthogonal polynomials is zero, when we multiply each by the weight function and integrate over the interval $[a, b]$. There are several types of orthogonal polynomials, including Legendre polynomials, Hermite polynomials and Chebyshev polynomials.

Now, let's add a trigonometric identity to these orthogonal polynomial functions to create orthogonal trigonometric polynomial functions.

As [9] explains in detail, we can start explaining orthogonal polynomial functions first by adding a trigonometric identity $2\cos m\theta\ \cos n\theta = \cos{(m+n)\theta} + \cos{(m-n)\theta}$ which will be used in the next step.

For $m \neq n$ we have that $cosm\theta$ and $cosn\theta$ are orthogonal in $(0, \pi)$ interval:

$$\int_0^\pi \cos m\theta\ \cos n\theta\, d\theta = 0, \quad m \neq n \quad m, n = 0, 1, 2... \tag{11}$$

This fact shows the vanishing of (11) $\{1, \cos\theta, \cos 2\theta, ... \cos n\theta, ...\}$ is orthogonal for $(0, \pi)$.

The Legendre polynomials are another example of orthogonal polynomial functions. They are defined over the interval $[-1, 1]$ and are orthogonal with respect to the weight function $w(x) = 1$. By using $x = cos\theta$ we can write (11) as:

$$\int_{-1}^1 T_m(x)\, T_n(x)(1-x^2)^{-1/2}\, dx = 0, \quad where \quad m \neq n \tag{12}$$

$$T_n(x) = \cos n\theta = \cos(n\cos^{-1} x) \quad -1 \leq x \leq 1$$
$$T_0(x) = 1$$
$$T_1(x) = \cos\theta = x$$
$$T_2(x) = 2x^2 - 1$$
$$T_3(x) = 4x^3 - 3x$$

$T_n(x)$ are orthogonal polynomials with regard to $(1-x^2)^{-1/2}$,

$$\begin{aligned} &\{P_n(x)\}_{n=0}^{\infty}, \\ &\int_a^b P_m(x)P_n(x)w(x)dx = 0 \quad m \neq n, \end{aligned} \tag{13}$$

where w is the weight function.

3.1.1 Hermite polynomial

Hermite polynomials are used in different areas because of their orthogonality in $(-\infty, \infty)$. In [10] it is mentioned that networks that are based on Hermite have nice noise stability.

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} \tag{14}$$

The polynomials H_n are orthogonal over $(-\infty, \infty)$ with weight function e^{-x^2}.

3.1.2 Chebyshev polynomial

First kind of the Chebyshev polynomials are extremal polynomials as stated in [11]:

$$(1-x^2)\frac{d^2}{dx^2}T_n - x\frac{d}{dx}T_n + n^2 T_n = 0 \tag{15}$$

The polynomials T_n are orthogonal over $[-1, 1]$ with weight function $(1-x^2)^{-1/2}$. Many mathematical functions are shown in [12].

Chebyshev polynomial degree of n shown as T_n:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1} &= 2xT_n(x) - T_{n-1}(x) \end{aligned} \tag{16}$$

4 Results

During the experiment Python programming language is used for numerical calculations with helper libraries such as NumPy, scikit-learn, PyTorch, SciPy, matplotlib and pandas for data analysis. Hermite and Chebyshev orthogonal

polynomials are selected as activation functions. Three different functions were calculated to represent the synthetic dataset.

$$\begin{aligned} f_1(x) &= x^2 \\ f_2(x) &= \sin 4x \cdot e^{\frac{-x}{4}} \\ f_3(x) &= (x-10)^4 + x^3 \end{aligned} \tag{17}$$

Each generated function has 1000 sample points. Datasets were split into 2 groups for training and testing purposes as 80% and 20%. Torch linear layer was used while constructing the network. Adam and SGD optimizers were selected for use and MSE was chosen as the loss function. Learning rates were 0.1 and 0.01. A list of values $[8, 16, 32, 128]$ were chosen as network unit values. Three were 3 different epoch values $[100, 500, 1000]$. The list of values $[2, 3, 4, 10, 20, 40]$ was representing polynomial degrees used by polynomial activation functions.

While Hermite polynomial's domain exists between $(-\infty, \infty)$ Chebyshev polynomials domain is $[-1, 1]$. Because of this, all of the calculated synthetic datasets were normalized to stay inside this domain before being fed into network.

MAE, max_error and MSE were 3 different error metrics which were selected.

At some points calculations failed and thrown exceptions because of the selected parameter's effect of gradients.

Calculations were conducted on a laptop running Linux operating system with a memory size of 16 Gb & Nvidia Geforce GTX 950M graphics card.

Total runtime for all calculations is 314 minutes and 28.7 seconds.

Calculations were repeated multiple times with different hyperparameters.

The list of values $[2, 3, 4, 10, 20, 40]$ was used as the degree of activation functions used in the network. However during the inspection of results it was visible that best fitting results which have minimum error value had a polynomial degree at most up to 10, and 10th-degree polynomial activation function was the highest degree that could give the minimized error through all possible parameters space, there were no results for 20th and 40th-degree solutions inside the best fitting values. During the experiment, it was observed that calculations that were conducted using Chebyshev activation functions were completing the calculations in a shorter period than the ones which had used Hermite polynomials as activation functions except for a few exceptions. One of the other findings is increasing the number of units of the network improves the fitting capability of the Hermite activation function on datasets defined as f1 and f3. For the dataset created by using a damped sine function defined as

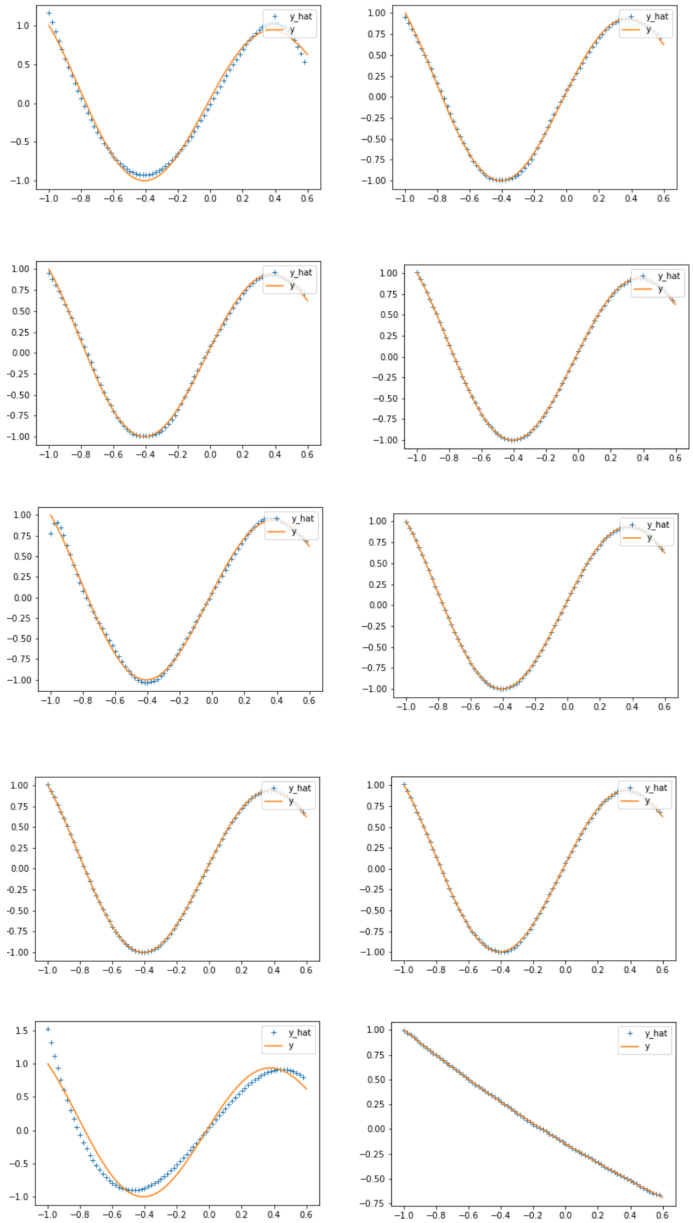

Fig. 13: Result (a)

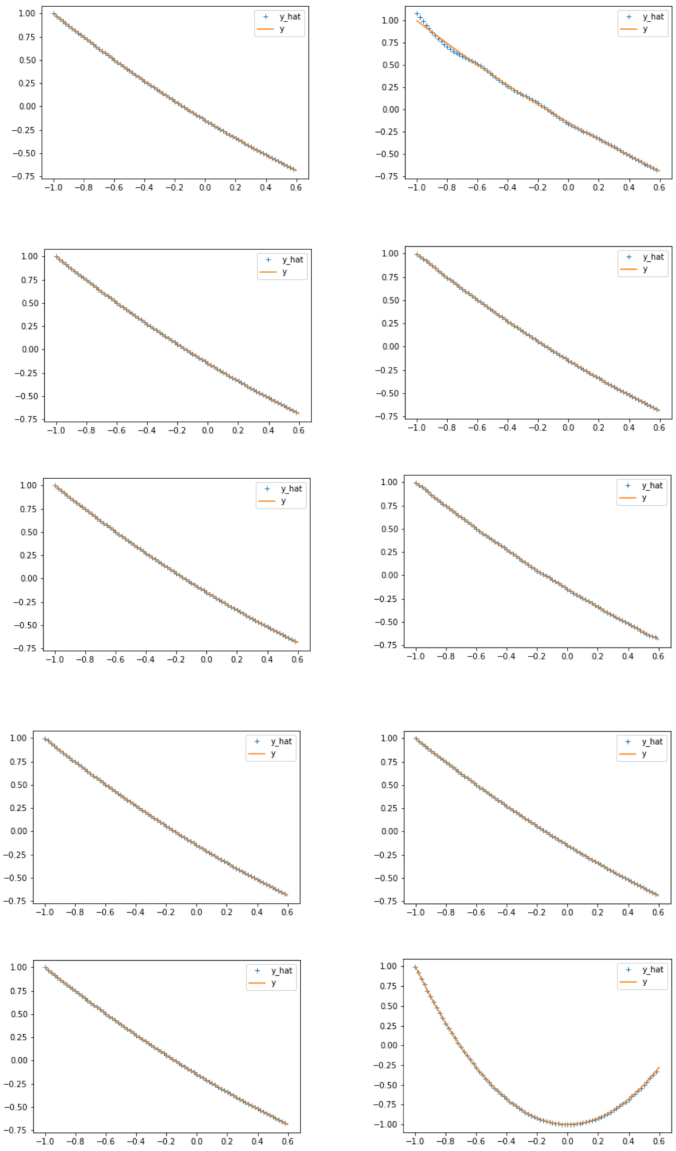

Fig. 14: Result (b)

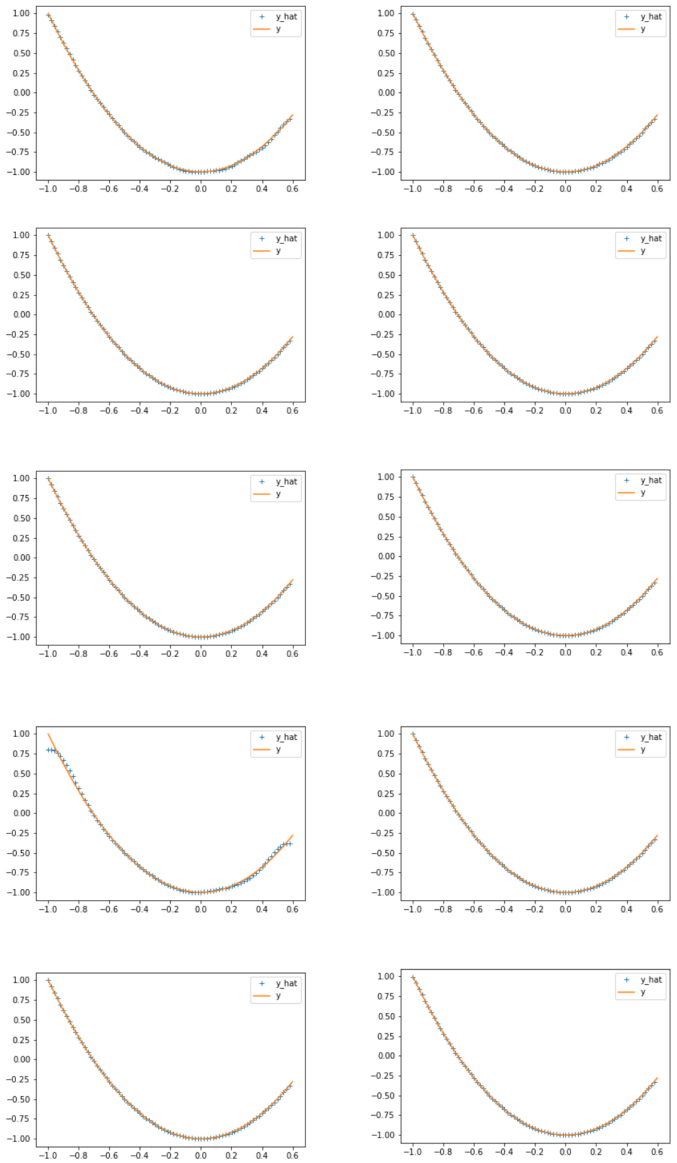

Fig. 15: Result (c)

f2, the Hermite activation function fits better than the Chebyshev activation function even starting from a low number of network units. In the damped sine dataset case or f2 case depending on the parameters Chebyshev's calculations failed or generated enormous error results.

5 Conclusion

As a result of this experiment, it is seen that both activation functions have advantages and disadvantages depending on the nature of the problem. In general, it is observed that calculations done by Chebyshev activation function takes less time, but Chebyshev can be more fragile depending on the solved problem. On the other hand, Hermite shows a more robust and generalized behavior, it is less dependent on the problem type, and it improves by necessary adjustments. Increasing the number of units used in the network increases the computation time. Chebyshev can converge with fewer iterations and a smaller number of network units. On the other hand Hermite converges with more iterations and more network units. Because of this there is a visible computation time difference between these activation functions.

References

[1] J. Wang, L. Chen, C. W. W. Ng, A New Class of Polynomial Activation Functions of Deep Learning for Precipitation Forecasting, *WSDM '22: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 1025–1035, 2022.

[2] S. Obla, X. Gong, A. Aloufi, P. Hu, D. Takabi, Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks, *IEEE Access*, 8:153098–153112, 2020.

[3] Y. Kütük, *Activation functions: Activation functions in deep learning with LaTeX applications*, Peter Lang AG, 2022.

[4] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: A comprehensive survey and benchmark, *Neurocomputing*, 503:92–108, 2022.

[5] T.-T. Lee, J.-T. Jeng, The Chebyshev-polynomials-based unified model neural networks for function approximation, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 28(6):925–935, 1998.

[6] L. Ma, K. Khorasani, Constructive feedforward neural networks using Hermite polynomial activation functions, *IEEE Transactions on Neural Networks*, 16(4):821–833, 2005.

[7] E. López-Rubio, F. Ortega-Zamorano, E. Domínguez, J. Muñoz-Pérez, Piecewise Polynomial Activation Functions for Feedforward Neural Networks, *Neural Processing Letters*, 50:121–147, 2019.

[8] A. Pinkus, Approximation theory of the MLP model in neural networks, *Acta Numerica*, 8:143–195, 1999.

[9] T. S. Chihara, *An Introduction to Orthogonal Polynomials*, Courier Corporation, 2011.

[10] V. S. Lokhande, S. Tasneeyapant, A. Venkatesh, S. N. Ravi, V. Singh, Generating Accurate Pseudo-Labels in Semi-Supervised Learning and Avoiding Overconfident Predictions via Hermite Polynomial Activations, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11435–11443 , 2020.

[11] T. J. Rivlin, *Chebyshev Polynomials*, Courier Dover Publications, 2020.

[12] M. Abramowitz, I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, U.S. Government Printing Office, 1968.