Evaluation of Posit Arithmetic on Machine Learning based on Approximate Exponential Functions

Hyun Woo Oh, Won Sik Jeong, and Seung Eun Lee*
Dept. of Electronic Engineering
Seoul National University of Science and Technology
Seoul, Korea
*seung.lee@seoultech.ac.kr

Abstract— **Recent advances in semiconductor technology lead to ongoing applications to adopt complex techniques based on neural networks. In line with this trend, the concept of optimizing real number arithmetic has been raised. In this paper, we evaluate the performance of the noble number system named posit on neural networks by analyzing the execution of approximate exponential functions, which is fundamental to several activation functions, with posit32 and float32. To implement the functions with posit arithmetic, we designed the software posit library consisting of basic arithmetic operations and conversion operations from/to C standard data types. The result shows that posit arithmetic reduces the average relative error rate by up to 87.12% on the exponential function.**

Keywords; posit; IEEE-754; real number arithmetic; activation functions; exponential approximation

I. INTRODUCTION

Nowadays, real number representation and arithmetic are fundamental in a variety of applications running on digital systems. Fixed-point and IEEE-754 floating-point is dominating formats to represent and calculate real numbers for decades [1]. Most of the applications that require real number arithmetic rely on one of those formats while fixed-point offers more numeric accuracy with lesser dynamic range and floating-point provides higher dynamic range at the expense of numeric precision. However, recent advances in semiconductor technology encourage modern applications to adopt more complex computing techniques which rely on real number arithmetic [2]. In consequence, the optimization issue of real number arithmetic for higher performance with fewer computation resources has been raised to the surface.

In line with this trend, posit, a novel number format for real number arithmetic, is introduced in 2017. Posit provides a higher dynamic range and higher precision on specific number intervals compared to IEEE-754 [3]. As higher precision refers to higher performance on accuracy-critical applications, applying posit can be effective.

Machine learning applications based on deep neural networks (DNN) are one of the accuracy-critical applications. In DNN, the approximate exponential function is essential because nonlinear activation functions such as sigmoid, softmax, and exponential linear unit (ELU) are built on the approximate exponential function. As these functions are often positioned and executed on the output layer [4], error in these functions directly affects the inference results on neural networks. Therefore, choosing the appropriate number system and approximate exponential to utilize the error rate and throughput by evaluating diverse cases is necessary.

In this paper, we evaluate the performance of the posit arithmetic in neural networks by comparing 32-bit posit (posit32) and single-precision floating-point (float32) through approximate exponential functions. To implement these functions on posit, we designed the software posit library for basic arithmetic operations including add, subtract, multiply, and division and convert operations from/to C standard 32-bit data types such as float, unsigned/signed integer, and double-precision floating-point (float64) to efficiently transform from the IEEE-754 based executions.

II. POSIT ARITHMETIC

Fig.1 shows the number representation of posit arithmetic. Different from IEEE-754, posit offers dynamic precision by splitting the exponent parts of the data format into two fields called regime field and exponent field. The regime field represents variable n_r, which is encoded based on the run-length method. The first bit on the regime field decides the polarity of the exponent value and the length of the regime field decides the most significant part of the exponent value. The length of the exponent field is fixed to a configurable value called *es* except when the regime field is too long to preserve the exponent field. For those situations, the exponent field is cut off from the least significant bits. The length of the fraction bit is decided by the length of the regime field. This attribute allows the fraction field to represent the mantissa value through dynamic precision, providing a higher dynamic range and higher precision on certain number intervals.

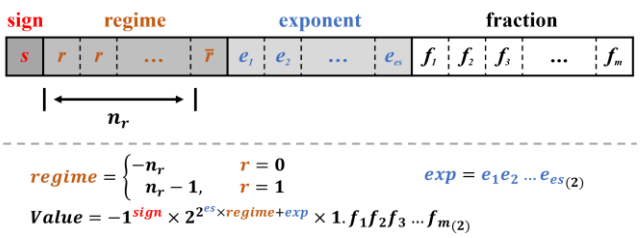

Figure 1. Real number representation on posit arithmetic

III. EXPONENTIAL APPROXIMATION

To compare the posit arithmetic with floating-point, at first, we configured the posit to appropriate settings (posit32) which provides a higher dynamic range than float32 with the same data size as follows: bitwidth parameter to thirty-two and *es* parameter to three. We set up the experimental group composed of three respective approximate exponential functions for either posit32 or float32. The approximate exponential functions are as follows: fifth-order polynomial derived from Taylor's expression, linear with two number intervals, and exponential function for float32 on free distributed libm (fdlibm), which is adopted on Newlib C standard library which is used for many embedded operating systems. The approximate functions share the following equation at the starting point to derive the output.

$$e^x = 2^{\log_2(e^x)} = 2^{x/\ln(2)} = 2^n \cdot 2^r, \quad n = \text{integer}, |r| \le 0.5 \tag{1}$$

Through the equation, the error range of the output is reduced because, as the variable n is an integer, 2^r is represented without error on the binary system. Further, obtaining 2^r on posit or floating-point can be done with only encoding exponent value on both data formats. Thus, the error rate of approximation on any range of numbers is minimized to the range of $[2^{-0.5}, 2^{0.5}]$. Differences between the functions originate from approximation methods for 2^r. Equation (2) shows the nth order exponential function approximation. To gain the 2^r value, equation (3) is acquired by equation (2), which is an equation to gain the e^x value.

$$e^x \approx 1 + x + (1/2!)x^2 + (1/3!)x^3 + (1/4!)x^4 + \dots \tag{2}$$

$$2^r = e^{r\cdot\ln(2)} \approx 1 + \ln(2)\cdot r + (1/2!)\ln(2)^2\cdot r^2 + (1/3!)\ln(2)^3\cdot r^3 + \dots \tag{3}$$

As each term of the polynomial, e.g., $(1/3!)\times\ln(2)^3$, are constant values, the terms are pre-calculated before runtime and compiled to software, reducing the amount of computation.

The exponential function on fdlibm is based on the Remez algorithm, which offers more accurate results but requires more computation than previous methods. The equation to obtain 2^r is expressed as follows:

$$R(r) = r - (p_1\cdot r^2 + p_2\cdot r^4 + p_3\cdot r^6 + p_4\cdot r^8 + p_5\cdot r^{10}) \tag{4}$$

$$2^r \approx 1 + r + r\cdot R(r) / (2 - R(r)) \tag{5}$$

IV. PERFORMANCE EVALUATION

We evaluated the posit arithmetic by comparing the results of the approximate exponential functions on both posit32 and float32. The control group is the result of the exponential function with float64 input included on fdlibm.

TABLE I. BENCHMARKS RESULTS WITH EXPONENTIAL FUNCTIONS

Approximation method	Average relative error (%)		Average time per one execution (ns)	
	posit32	*float32*	*posit32*	*float32*
5th order Taylor series	7.07×10^{-6}	8.57×10^{-6}	473.76	35.61
Remez Algorithm	2.58×10^{-7}	2.00×10^{-6}	580.15	36.26
2-interval linear	6.78	6.78	240.40	35.34

Compiled with MSVC and executed on Windows 10 running on Ryzen 2700 processor.

Table 1 presents the average relative error ratio and execution time per one iteration of the approximate exponential functions. The overall result shows that when applying the same approximation method, the relative error on exponential function using posit32 is reduced compared to the function using float32. In the case of 5th order polynomial, applying posit refers to a 17.49% reduced average relative error rate. Additionally, applying posit on the standard library function offers an 87.12% reduced error rate. However, according to the benchmark, the average execution time on posit-based functions has much longer than on floating-point-based functions. These results originate from the lack of hardware acceleration support for posit arithmetic in contrast to floating-point. Therefore, hardware acceleration should be preceded to adopt the posit arithmetic for a wide range of applications.

V. CONCLUSION

In this paper, we evaluate the posit arithmetic by analyzing the results of three approximate exponential functions with posit32 and float32 data types. The result presents that replacing the IEEE-754 to posit achieves higher performance but lacks the throughput because of the lack of hardware acceleration support. We are planning to design the hardware accelerator for posit arithmetic in the near future.

ACKNOWLEDGMENT

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2022-RS-2022-00156295) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation)

REFERENCES

[1] H. Saadat, H. Javaid, and S. Parameswaran, "Approximate Integer and Floating-Point Dividers with Near-Zero Error Bias," 56th ACM/IEEE Design Automation Conference, 2019, pp. 1-6.

[2] A. Guntoro, C. D. L. Parram F. Merchant, F. D. Dinechine, J. L. Gustafson, M. Langhammer, R. Leupers, and S. Nambiar, "Next Generation Arithmetic for Edge Computing," Design, Automation & Test in Europe Conference & Exhibition, 2020, pp. 1357-1365.

[3] J. L. Gustafson and I. T. Yonemoto, "Beating Floating Point at its Own Game: Posit Arithmetic," Supercomputing Frontiers and Innovations, vol. 4, no. 2, 2017, pp.71–86.

[4] C. Nwankpa, W. ljomah, A. Gachagan, S. Marshall, "Activation functions: comparison of trends in practice and research for deep learning," 2nd International Conference on Computational Sciences and Technology, 2021, pp. 124–133.