


Article

A Method for Calculating the Derivative of Activation Functions Based on Piecewise Linear Approximation

Xuan Liao ¹, Tong Zhou ², Longlong Zhang ¹, Xiang Hu ¹  and Yuanxi Peng ^{1,*}¹ State Key Laboratory of High-Performance Computing, College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China² Beijing Institute for Advanced Study, National University of Defense Technology, Beijing 100000, China

* Correspondence: pyx@nudt.edu.cn

Abstract: Nonlinear functions are widely used as activation functions in artificial neural networks, which have a great impact on the fitting ability of artificial neural networks. Due to the complexity of the activation function, the computation of the activation function and its derivative requires a lot of computing resources and time during training. In order to improve the computational efficiency of the derivatives of the activation function in the back-propagation of artificial neural networks, this paper proposes a method based on piecewise linear approximation method to calculate the derivative of the activation function. This method is hardware-friendly and universal, it can efficiently compute various nonlinear activation functions in the field of neural network hardware accelerators. In this paper, we use least squares to improve a piecewise linear approximation calculation method that can control the absolute error and get less number of segments or smaller average error, which means fewer hardware resources are required. We use this method to perform a segmented linear approximation to the original or derivative function of the activation function. Both types of activation functions are substituted into a multilayer perceptron for binary classification experiments to verify the effectiveness of the proposed method. Experimental results show that the same or even slightly higher classification accuracy can be achieved by using this method, and the computation time of the back-propagation is reduced by 4–6% compared to the direct calculation of the derivative directly from the function expression using the operator encapsulated in PyTorch. This shows that the proposed method provides an efficient solution of nonlinear activation functions for hardware acceleration of neural networks.

Keywords: activation functions; piecewise linear approximation; back-propagation

Citation: Liao, X.; Zhou, T.; Zhang, L.; Hu, X.; Peng, Y. A Method for Calculating the Derivative of Activation Functions Based on Piecewise Linear Approximation. *Electronics* **2023**, *12*, 267. <https://doi.org/10.3390/electronics12020267>

Academic Editor: Cheng-Chi Lee

Received: 16 December 2022

Revised: 30 December 2022

Accepted: 3 January 2023

Published: 4 January 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In recent years, various artificial neural networks (ANNs) accelerators have emerged, including hardware accelerations for convolutional neural networks [1–3], and transformer [4–6], which all involve the implementation of activation functions, however, these accelerators are basically oriented to the inference only, and there are only a few accelerators for training tasks, such as [7,8]. Above work either uses simple activation functions, such as ReLu, whose derivatives is also simple, or skip the calculation of the derivative of the activation function. The calculation of the derivative of the activation function is often one of the difficult points in the training task of ANNs. Most of the activation functions are complex transcendental functions, such as sigmoid, swish [9], and softplus [10], which provide nonlinear fitting capability for ANNs, and their derivative calculations involve not only derivative operations, but also complex transcendental function calculations. Therefore, it is necessary to optimize the calculation of the derivative of the activation function.

Similar to the way many accelerators implement the activation function, this paper uses a piecewise linear (PWL) approximation method, which requires only one multiply-add to calculate the derivative of the activation function. Activation function can be divided into

two types: (1) the derivative function can be represented by the original function, such as sigmoid: $y = 1/(1 + \exp(-x))$, whose derivative function $y' = y(1 - y)$; (2) the derivative function can't be expressed by the original function, such as softplus: $y = \log(1 + \exp(x))$, swish: $y = x/(1 + \exp(-x))$. For the first type of activation function, the original function value is first calculated using the PWL approximation method, and then the derivative value is calculated indirectly, meanwhile, analyze the transfer error, while for another type of activation function, the derivative function is fitted directly using a PWL approximation method. Then, we add the function obtained by approximation to the neural network in the form of a custom activation function and complete the training test task. In the following, the background knowledge of neural network training and the development of PWL approximation algorithm are briefly introduced.

The training of ANNs is done by adjusting the parameters of the hidden and output layers so that the results computed by the network are as close to the real ones as possible. The training process consists of two parts: forward propagation and backward propagation [11]. In the forward propagation process, the training data is computed through the weights, bias, and activation functions to obtain the hidden layer, and the hidden layer gets the next hidden layer through the weights, bias, and activation functions of the next level, and the output vector, usually the classification result, is finally obtained after layer-by-layer iteration.

The basic principle of back-propagation is that the loss function is first calculated based on the forward propagation output and the label, and then some optimization methods such as gradient descent are used to calculate the bias derivative of the loss function for each weight and bias by the chain rule, the effect of the weight or bias on the loss function, and finally the weights and biases are updated. According to [12], the process of calculating the bias derivatives by the chain rule is actually done by constructing the Jacobi matrix of each layer and then calculating the vector Jacobi product (VJP), where the activation function layer is a point-wise layer and its Jacobi matrix is a diagonal matrix, in practice, it is more efficient to calculate the VJP directly.

PWL is popular in the field of transcendental function approximation calculations due to its computational efficiency and memory friendliness. Many PWL algorithms have been proposed, and the core idea is the trade-off between the number of segments and accuracy, since more segments means more storage space is required. Frenzen et al. [13] studied the number of segments required for approximation calculations of various commonly used functions. The development of segmentation algorithms can be broadly summarized as follows: uniform segmentation, non-uniform segmentation, and adaptive segmentation. Initially, PWL basically used uniform segmentation, and [14] implement the approximate computation of \log functions in hardware based on uniform segmentation. Obviously, uniform segmentation has great limitations, and the error gap is large for different functions and different segments. So non-uniform segmentation is proposed, and [15] propose a non-uniform segmentation approach to approximate the fit of the \log function, and it divides the whole interval of independent variables into 15 segments and more segments near the zero point to improve the computational accuracy. This approach is still not accurate enough and cannot be extended to other functions.

Recently, the segmentation algorithm gradually tends to a general way with controlled error and fewer segments, [16] propose a method to determine the segmentation points based on the second-order derivatives, it holds that the second-order derivatives reflect the degree of concavity of the function curve, and more segments should be divided where the absolute value of the second-order derivatives is large, this work can make the L_2 error below 10^{-4} with 64 segments, however, the obvious drawback is that it needs to calculate the 2/5th power of the second order derivative and introduces an integration operation, which makes it difficult to calculate segmentation points. In [17], S.R. Chiluveru et al. use the least squares method to approximate the transcendental function and iterate over the input interval to find the interval that satisfies the error requirement, however, the method is proposed for continuous intervals and needs improvement in hardware

implementation. A general PWL method with controllable absolute error is proposed in [18], which determines the slope and intercept of the fitted line by the starting and ending points of the subinterval, and moves the line vertically to control the maximum absolute error. To the best of our knowledge, it is state of the art. On its basis, we use the least squares method, instead of simply using the subinterval start and end points, to determine the slope and intercept of the fitted straight line, which means that all points within the subinterval are taken into account. Experiments show that the improved method, in specific cases, yields a smaller average error or a smaller number of segments. We call the method PWLMMAE (Piecewise Linear Minimize Maximum Absolute Error) and will describe it in detail in the next section.

To summarize, this paper focuses on the computation of the derivative values of the activation function for back-propagation in ANNs training tasks and makes the following contributions.

1. We use least squares to improve a general, error-controlled PWL approximation method to obtain fewer segments or smaller average errors, and then extend it to the calculation of various activation functions and their derivatives.
2. We evaluate the applicability of the method in neural networks in terms of convergence speed, generalization ability, and hardware overhead.
3. We replace the derivative calculation in the neural network training task with proposed method and verified its effectiveness on a three-layer perceptron: our method reduced the backpropagation computation time by 4–6% with the same or even slightly higher classification accuracy.

2. Methods

In this section we first describe the flow of the PWLMMAE algorithm and then apply it to two typical activation functions and analyze their errors.

2.1. PWLMMAE

Similar to [19–21], the core idea of the algorithm is to determine the subinterval straight line by least squares, then calculate the maximum absolute error between the line and the real curve, and find the maximum absolute error less than the predetermined error through continuous iteration, the steps of the algorithm are as follows.

2.1.1. Input Range Discretization

Considering the hardware implementation, the input range should be discretized. For a given input interval $[M, N]$, the input x should be defined as a vector

$$x = x(1 : NUM) = M, M + \frac{1}{2^Q}, M + \frac{1}{2^Q}, \dots, N \quad (1)$$

where Q is the number of fractional bits setting in hardware and NUM is the length of the vector.

2.1.2. Minimization of MAE for a Given Width of Subinterval

The slope b and intercept a of the subinterval approximation line are first calculated using the least squares method by Equations (2) and (3), and we can use $h'(x)$ to represent the approximation line.

$$bn + a \sum_{i=j}^k x_i = \sum_{i=j}^k f(x_i) \quad (2)$$

$$b \sum_{i=j}^k x_i + a \sum_{i=j}^k x_i^2 = \sum_{i=j}^k x_i f(x_i) \quad (3)$$

where n is the number of discrete points in the subinterval, $x \in (j:k)$, $1 \leq j \leq k \leq \text{NUM}$, then, the objective function is denoted by $f(x)$, so the error vector can be expressed as Equation (4)

$$\delta = f(x(j:k)) - h'(x(j:k)) \quad (4)$$

The corresponding MAE can also be calculated as

$$\text{MAE} = \{ |\max(\delta)|, |\min(\delta)| \} \quad (5)$$

2.1.3. Segmentation Points

To obtain the maximum segmentation interval, we determine the segmentation points from right to left. Initially, we set $\text{START} = x(1)$, $\text{END} = x(\text{NUM})$, then perform the PWL method on $x(\text{START} : \text{END})$ to calculate MAE by Equation (4). if $\text{MAE} < E_c$, where E_c is a predefined error, approximation succeeds; otherwise $\text{END} = \text{END} - 1$ and repeat above step. Once approximation succeeds, we will update START and END to find the next subinterval, where $\text{MAE} < E_c$. The values of START and END record the segmentation points. This process is shown in Figure 1, and the flow chart of the whole algorithm is as Figure 2.

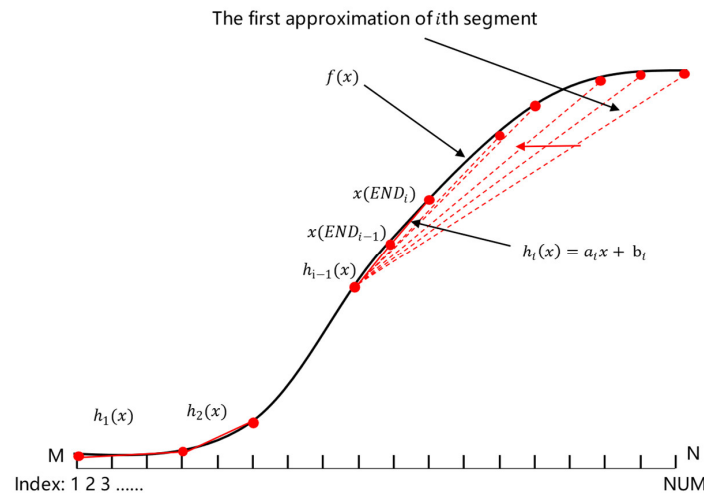


Figure 1. This figure shows the process how to modify the i th segment to meet the error requirement. The direction indicated by the arrow represents the direction of the update of the end point.

2.2. PWL Approximation to Derivatives

We choose two typical activation functions, one is sigmoid and the other is softplus, and approximate their derivatives separately setting $E_c = 0.001$.

For sigmoid, its function expression and derivative expression are as follows.

$$y = \frac{1}{1 + e^{-x}} \quad (6)$$

$$y' = y(1 - y) \quad (7)$$

We use PWLMMAE to fit its original function and then calculate its derivative by Equation (6). Here, we need to consider the problem of error transmission. When fitting the original function, we can control the absolute error below E_c , and this error will be transmitted to the derivative with Equation (7). According to the error propagation law, the error of derivative can be expressed as

$$\delta y' = (1 - 2y)\delta y \quad (8)$$

In Equation (8), δy is the original error that is smaller than E_c , and $y \in (0, 1)$, so that we can find that this calculation does not enlarge the absolute error of the derivative, $\delta y'$.

It's also smaller than E_c . The same problem will happen to the tanh function, the difference is that the error of derivative of tanh will be enlarged to twice of the original error, however its effects is marginal, and we can still control the error of its derivative within a predefined value. The reason why we do not use PWLMMAE to fit the derivatives directly is that the above approach can simplify both the calculation of the original function in forward propagation and the derivatives in backward propagation with controlled errors and no increase in hardware area.

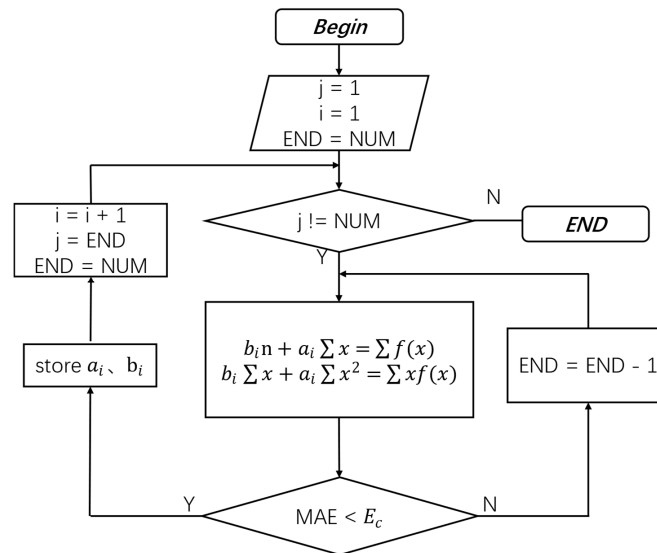


Figure 2. PWLMMAE flow chart.

For softplus, its function expression and derivative expression are as follows.

$$y = \log(1 + e^x) \quad (9)$$

$$y' = \frac{e^x}{1 + e^x} \quad (10)$$

We fit its derivative function directly using PWLMMAE, so there is no need to consider the error transfer. Figure 3 shows the approximation of the derivative of softplus and sigmoid. We can find that the max absolute error in the derivative of sigmoid is expanded by a factor of two.

3. Experiment

In this section, we first perform comparison experiments of PWL methods. Then, we perform binary classification experiments on a three-layer perceptron, in which we evaluate the suitability of the PWLMMAE algorithm for implementing derivatives of the activation function in three ways. Finally, we compare the classification accuracy and computational speed of our method with the operator encapsulated in PyTorch.

3.1. Experimental Setup

In the first part, we approximate the two typical activation functions, sigmoid and tanh, and compare with [18]. In the binary classification experiments, we use the ionosphere dataset [22]. We replaced the activation layers in the network with our own implementation of sigmoid and softplus, respectively. For comparison, we implemented two versions of each activation function, one version using the PWLMMAE and the other version using operators such as torch.exp, torch.log in PyTorch. To determine the interval, we counted the input distribution of the activation layers in the network several times, as shown in the Figure 4, which allows us to determine the input range as $(-15, 15)$, and for the very few

inputs that are not in that range, we use the nearest segment linear approximation when calculating the derivatives.

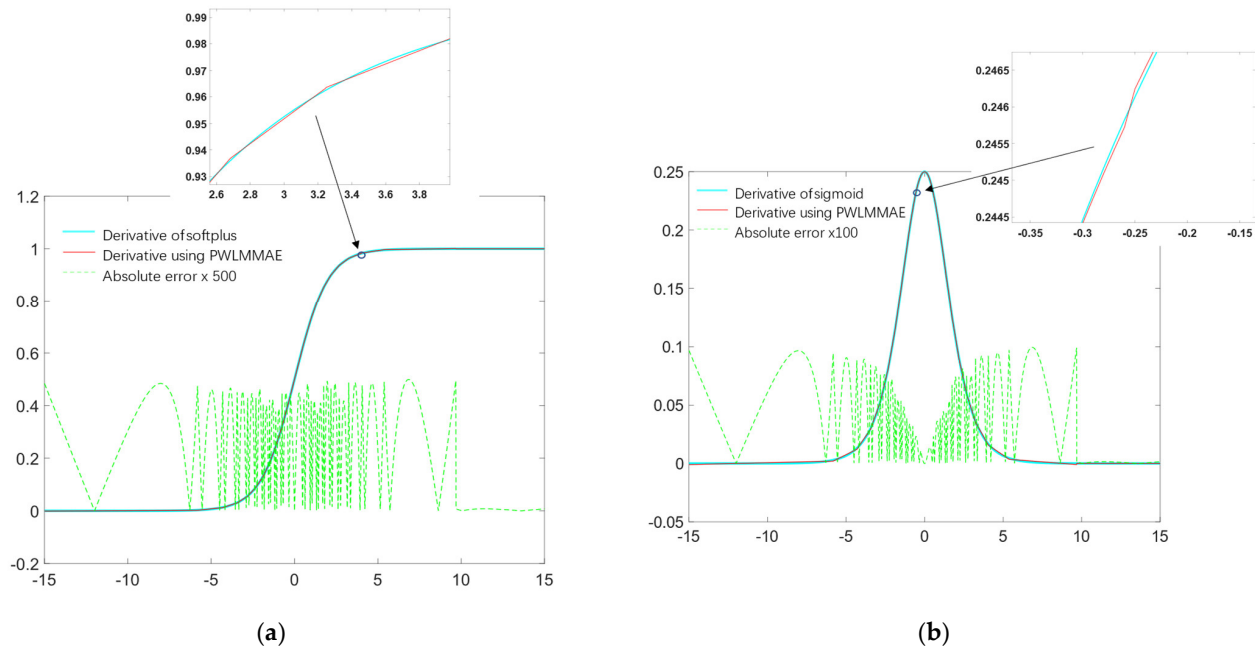


Figure 3. (a) approximation of derivative of softplus. (b) approximation of derivative of sigmoid.

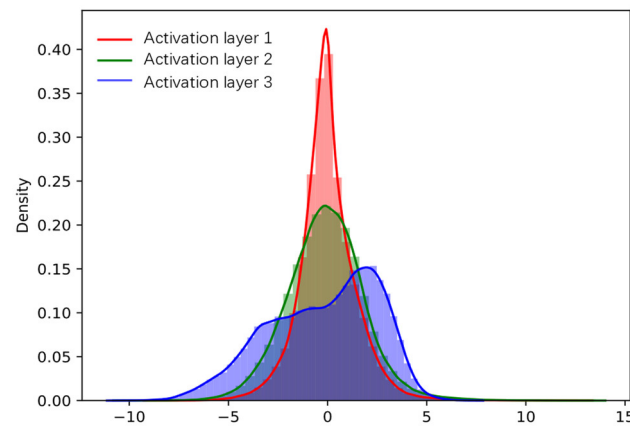


Figure 4. Distribution of inputs for each activation layer.

3.2. Segment Approximation Comparison

We conduct comparison experiments for the sigmoid function and tanh function at different maximum absolute errors, respectively. The experimental settings and results are shown in the Table 1 below. The input range of the sigmoid function is set to $(-15, 15)$, while the input range of the tanh function is set to $(-4, 4)$, beyond which the values of the two functions converge separately. The fraction bit width Q is set to 4 bits. The preset maximum absolute errors are set for two sets, $E_c = 0.001$ and $E_c = 0.0005$. As can be seen from the Table 1 for the sigmoid function, our method is able to obtain a smaller E_a (average error) with a reduced number of segments when $E_c = 0.001$, and for the tanh function, a significant reduction in the number of segments is achieved when $E_c = 0.0005$.

3.3. Evaluate PWLMMAE

According to [23], there are three criteria for selecting the implementation of the derivative of activation function: (1) speed of convergence, (2) capability of generalization, and (3) hardware overhead and speed. In the following, we will evaluate the suitability of

the PWLMMAE algorithm for implementing the derivative of the activation function in terms of these three aspects.

Table 1. Segment Approximation Comparison.

Function	Input_Range	Q	E_c	Method	E_a	NO. of Segment
sigmoid	(-15,15)	4	0.001	[18]	4.65×10^{-4}	21
				This	3.96×10^{-4}	20
			0.0005	[18]	2.61×10^{-4}	29
				This	2.16×10^{-4}	29
			0.001	[18]	5.00×10^{-4}	30
				This	4.94×10^{-4}	31
tanh	(-4,4)	4	0.0005	[18]	2.39×10^{-4}	52
				This	2.61×10^{-4}	46

3.3.1. Speed of Convergence

As can be seen from Figure 5, the loss decreases rapidly in the first 50 epoch. After the same number of epoch, the network using the PWLMMAE method has a smaller loss value. In the last 50 epochs, loss decreases slowly and tends to converge. The sigmoid, softplus functions implemented using the PLWMMMAE algorithm converge faster than sigmoid, softplus function encapsulated in PyTorch.

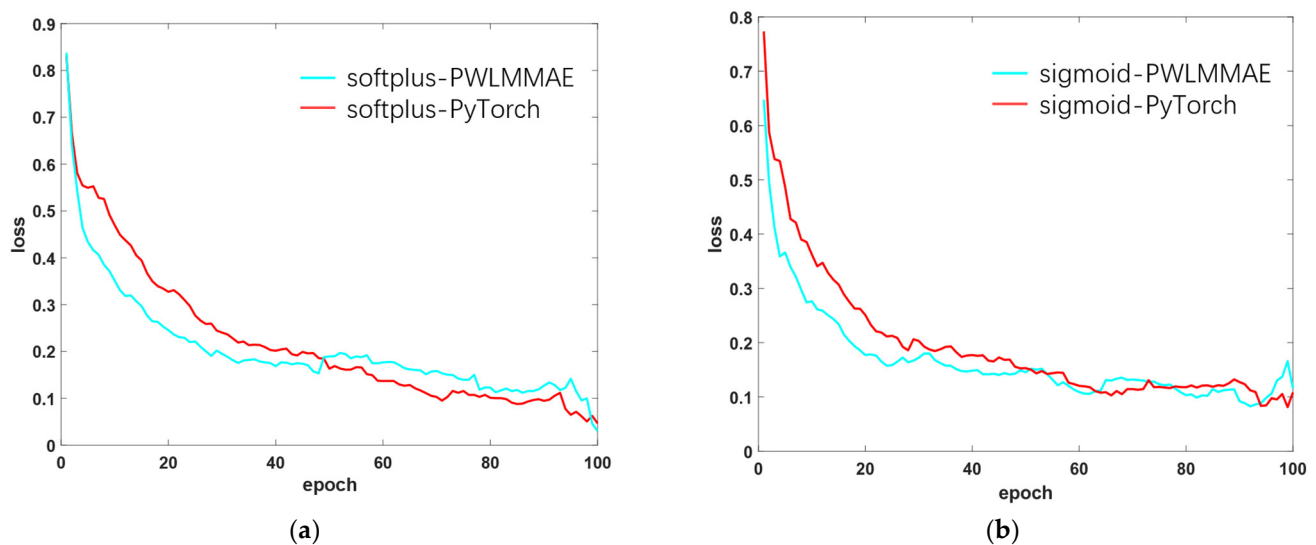


Figure 5. Comparison of the convergence speed of the activation function implemented using the PWLMMAE algorithm and the activation function encapsulated by PyTorch. (a) Comparison of the convergence speed of softplus. (b) Comparison of the convergence speed of sigmoid.

3.3.2. Capability of Generalization

We use the confusion matrix and the F-score to evaluate the model. The Tables 2 and 3 below shows the macro precision macro-p, macro recall macro-R and macro-F1 calculated from the confusion matrix of 100 training, testing. The model using this method has a very small drop in macro-F1 compared to the model using the PyTorch activation layer, which means that using our method does not significantly reduce the generalization ability of the model.

Table 2. Evaluate sigmoid.

Activation Function	Macro-P	Macro-R	Macro-F1
sigmoid-PyTorch	0.90195	0.82159	0.85989
sigmoid-PWLMMAE	0.90438	0.81338	0.85647

Table 3. Evaluate softplus.

Activation Function	Macro-P	Macro-R	Macro-F1
softplus-PyTorch	0.92694	0.79302	0.85477
softplus-PWLMMAE	0.91681	0.80705	0.85844

3.3.3. Hardware Overhead and Speed

It is obvious that only one multi-add operation is required to complete a derivative calculation using PWLMMAE. In [18], by designing the indexing circuit properly and using pipelining, it just takes one clock cycle to complete one derivative calculation and the hardware area is significantly better than other PWL methods. In addition, we study the accuracy of the input interval discretization, i.e., the bit width of the fractional part of the hardware implementation. In the hardware implementation of neural networks, aggressive reduction of the data bit width can reduce the storage and operation overhead to a great extent [24], and for PWL approximation, it can also reduce the number of segments and reduce the storage overhead. As shown in the Table 4 below, when the bit width is reduced, the corresponding predefined maximum error E_c changes to accommodate the max precision that can be represented by the bit width. For the sigmoid function, the classification accuracy of the neural network decreases when the bit width of the fractional part is reduced, while for the softplus function, this does not occur. We analyze the reason is that in the custom implementation of both activation functions, the sigmoid original function is approximated, while the derivative function of the softplus function is approximated, which means that the value of its original function is still exact.

Table 4. Effect of bit width of data on classification accuracy.

Function	Q(Bits)	E_c	Acc Average	NO. of Segment
Sigmoid	2	0.25	87.27	2
	3	0.125	88.93	3
	4	0.0625	89.82	3
Softplus	2	0.25	90.10	2
	3	0.125	90.01	3
	4	0.0625	90.19	3

3.4. Performance Comparisons

We train the network 100 times and count the average backward time during training, while verifying the results of each training on the test dataset and recording the Distribution of inputs for each activation layer classification accuracy. The Backward time refers to the time used to take the reverse derivative from the loss function and update the weights of each layer, which is called back-propagation. From the Table 5, we can see that our method can reduce the backward time by more than 4% for the sigmoid function and about 6% for the softplus function, while guaranteeing almost no impact on the classification accuracy.

Table 5. Evaluate sigmoid.

Activation Function	Accuracy	Backward Time(s)
Sigmoid-PyTorch	90.00	2.031
Sigmoid-PWLMMAE	89.82	1.935
Softplus-PyTorch	89.91	2.139
Softplus-PWLMMAE	90.19	2.005

4. Conclusions

This paper focuses on the calculation of nonlinear activation functions in ANNs. We propose a generalized, error-controlled PWL approximation method PWLMMAE using least squares, which is capable of obtaining smaller average approximation errors with

fewer segments. This method can calculate the approximate calculation of any nonlinear function quickly with less hardware resources. We explore the possibility of applying this method to neural network hardware acceleration and use it to calculate the derivative values of the activation functions in the neural network training task. We first evaluate the applicability of the method in ANNs in terms of convergence speed, generalization capability, hardware area and speed. Experimental results show that the activation function implemented using the method is almost indistinguishable from the activation function encapsulated in PyTorch and is capable of performing well. Finally, we compare the accuracy, back-propagation computation time of the activation function implemented by this method and PyTorch operator on the test dataset, and our method reduces the back-propagation computation time by 4–6% with the same or even slightly higher classification accuracy. Based on the above experiments, it can be seen that the proposed approximate calculation method is suitable for the hardware acceleration of ANNs, including reasoning and training. However, running the algorithm at the software level is limited by the indexing time when calculating the derivatives, and we have only experimented on a three-layer perceptron, future work will see us apply the method to larger-scale networks and complete the hardware implementation.

Author Contributions: Methodology, X.L.; validation, T.Z. and Y.P.; writing—original draft preparation, X.L.; writing—review and editing, X.L. and T.Z.; supervision, L.Z. and X.H. All authors have read and agreed to the published version of the manuscript.

Funding: The Opening Foundation of State Key Laboratory of High-Performance Computing, National University of Defense Technology, under Grant No. 202201-05.

Data Availability Statement: The data presented in this study are available in the text.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, Z.; Dou, Y.; Jiang, J.; Xu, J.; Li, S.; Zhou, Y.; Xu, Y. Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks. *ACM Trans. Reconfig. Technol. Syst.* **2017**, *10*, 1–23. [\[CrossRef\]](#)
2. Qiao, Y.; Shen, J.; Xiao, T.; Yang, Q.; Wen, M.; Zhang, C. FPGA-Accelerated Deep Convolutional Neural Networks for High Throughput and Energy Efficiency. *Concurr. Computat. Pract. Exper.* **2017**, *29*, e3850. [\[CrossRef\]](#)
3. Yu, Y.; Wu, C.; Zhao, T.; Wang, K.; He, L. OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks. *IEEE Trans. VLSI Syst.* **2020**, *28*, 35–47. [\[CrossRef\]](#)
4. Li, B.; Pandey, S.; Fang, H.; Lyv, Y.; Li, J.; Chen, J.; Xie, M.; Wan, L.; Liu, H.; Ding, C. FTRANS: Energy-Efficient Acceleration of Transformers Using FPGA. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, Boston, MA, USA, 10–12 August 2020.
5. Lu, S.; Wang, M.; Liang, S.; Lin, J.; Wang, Z. Hardware Accelerator for Multi-Head Attention and Position-Wise Feed-Forward in the Transformer. In Proceedings of the 2020 IEEE 33rd International System-on-Chip Conference (SOCC), Las Vegas, NV, USA, 8–11 September 2020.
6. Khan, H.; Khan, A.; Khan, Z.; Huang, L.B.; Wang, K.; He, L. NPE: An FPGA-Based Overlay Processor for Natural Language Processing. In Proceedings of the 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 28 February–2 March 2021.
7. Zhao, W.; Fu, H.; Luk, W.; Yu, T.; Wang, S.; Feng, B.; Ma, Y.; Yang, G. F-CNN: An FPGA-Based Framework for Training Convolutional Neural Networks. In Proceedings of the 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), London, UK, 6–8 July 2016.
8. Liu, Z.; Dou, Y.; Jiang, J.; Wang, Q.; Chow, P. An FPGA-Based Processor for Training Convolutional Neural Networks. In Proceedings of the 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, Australia, 11–13 December 2017.
9. Ramachandran, P.; Zoph, B.; Le, Q.V. Swish: A Self-Gated Activation Function. *arXiv* **2017**, arXiv:1710.05941v1.
10. Dugas, C.; Bengio, Y.; Bélisle, F.; Nadeau, C.; Garcia, R. Incorporating Second-Order Functional Knowledge for Better Option Pricing. In Proceedings of the 13th International Conference on Neural Information Processing Systems, Denver, CO, USA, 1 January 2000.
11. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back propagating errors. *Nature* **1986**, *323*, 533–536. [\[CrossRef\]](#)
12. Baydin, A.G.; Pearlmutter, B.A.; Radul, A.A.; Siskind, J.M. Automatic Differentiation in Machine Learning: A Survey. *J. Mach. Learn. Res.* **2015**, *18*, 153:1–153:43.

13. Frenzen, C.L.; Sasao, T.; Butler, J.T. On the Number of Segments Needed in a Piecewise Linear Approximation. *J. Comput. Appl. Math.* **2010**, *234*, 437–446. [[CrossRef](#)]
14. Gutierrez, R.; Valls, J. Low Cost Hardware Implementation of Logarithm Approximation. *IEEE Trans. VLSI Syst.* **2011**, *19*, 2326–2330. [[CrossRef](#)]
15. Kim, H.; Nam, B.-G.; Sohn, J.-H.; Woo, J.-H.; Yoo, H.-J. A 231-MHz, 2.18-MW 32-Bit Logarithmic Arithmetic Unit for Fixed-Point 3-D Graphics System. *IEEE J. Solid-State Circuits* **2006**, *41*, 2373–2381. [[CrossRef](#)]
16. Berjón, D.; Gallego, G.; Cuevas, C.; Morán, F.; García, N. Optimal Piecewise Linear Function Approximation for GPU-Based Applications. *IEEE Trans. Cybern.* **2016**, *46*, 2584–2595. [[CrossRef](#)]
17. Chiluveru, S.R.; Tripathy, M.; Chunarkar, S. A Controlled Accuracy-Based Recursive Algorithm for Approximation of Sigmoid Activation. *Natl. Acad. Sci. Lett.* **2021**, *44*, 541–544. [[CrossRef](#)]
18. Sun, H.; Luo, Y.; Ha, Y.; Shi, Y.; Gao, Y.; Shen, Q.; Pan, H. A Universal Method of Linear Approximation With Controllable Error for the Efficient Implementation of Transcendental Functions. *IEEE Trans. Circuits Syst. I* **2020**, *67*, 177–188. [[CrossRef](#)]
19. Srivastava, H.M.; Ansari, K.J.; Özger, F.; Ödemiş Özger, Z. A Link between Approximation Theory and Summability Methods via Four-Dimensional Infinite Matrices. *Mathematics* **2021**, *9*, 1895. [[CrossRef](#)]
20. Cai, Q.-B.; Ansari, K.J.; Temizer Ersoy, M.; Özger, F. Statistical Blending-Type Approximation by a Class of Operators That Includes Shape Parameters λ and α . *Mathematics* **2022**, *10*, 1149. [[CrossRef](#)]
21. Özger, F.; Aljimi, E.; Temizer Ersoy, M. Rate of Weighted Statistical Convergence for Generalized Blending-Type Bernstein-Kantorovich Operators. *Mathematics* **2022**, *10*, 2027. [[CrossRef](#)]
22. Sigillito, V.G.; Wing, S.P.; Hutton, L.V.; Baker, K.B. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Tech. Dig.* **1989**, *10*, 262–266.
23. Gironés, R.G.; Gironés, R.G.; Palero, R.C.; Boluda, J.C.; Boluda, J.C.; Cortés, A.S. FPGA Implementation of a Pipelined On-Line Backpropagation. *J. VLSI Signal Process. Syst. Signal Image Video Technol.* **2005**, *40*, 189–213. [[CrossRef](#)]
24. Horowitz, M. 1.1 Computing's Energy Problem (and What We Can Do about It). In Proceedings of the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 20–26 February 2014.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.