A Novel Posit-based Fast Approximation of ELU Activation Function for Deep Neural Networks

Marco Cococcioni[1], Federico Rossi[1], Emanuele Ruffaldi[2], and Sergio Saponara[1]

***Abstract*— Nowadays, real-time applications are exploiting DNNs more and more for computer vision and image recognition tasks. Such kind of applications are posing strict constraints in terms of both fast and efficient information representation and processing. New formats for representing real numbers have been proposed and among them the Posit format appears to be very promising, providing means to implement fast approximated version of widely used activation functions in DNNs. Moreover, information processing performance are continuously improved thanks to advanced vectorized SIMD (single-instruction multiple-data) processor architectures and instructions like ARM SVE (Scalable Vector Extension). This paper explores both approaches (Posit-based implementation of activation functions and vectorized SIMD processor architectures) to obtain faster DNNs. The two proposed techniques are able to speed up both DNN training and inference steps.**

I. Introduction

Nowadays, Deep neural networks (DNNs) are being employed as a pervasive tool to process data and effort is being put by both industry and academia. The most active thread is bringing real-time DNNs performance at the lowest cost possible in terms of power and resource consumption.

An important emerging industry trend in this sense is the progressively reduction of DNNs complexity, reducing the information representation bits trying to avoid complex high-precision arithmetic (e.g. double precision, 64-bit arithmetic). Some formats have already been proposed like Google's BFLOAT16 (i.e. a revised Float16 representation embedded in Tensor Processing Units) and Intel's Flexpoint [1, 2]. NVIDIA has put effort in the transprecision neural network training field in its latest GPU architectures, enabling the use of INT32 down to INT4 integral types alongside with Float32 single-precision type [3, 4]. Furthermore, one of the most promising alternative representation for low-precision real arithmetic is the Posit number system [5]–[8] (see details in Section II). When trying to address the bottlenecks in training and evaluation of DNNs we need to take into account two important components. One is the massive use of convolution, pooling and small matrix-vector product operations that can be more or less brought back to accelerating vector operations. The other less impactful, yet meaningful to address, is the enormous use of non-linear activation functions nearly after each layer of a DNN.

[1]University of Pisa, DII sergio.saponara@unipi.it, federico.rossi@ing.unipi.it, m.cococcioni@gmail.com
[2]MMI spa, Via del Paduletto 10A, 56011, Calci (PI), Italy emanuele.ruffaldi@mmimicro.com

On the activation function side the use of non-linear operator is mandatory to offer enough complexity to let the neural network learn. The most commonly used non-linear operators are Sigmoid, Tanh (Hyperbolic tangent), ReLU (Rectified Linear Unit) and ELU (Extended Linear Unit) [9, 10]. When choosing activation functions, both data distribution and information representation must be taken into account. In particular, the possibility offered by Posits of efficient and hardware-friendly non-linear activation functions has to be investigated in order to provide fast approximation of commonly used non-linear operators.

On the acceleration of matrix and vector operation side a lot of work has been done using Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) ability to massively parallelize operations on vectors and matrices, but with a great power cost. On the general purpose CPU side exploiting vectorization levels offered by processors is critical. Both Intel [11] and ARM [12] provide specific compilers that offer a starting level of vectorization called auto-vectorization, i.e. automatic loop unrolling followed by the generation of SIMD (single instruction-multiple data) instructions at compilation time. Thanks to this approach, the SIMD instructions operate on multiple data elements at the same time, thus increasing the efficiency of the loop execution. Moreover, both Intel and ARM offer a set of high-level instruction (e.g. C/C++ directives) called intrinsics (i.e. AVX/2 or SSE for Intel [13] and SVE/2 or NEON for ARMv8 [14]). These instructions allow the developer to explicitly instrument low-level vectorization engines with a high-level interface.

II. Posit format and arithmetic

The novel Posit format has been proposed by John L. Gustafson in [5]. The format is a fixed length encoding configurable in the total number of bits $nbits$ and the number of exponent bits $esbits$. We identify a Posit with $nbits$ and $esbits$ as *Posit(nbits,esbits)* (e.g. Posit16,0). It is composed by four fields:

- Sign field (1-bit, ●)
- Regime field (variable length, ●)
- Exponent field (maximum length of $ebsits$, ●). This field can be shorter or even missing at all, for some representations, even when $ebsits > 0$
- Fraction field (variable length, ●): can be missing too.

Among the other fields, the regime one is particularly interesting. The bits composing that field are discovered at

run-time as a bit-string composed only by 0s or 1s and terminated, respectively, by a single 1 or 0. Then the value of the regime field is dictated by the number of leading 0s or 1s. Given a Posits on $nbits, esbits$, represented by the integer l and let e and f be respectively the exponent and fraction values, the real number r represented by that encoding is:

$$r = \begin{cases} 0, \text{ if } l = 0 \\ \text{NaN, if } l = -2^{(nbits-1)} \\ sign(l) \times useed^k \cdot 2^e \cdot (1+f), \text{ otherwise} \end{cases}$$

where $useed = 2^{2^{esbits}}$ and k is the value dictated by the regime bits. Decoding a Posit presents interesting aspects:

- When the sign bit is 1 then the remaining bits are complemented before decoding, removing the need for a redundant representation of a negative 0.
- The value k identified by the regime bits acts as a super-exponent that scales the value of $useed$. If the bit-string is composed by 0s the value of k will be negative. Note that when we are dealing with consecutive 1s, the value of k is one less than the number of equal 1s, in order to be able to represent the value 0.

A. Projective reals

Unlike real numbers, Posits map to a circle called *Posit ring*. If we split the ring into its four quadrants we can detect two important regions in Posit arithmetic:

- The $[-1, 1]$ interval: this interval occupies the bottom half of the ring, meaning that half of a Posit range concentrates in this interval.
- The $(1, \inf)$ and $(-\inf, -1)$ intervals: these intervals occupy a quarter of ring each, meaning that there is less decimal accuracy for numbers outside the $[-1, 1]$ interval.

B. The `cppPosit` library

`cppPosit` is a Posit arithmetic software library developed at University of Pisa. It exploits some of modern C++ features, like templatization, to provide a flexible and interoperable way to handle Posit numbers. It is designed in order to decouple the front-end Posit packed representation and the back-end approach to mathematical operations.

An important aspect of the cppPosit library is the presence of four different operational levels, from L1 to L4, that correspond to different efficiency of operations on Posits. Level 1 (L1) operations are simply bit manipulations of Posit representation that can be performed at the cost of an integer ALU operation. Managing to design L1 activation functions in DNNs is crucial to speedup activation layers when using Posits. Table I shows some important L1 operations implemented in cppPosit. Level 2 (L2) operations require the extraction of Posit fields without further computations. The cost is determined by the format encoding/decoding operations. Level 3 (L3) operations require the unpacked Posit version to be built thus including full computation of regime and exponent as added cost with respect to L2. Level 4 (L4) operations require conversion to Float format, exploiting either software or hardware back-ends.

TABLE I
MOST INTERESTING L1 FUNCTIONS IMPLEMENTED IN CPPPOSIT AND THEIR REQUIREMENTS TO BE APPLIED ON THE ARGUMENT x

Operation	Approximated	Requirements
$2 \cdot x$	no	*esbits=0*
$x/2$	no	*esbits=0*
$1/x$	yes	none
$1 - x$	no	*esbits=0,* $x \in [0, 1]$
FastSigmoid(x) [5]	yes	*esbits=0*
FastTanh(x) [6]	yes	*esbits=0*

III. THE EXTENDED LINEAR UNIT (ELU)

As reported in Table I, both Sigmoid and Tanh activation functions have an approximated fast version. The applicability of those activation functions in large DNNs may result in the well-known phenomenon of vanishing gradients. The ReLU activation function:

$$\text{ReLU}(x) = \begin{cases} 0, \text{ if } x \leq 0 \\ x \text{ otherwhise} \end{cases}$$

has been introduced to cope with this kind of phenomenon, having an unbounded co-domain in the positive x-axis.

$$\text{ELU}(x) = \begin{cases} e^x - 1, \text{ if } x \leq 0 \\ x \text{ otherwhise} \end{cases}$$

As pointed out in [15], if the ELU is scaled by a predetermined parameter, it applies a normalization across the layers of the networks, without the need of additional normalization layers (e.g. batch normalization layer).

A. *Fast approximation:* FastELU

In order to build a L1 approximation of the ELU function we must focus on the negative x-axis, since the first quadrant is simply an identity function. If we look at the Sigmoid function expression $\text{Sigmoid}(x) = \frac{1}{e^{-x}+1}$ we can manipulate it with some algebraic steps:

$$\begin{aligned} &(1) && \text{Sigmoid}(-x) \\ &(2) && 1/\text{Sigmoid}(-x) \\ &(3) && 1/(2 \cdot \text{Sigmoid}(-x)) \\ &(4) && 1/(2 \cdot \text{Sigmoid}(-x)) - 1 \\ &(5) && 2 \cdot [1/(2 \cdot \text{Sigmoid}(-x)) - 1] \end{aligned}$$

If we substitute back the sigmoid expression in it we get the ELU expression for negative values of the argument :

$$2 \cdot \frac{e^x + 1}{2} - 2 = e^x - 1.$$

Referring to Table I we can prove that this expression is an L1 one. Step (1) is L1 for $esbits = 0$ while Step (2) is always L1. Division by 2 at step (3) is L1 for $esbits = 0$. Step (4) is L1 since the previous step produce a result in $[0, 1]$. Step (5) is again L1 for $esbits = 0$.

TABLE II

VECTORIZATION COMPARISON RESULTS. THE BENCHMARK IS EXECUTED USING THE GTRSB DATASET DESCRIBED BEFORE.

SVE Vector length	SVE instructions (%)	Processing time (s)
None	-	41.9
256-bit	15	21.89
512-bit	11	12.56
1024-bit	9	8.17

TABLE III

COMPARISON USING POSITS FOR THE MNIST DATASET FOR THREE DIFFERENT ACTIVATION FUNCTIONS: FAST APPROXIMATED VERSION OF ELU (FASTELU), EXACT ELU AND RELU. ACCURACY OF THE NEURAL NETWORK AND MEAN SAMPLE INFERENCE TIME ARE REPORTED.

Activation	FastELU (this paper)		ELU		ReLU	
	Acc.(%)	Time (ms)	%	ms	%	ms
SoftFloat32	-	-	98.6	8.8	96.0	6.3
Posit16,0	98.54	3.2	98.6	3.9	96.0	2.0
Posit14,0	98.53	2.4	98.6	3.1	96.0	2.0
Posit12,0	98.5	2.3	98.6	3.1	96.0	2.0
Posit10,0	98.39	2.3	98.5	3.0	96.0	1.9
Posit8,0	91.14	2.2	90.1	3.0	88.4	1.9

IV. EXPERIMENTAL RESULTS

The benchmark used for experimental analysis is an image classification task on two different data sets: MNIST and GTRSB (German Traffic Road Sign Benchmark). The LeNet-5 deep neural network model [16] has been used during the experimental phase. For vectorization comparison we chose ARM SVE since it allows changing SIMD vector sizes at runtime without the need of compiling the code again (unlike Intel SIMD engine). The benchmark is compiled both with and without ARM SVE support to assess the performance increase, using the `armclang++ 19.2` compiler. Table II shows performance comparison for the GTRSB dataset benchmark executed in the ARMv8 (AARCH64) instruction emulator with different levels of vectorization. As reported, when increasing the lenght of SVE registers the number of SVE instructions decreases, thus the processing time for the benchmark decreases as well, showing the effectiveness of the vectorization.

V. RESULTS ON FASTELU USING POSITS

Table III and IV show performance comparison in the two datasets between both different activation functions and underlying information representation. As reported therein, Float32 accuracy are easily matched by Posits with 16 down to 10 bits, and, in particular, for GTRSB similar performance are obtained even with a Posit8,0. According to these results the adoption of Posit and ELU can lead to nearly the same processing accuracy of Float32 but with a remarkable reduction, up to a factor of 4, of the data storage.

VI. CONCLUSIONS

In this work we have introduced a fast way to approximate the well-known ELU activation function in DNNs, when using the novel Posit format for representing the reals, instead of classic IEEE-754 Floats. Then we have reported the preliminary results on an activity carried out within the H2020 European Processor Initative (EPI) project, namely the exploitation of autovectorization and other vectorization techniques in ARMv8 processors, again with the aim to speed up DNN on this architecture. The achieved results show that vectorization has a positive effect on network processing time. Moreover, combining the use of Posits and of the ELU activation functions, DNN computation can achieve the same accuracy levels of IEEE-754 Floats but with a reduction up to a factor of 4 of data storage and transfer complexity. Future work will include the development of hardware accelerators for Posit operations.

TABLE IV

COMPARISON USING POSITS FOR THE GTRSB DATASET (SEE TABLE III)

Activation	FastELU (this paper)		ELU		ReLU	
	Acc.(%)	Time (ms)	%	ms	%	ms
SoftFloat32	-	-	94.2	15.86	92.7	8.2
Posit16,0	94.0	5.8	94.2	6.37	92.7	5.0
Posit14,0	94.0	4.6	94.2	5.21	92.7	4.3
Posit12,0	94.0	4.6	94.2	5.08	92.7	4.3
Posit10,0	94.0	4.6	94.2	5.0	92.7	4.2
Posit8,0	92.0	4.6	91.8	5.0	86.8	4.0

ACKNOWLEDGEMENTS

Work funded by the H2020 European Processor Initiative project.

REFERENCES

[1] U. Köster *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in Neural Inf. Processing Systems*, 2017, pp. 1742–1752.

[2] V. Popescu *et al.*, "Flexpoint: Predictive numerics for deep learning," in *2018 IEEE 25th Symp. on Comp. Arith. (ARITH)*, June 2018.

[3] A. Malossi *et al.*, "The transprecision computing paradigm: Concept, design, and applications," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1105–1110.

[4] "NVIDIA Turing GPU Architecture, graphics reinvented," https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, pp. 1–80, 2018.

[5] J. L. Gustafson and I. T. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[6] M. Cococcioni *et al.*, "A fast approximation of the hyperbolic tangent when using posit numbers and its application to deep neural networks," *Springer LNEE vol.627*, 2020.

[7] ——, "Novel arithmetics to accelerate machine learning classifiers in autonomous driving applications," *26th IEEE Int. Conf. on Electronics Circuits and Systems*, 2019.

[8] ——, "Exploiting posit arithmetic for deep neural networks in autonomous driving applications," *IEEE Automotive 2018*, 2018.

[9] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on MNIST classification task," *CoRR*, vol. abs/1804.02763, 2018.

[10] V. Nair *et al.*, "Rectified linear units improve restricted boltzmann machines," in *27th Int. Conf. on Machine Learning*, 2010.

[11] "Improve performance with vectorization," https://software.intel.com/en-us/articles/improve-performance-with-vectorization, 2016.

[12] "ARM Automatic Vectorization," http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dht0002a/ch01s04s03.html, 2009.

[13] "Intel intrinsics guide," https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[14] "ARM HPC tools for SVE," https://developer.arm.com/tools-and-software/server-and-hpc/arm-architecture-tools/documentation/introducing-scalable-vector-extension-sve.

[15] G. Klambauer *et al.*, "Self-normalizing neural networks," in *Advances in Neural Information Processing Systems 30*, I. Guyon *et al.*, Eds. Curran Associates, Inc., 2017, pp. 971–980.

[16] Y. Lecun *et al.*, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.