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

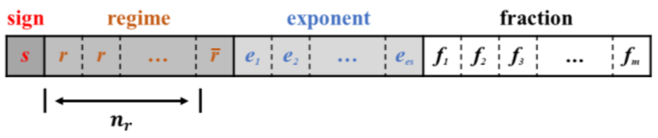
Hyun Woo Oh, Won Sik Jeong and Seung Eun Lee*
Dept. of Electronic Engineering
Seoul National University of Science and Technology

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.

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 nr, 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.



$$regime = \begin{cases} -n_r & r = 0 \\ n_r - 1, & r = 1 \end{cases} \qquad exp = e_1 e_2 \ldots e_{es(2)}$$

$$Value = -1^{sign} \times 2^{2^{es} \times regime + exp} \times 1.f_1 f_2 f_3 \ldots f_{m(2)}$$

[Real number representation on posit arithmetic]

Exponential Approximation

The approximate exponential functions we evaluated 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, used for many embedded operating systems. The approximate functions share the following equation at the starting point to derive the output.

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

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 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. The following equations represents the differences between approximation methods.

n-th order polynomial

$$e^x \approx 1 + x + (1/2!)x^2 + (1/3!)x^3 + (1/4!)x^4 + \ldots$$

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

Remez' algorithm

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

$$2^r \approx 1 + r + r \cdot R(r) / (2 - R(r))$$

Performance Evaluation

We evaluated the posit arithmetic by comparing the results of the approximate exponential functions on both posit32 and float32. 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**.

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.

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) .