Journal of Signal Processing, Vol.26, No.5, pp.131-140, September 2022

PAPER

Auto-Differentiated Fixed Point Notation on Low-Powered Hardware Acceleration

Robert Nsinga, Stephen Karungaru and Kenji Terada

Graduate School of Advanced Technology and Science, Tokushima University
2-1 Minamijosanjima-cho, Tokushima-shi, Tokushima 770-8506, Japan
E-mail: st_gakmuk@tokushima-u.ac.jp, karungaru@tokushima-u.ac.jp

Abstract **Using less electric power or speeding up processing is catching the interests of researchers in deep learning. Quantization has offered distillation mechanisms that substitute floating numbers for integers, but little has been suggested about the floating numbers themselves. The use of Q-format notation reduces computational overheads that frees resources for the introduction of more operations. Our experiments, conditioned on varying regimes, introduce automatic differentiation on algorithms like the fast Fourier transforms and Winograd minimal filtering to reduce computational complexity (expressed in total number of MACs) and suggest a path towards the assistive intelligence concept. Empirical results show that, under specific heuristics, the Q-format number notation can overcome the shortfalls of floating numbers, especially for embedded systems. Further benchmarks like the *FPBench* standard give more details by comparing our proposals with common deep learning operations.**

Keywords: embedded system, IEEE 754-2008 floating-point, digital signal processor, Q format notation

1. Introduction

In recent years deep learning has emerged considerably and cemented itself as a part of the daily discourse on scientific progress and cutting-edge technology. Multiple advances were registered in many other fields that chose to use deep learning methods in new or existing computer-assisted practices. Deep learning as a problem-solving scheme has therefore gained consideration. Along with the rapid rise, expensive computer configurations and intimidating verbiage, the pervasive sentiment claiming computers to be sentient made it hard to join the field of deep learning. This report reviews the historic perspective under the assistive intelligence concept and suggests solutions and available alternatives.

The true potential of computer systems lies within two essential capabilities: to automate given tasks and to help make informed decisions. Historically, computers were considered advanced calculators that carried human tasks with better precision and better speed. This notion has evolved far beyond the initial conditions. Communication would also be considered, but it falls outside the scope of this work. From the beginning of the modern computing era these two capabilities have been harnessed, improved, and applied to the workplace, transportation, businesses industries, science, space, personal, home needs and many other lifestyles.

Most machine learning models use data that has been prepared in advance by a human. The human-computer interaction has evolved considerably from the early days of modern computing. Human-in-the-loop systems, or HITL systems, allow both sides to interact continuously to automate tasks or to take decisions. Human supervision is key in this arrangement as it provides a well-defined interface for accountability or other inquiries.

Predictions and estimates generated by a computer are not always 100% accurate as their "understanding" of the data used in the process is largely built from primitive operations such as summation and multiplication of floating-point numbers, known to be error prone [1]. Human interference is meant to provide feedback for the computer to adjust its accuracy, which improves the computer's representation of the information, resulting in better predictions. In this context the "understanding" of the data and the representation of information could be used interchangeably.

1.1 Justification for the research

The nascent field of deep learning is already regarded as a powerful problem-solving technique that any individual should consider. We propose targeting embedded systems for various social-economic reasons, but this research focuses on addressing some of the limitations caused by IEEE 754-2008 floating-points computations.

The motivation behind targeting low-powered devices has long been argued even though the most cited applications of deep learning are utilitarian, and thus demand to be deployed closer to their intended users, on their personal computers, cell phones, or cameras, for instance. Deep learning has hindered the motivation for low-powered devices by increasing the computation complexity even further, giving rise to alternative, after-the-fact options like distillation, model pruning and quantization [2] [3].

Floating numbers are a good choice to avoid precision decay over multiple rounds of operations. For this reason, the deep learning community has given preference to

expensive computer systems for training with floating numbers, and the resulting models are converted to integers format for smaller form factors like embedded systems. However, research has shown that replacing floating numbers with integers does not deal a heavy toll on the accuracy of a model but instead improves its storage and access speed [4].

Deep learning has shown potential in other areas like healthcare, home consumer products, and the creative industry. There is growing interest that is pushed back by the relatively high barrier of entry caused by expensive computing requirements and complex verbiage. Transfer learning has provided some improvement, but more efforts are required to allow anyone to contribute new ideas from scratch, free from any prior limitations or cumbersome requirements.

1.2 Overview of experiments

We formulate and analyze a configuration based on a mix-and-match of existing solutions in deep learning. We are interested in testing embedded systems capabilities when applying these solutions. Our approach applies to the mathematical operations at the core of computations.

Specifically, we modify arithmetic operations in existing software algorithms to manipulate a different type of number notation to assess whether models can be trained without precision loss on operations like convolution or automatic differentiation and challenge the boundaries of power and latency for embedded systems.

In general, ongoing contributions can be split into the hardware and software distinctions where each proposes specialized methods. A third distinction that is a hybrid software-hardware mix tries to challenge the shortfalls in linking innovations in hardware and the rapidly changing software. This work is on the latter distinction, using domain-specific accelerators and software.

For our experimentations, hardware and software considerations include existing proprietary and open-source devices and libraries.

1.3 Scope and key assumptions

The paradigm shift introduced with the popularization of machine learning, especially deep learning, has reaffirmed the importance of human supervision, especially in the early stages of experiments.

Because we are interested in the limitations of floating numbers on small form factors like embedded systems, we outline key heuristics or conditions. First and foremost, we are not interested in any type of dataset, nor are we measuring the accuracy or overall effectiveness of a deep learning model. Instead, we solely consider computations which manipulate floating numbers and analyze their performance against other alternatives.

Posit arithmetic also falls out of scope in this report simply because the energy requirements do not improve on IEEE 754-2008 floating-point number arithmetic [5].

Metrics like accuracy or precision are used with regards to digital arithmetic operations. Performance and latency are measured on the computational instructions carried by a host processing unit. Performance represents the computational effort of a given task, while latency is the time to perform.

The fundamental basis on which tensor calculus is implemented digitally remains open to experimentation. Automatic differentiation is also applied at the code level to take advantage of the benefits this approach may have on small form factors.

This work aims to contribute efforts to the use of low-powered devices for deep learning research. Lastly, another scope limiting factor is the use of a digital signal processing (DSP) accelerator for domain-specific computations, namely convolutions.

1.4 Outline of the paper

This paper is organized into 4 parts. The introduction gives an overview of the ecosystem and context necessary for this research. The research problem, justification, scope, and key assumptions are also presented in the first part. The second part presents the literature for the research problem. Here we review similar works and their key contributions, paying attention to the best results. The third part describes the methodology used. The tools necessary for our experiments are highlighted before the fourth and last part closes with our results, conclusion, and recommendations.

2. Literature Review

By nature, computers have limited capacity. There is only so much they can do. However, the limitations are less and less of a concern nowadays. In the early days of modern computing, algorithms were programmed with the same procedural logic used in manual calculations. Floating numbers are arranged in large multi-dimensional matrices called tensors. Tensor operations are computationally expensive, and more so on embedded devices.

In digital signal processing, a Multiply-Accumulate Operation (MAC) is an instruction execution step that computes the product of two scalars and adds the result to a summation, or accumulator. A single DSP core can handle up to 50,000 multiply-accumulate operations per second (MACS) due to its relatively low wattage. Typically, this is done fast for integers but not for floating-point numbers.

The 2008 revision of IEEE-754 that improves digital floating-point arithmetic, or IEEE 754-2008, specifies a novel instruction, called Fused Multiply-Add (FMA), that fixes the inherent properties of digital floating-point arithmetic of non-associativity and non-distributivity by a single rounding rather than two as is the case for common DSPs [6]. Certain operations with higher order polynomials could lead to errors unless careful considerations are made about the memory space allotted to the result of the operations [7]. FMA is also great for the software implementation of arithmetic division and square

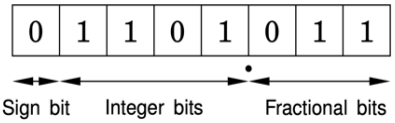

Fig. 1 In fixed-point arithmetic the position of the point is "fixed", i.e., decided in advance

root, but it is relatively inadequate for low-powered devices [8].

2.1 Low-powered computing accelerators

In recent years, hardware accelerated neural networks have been proposed in lieu of their traditional CPU focused alternatives for improved processing speeds [9]. Different types of existing hardware were repurposed, and new types of hardware were designed specifically to support acceleration for the most critical computations found on many neural networks [10]. GPUs are the most common and consumer-ready, plug-and-play type of accelerator [11]. Given a supporting architecture, and a pre-installed special purpose kernel library like the proprietary CUDA or the open-source OpenCL, a GPU can significantly improve computational speeds.

This is not exclusive to GPUs, however, as industry efforts continue to provide drivers for other types of processors like DSP. A notable innovation in high-performance computing is the TPU, a type of ASIC (application-specific integrated circuit) accelerator designed purposefully for neural networks on large data centers. The motivation for TPUs grew out of the need to address limitations on consumer devices like smartphones that could not process computations fast enough, therefore posting requests to a server and returning the results. For some of these special-purpose hardware, a dedicated matrix calculator is implemented at the hardware level instead of the common software level. TPUs, for instance, have a systolic array configuration to reduce delay.

Matrix multiplication continues to keep researchers busy to date. It helps a lot with tensor calculus. By defining a matrix with coordinates corresponding to rows and columns in a grid-like pattern, each element of the matrix can be found wherever a row and column intersect. It would take at least n3 operations to multiply a n*n square matrix. Over the years researchers have attempted to optimize this task by rewriting the algorithm to computer specifications. In 1969 the best algorithm used $n^{2.8074}$ operations. In 2014 François Le Gall achieved $n^{2.3728639}$, and as of 2020 $n^{2.3728596}$ was achieved [12].

2.2 Execution steps in accelerated computing

Once a given deep learning task has been compiled and scheduled for execution it must be sent to the correct hardware accelerator that will carry said execution and return its results [13]. This is also the time to parallelize. Specific libraries help in this regard; like CUDA from NVIDIA, or OpenCL from the Khronos Group, an open-source consortium of industry and academia leaders, of which NVIDIA is a part of. The reason NVIDIA is mentioned is because of its monopoly in the deep learning space.

Table 1 Arrangements of 32-bit Q-format, including ours

Exponent	Significand	Sign bit	Comments
8	23	1	IEEE-754 like
16	**16**	**-**	**Ours**
10	22	-	-
14	17	1	-

Vendors make different hardware accelerators, and OpenCL is meant to be compatible with as many as possible. A GPU may require a specific workflow between its components, which is different from how a CPU implements a similar workflow. A CPU may require vectorization to execute vector instructions but doesn't have components that are specialized for vectorization. A platform may have one or more of different types of accelerators, therefore achieving parallelism requires partitioning the task amongst a heterogeneous ecosystem to appropriately distribute the execution workload.

2.3 Floating versus fixed point numbers

Generating deep learning models with fixed-precision parameters as opposed to floating-points has shown advantages [14] in terms of accuracy [15]. Fig. 1 shows why digital floating-point arithmetic is non-associative and non-distributive. Floating-point precision has long been a standard in computers [16], but the popularization and consumer adoption of deep learning applications has pushed scientists and engineers to rethink this standard because of major concerns on embedded systems [17]. Floating-point scalars consume more energy compared to fixed point scalars.

Fixed-point arithmetic is a more efficient alternative, but the programmer must manually control and accurately determine the range and flexibility of the variables as shown in Table 1. Digital signal processing falls between two computationally distinct categories:

- Fixed point
- Floating-point

These categories refer to the format used to persist and manipulate numeric data. Integers are represented with 16 bits and floats, as they are commonly called, with 32 bits. This way, floating-point is fit for scientific purposes for its wider range; from very large numbers to extremely small ones. For operations that quickly grow in complexity such as exponentiation, a dynamic format like float is, once again, preferred over fixed point.

In every DSP execution a quantization step is applied which yields errors. This error is a gap between the actual manual computed result and the digital result. Quantization here is a process of rounding and or truncating a result to the nearest value that satisfies a given format [18]. In fixed point format the gap is noticeably bigger, hence floating-point format is, once again, preferred.

Without disregarding the many great benefits of fixed-point and floating-point DSP alike, generally the latter is meant for computationally intensive applications where

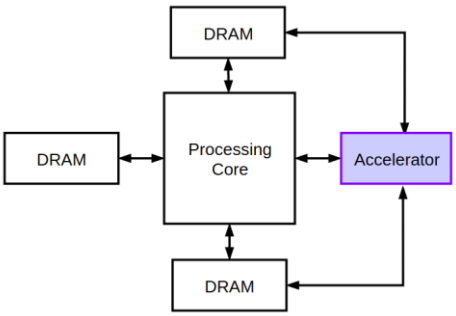

Fig. 2 Illustration of the CPU sharing resources with the accelerator in a parallel computing environment

precision is paramount. The former works great for general purpose, high-volume applications. However, in the next section we reveal ways to manually manage the gaps derived from fixed-point operations in DSP, and even include differential steps in the execution.

2.4 Q-format number notation

The Q format is a proprietary, logical representation, or notation, of fractional bits and optional integer, non-fractional bits, that is fixed in advance; in other words, with Q-format numbers are treated as integers. This notation is intended for use in programming implementations of routines that manipulate numbers on hardware that does not support floating precision. Under strict precautions, Q format numbers can be used in automatic differentiation, although the memory capacity of the resulting gradient must be known in advance to avoid memory overflow errors.

Q-format is an abstraction that allows representing rational numbers that behave like integers by pre-defining the space required for the fixed part and the space required for the fractional part. Using this abstraction, it becomes easy to bring together the benefits of floating-point format into the fixed-point format and maintain a reasonably low rounding gap error. One of the caveats is to consider saturations or scenarios that may lead to an output of a higher format range than the inputs.

Consider A=64 and B=64, 2 scalars of Q-format type with a fixed part of size 8 and a fractional part of size 24, noted Q8.24. This is a signed format that can hold numbers up to 127 (one of the eight fixed part bits is used to hold the sign). The addition of the values yields 128, a number that this format cannot support. This saturation can be handled by either forcing the result to 127, therefore creating a gap of 1, or by choosing a larger Q-format type.

2.5 Convolution operations in deep learning models

Considering the architecture diagram in Fig. 2 with hardware acceleration, manipulating tensors is a multi-step process for computers. 2D convolution is among the most repeated tasks in deep learning [19]. It is implemented in 2 major ways: the sliding-window technique or a large matrix multiplication. The former has come to be known simply as 2D convolution and is focused around a smaller 2-dimensional matrix "sliding" over the larger matrix, such as the pixel values of an image. Each step is a Hadamard product between the small matrix, called a kernel filter, and the corresponding portion of the large matrix. The second method converts the input map into a matrix and simply multiplies it by the kernel. Either method presents advantages and setbacks that fall outside the scope of this work, but we experiment on differentiation applied to the following algorithms:

1) Fast Fourier Transform (FFT): research showed that a 2D convolution solution is the same as the Hadamard product of two FFT transformations followed by an inverse FFT [20]. This discovery helped reduce the complexity from $O(n^2k^2)$ to $O(n^2\log n) + O(n^2)$ when assuming a square matrix. The approach is not optimal for small kernel sizes, which most convolutional neural networks use [21].

2) Winograd Minimal Filtering was proposed to reduce the complexity of convolutions with 3x3 kernels by slicing 4x4 tiles out of the input image [22]. Although this method is limited to a 3x3 size it is in response to a growing trend in the scientific community.

Models used in deep learning are highly structured with particularly few control dependencies, sometimes borrowing from other structured concepts such as graph theory. FFT consists of multiply-accumulate operations that are easy to track in partial or complete executions [23][24]. This, in turn, enables elaborate synergies between hardware and software in ways that could greatly benefit embedded systems. FFT is also highly differentiable.

2.6 Automatic differentiation

Automatic differentiation is increasingly proposed as the default method of computing derivatives in deep learning [25]. Differentiation is a mathematical algorithm used in tensor calculus since most problems in deep learning are expressed in terms multi-dimensional matrix multiplications and summations [26]. Software programming frameworks like TensorFlow [27], and PyTorch [28] are efficient in part due to their automatic computation of gradients. Scalar summation and multiplication are the basis of convolution operations in deep learning. While digital integer is associative, that is notoriously not the case with floating-point, causing problems with parallelization and reproducibility. There exist solutions that unfortunately are not implemented on modern computers.

One of the main contributions of this work is the use of differentiation [25] on the Q-format notation. Automatic

differentiation is the computation of gradients during execution without explicit request from the user [29]. During model definition, the user typically defines the forward path of execution, and the automatic differentiation generates a backward path of execution for computing gradients, see Algorithm 1.

Algorithm 1 Automatic Differentiation

$(q_1, q_2 \dots, q_m) \leftarrow$Input in Q-format
$d_i \leftarrow (d_{ij})_{j=1}^{|previous(i)|} \in D_i(q_{previous(i)})$
$i \leftarrow p+1 \dots q$
$e \leftarrow$ Epsilon (host approximate)
procedure (FORWARD MODE)
  Initialize $\frac{\delta q_k}{\delta q} = \mathbf{e}_k, k = 1, \dots, p$
  for $k = p+1, \dots, m$ **do**
    $\frac{\delta q_k}{\delta q} \leftarrow \sum_{j \in previous(k)} \frac{\delta q_j}{\delta q} d_{kj}$
    where $q = (q_1, \dots, q_p)$
  end for
  return $\frac{\delta q_m}{\delta q_{1,\dots,m}}$
end procedure
procedure (BACKWARD MODE)
  map$\{previous(t)\} \leftarrow t \in (1, \dots, q)$
  Initialize $v \leftarrow (0, 0, \dots, 0, 1) \in \mathbb{R}^q$
  for $t = q, \dots, p+1$ **do**
    for $j \in previous(t)$ **do**
      $v[j] \coloneqq v[j] + v[t] d_{tj}$
    end for
  end for
  return $(v[1], v[2], \dots, v[p])$
end procedure

3. Proposed Method

To increase computing resources savings and reduce latency, we propose to use the Q-format notation. The Fast Fourier Transform and Winograd minimal filtering algorithms are evaluated on this notation. Human-in-the-loop systems built on the assistive intelligence concept are conditioned on fast processing and portability, the former being the missing feature for embedded systems. Our contribution provides a proof of concept for deep learning tasks on embedded systems by applying automatic differentiation on the Q-format notation.

3.1 Assistive intelligence

There are multiple ways to observe intelligent abilities, whether shown by a human or not. The imitation game, for instance, used language to design observable tasks that, once executed, could lead to a measure of intelligence by using a human as benchmark. Understandably, using natural language ability alone was a limiting proposal.

Over the years more tests have been designed and implemented by including other abilities, such as speech understanding, speech, object recognition and more. Subject-specific benchmarks were also devised to monitor advances in technology from multiple fronts.

Embedded systems offer the least requirements to host assistive intelligence, but heavy reliance on floating-point precision makes it difficult. Human-computer interaction scenarios should flow quickly and seamlessly. Therefore, embedded systems should be able to support the requirements of assistive intelligence.

The experiments we conducted provide results towards the realization of assistive intelligence.

3.2 Accelerated computing in embedded systems

Using a kernel library such as OpenCL means critical computation that has been marked for acceleration is compiled to a lower machine instruction code. Memory management may decide which resources to use, either the ones closest to the processor unit or even larger ones availed by the platform itself, such as RAM or hard disks. Luckily this complex undertaking is easy to recognize as a pattern of control and data flow. Memory management can be done manually by the programmer or automated by a caching mechanism that prioritizes reusability and handles overflows. It becomes a matter of balancing, even when units perform in a heterogeneous way. Kernel libraries are responsible for tiling or splitting tensors into smaller tensors and passing them to different caching orders at the same time while auto-evaluating the tiling process itself.

To make optimal use of caching, it is better to concentrate the bulk of operations on one or a few tiles to avoid bottlenecks on the I/O routes. Execution may include repetitions such as loops that cascade into other loops. In this case multiple inner-loop executions for multiple tiles can be serialized to their corresponding outer loops in a process called fusion. Generally, caching is implemented by a technique referred to as SIMD, or single instruction multiple data, and is part of the instruction set architecture (ISA) of a computing device.

Parallel computing addresses some processing bottlenecks by spreading the instructions over multiple simultaneous processors, or accelerators, to both speed up computation and freeing the core processor, or CPU, to focus on orchestration.

Neural networks rely on parallel computing [30], constituting a major setback for embedded systems whose resources and energy consumption discourage parallelism. However, as we will demonstrate in this report, there are domain-specific accelerators that are tuned for convolution operations and consume little energy.

There is a strong correlation between progress registered in machine learning, deep learning, and progress in parallel computation [31]. Deep learning applications, particularly tasks like computer vision, machine translation, speech recognition, etc. have shown that more computation leads to more energy consumption. Operating costs have forced deployments to a client-server approach because low-powered devices cannot bear the energy consumption demands of deep learning applications. There are increasingly more energy-limiting options because industry trends lean towards privacy-preserving options.

3.3 Representing and manipulating numbers

High-performance computing (HPC) remains the de facto environment for many deep learning projects. Ongoing efforts are required to contextualize machine learning to edge devices. In the early days of modern computing, the energy requirements far exceeded what is experienced today. Part of the efforts is therefore to

Algorithm 2 Q-format Conversion

```
l ←Bit scheme (short, single, or double)
e ←Lower bit scheme
p ←Higher bit scheme
m ←Mantissa bit size
x ←Exponent bit size
a ←Exponent value
b ←Mantissa value
s ←Sign value
o ←Output
procedure (ALTER)
    while l < p do
        l = l << 2
        m++
    end while
    while l > e do
        l = l >> 2
        x++
    end while
    a = l + (1 << m)
    b = l − (1 << x)
    for i = 1, . . . , l do
        if b < 0 then
            a = a + (b << i)
            b = b + (a << i)
        else b > 0
            a = a − b(b << i)
            b = b − a(a << i)
        end if
    end for
    o = b × s
    o << (x − m − 1)
    return o
end procedure
```

continue achieving high results with minimal consumption. This trend is gaining more traction in machine learning and deep learning.

The IEEE-754 along with its 2008 revision are engineering standards that specify how digital floating numbers arithmetic is implemented on software and hardware. Although they continue to be challenging engineering problems, little consideration was given to the importance of energy consumption and latency, especially for embedded systems. With the high availability of data and the rise of deep learning [32], power and speed are exacerbated with every new development in the field.

Tensor representations on computer systems have evolved with the remarkable achievements in computer vision tasks [31]. Tensors are represented as large multi-dimensional arrays of floating numbers. Although the manipulations do not go beyond mere summations and multiplications, the tensor itself as a data structure takes millions of steps to be fully traversed, often using multiple levels of control loops, and other repetitive instructions. Floating numbers are suited for this type of computation at the expenses high energy and latency costs, which embedded systems cannot overcome easily.

As shown in Algorithm 2, we propose a 32-bit width with user defined rounding mechanisms. The algorithm shows how a single precision floating number can be converted to the proposed Q-format notation in Section 4.

Efficiency in deep learning, especially for embedded systems, is measured according to:

- The energy consumption of the architecture for any evaluated algorithm, model, or computation [33]. Here we detail the specifications of the process that we subject to computational analysis, although we return to the convolution since it is the most used. The number of trips from processor to memory should also be accounted for.
- The latency corresponds mostly to memory bandwidth [34]. This metric should be found in the actual run time for various executions by comparing timestamps at different stages of execution.
- The area cost of the physical processor or chip. It is the core execution area spent by the instructions.

3.4 Hardware equipment used in experiments

We compare our configuration with different other modes available, including accelerators like the Intel Movidius2 Neural Stick. Popular frameworks like PyTorch and TensorFlow already provide most of the proposed features like automatic differentiation.

Our implementations focus on embedded systems (see Fig.3) by applying automatic differentiation to a proprietary number format that is more adapted to low-powered, low-latency environments.

DSP can only support a limited range of MACs [35] but we feature this type of accelerator in our experiments because it requires little power. Floating-point format is supported in most DSP accelerators at the cost of increased power consumption. Integers are known to consume less energy compared to floating numbers.

Consumer devices have features that address some of the challenges raised in this report, and therefore provide a benchmark on which to assess our implementations. Expressions are evaluated on these pre-packaged devices using Numpy and Python with hardware acceleration handled by the open-source OpenCL kernel library that is supported in all 3 devices shown in Fig. 3.

We compared automatic differentiation for first-order derivatives on our implementations against open-source alternatives mentioned in Section 2.5. We also focused on scalar-valued functions that allowed the use of fixed-precision format.

4. Experiments and Results

For our experiments we designed a 32-bit implementation of the Q-format notation using a modified open-source code base, and we also used the *OpenBLAS*

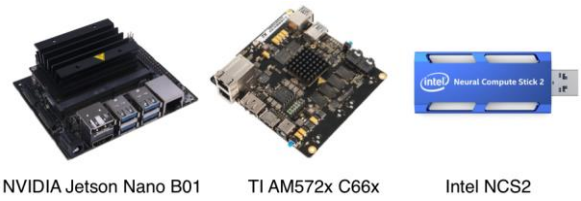

Fig. 3 Three embedded systems used for our experiments: Most use proprietary software and hardware. Vendors continue to show interest for deep learning on small form factors.

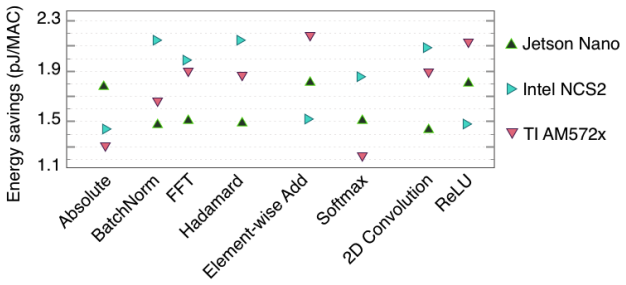

Fig. 4 Energy gains (pico-Joules per multiply-accumulate operation) in comparison to IEEE 754-2008 with similar operations on three embedded devices

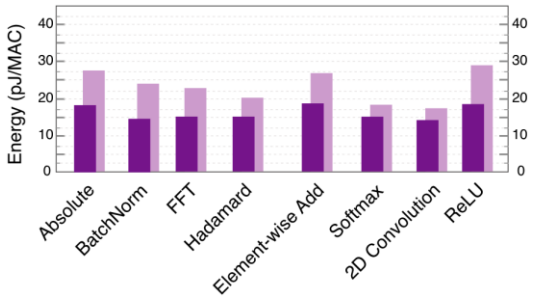

Fig. 5 Q-32 notation (dark purple) shows performance gains over its IEEE 754-2008 alternative on Jetson Nano

scientific calculations library as a benchmark for latency measurements.

The structure of a neural network in deep learning is comprised of layers in a graph-like execution process. Each layer has nodes where certain mathematical operations take place, the result of which are passed on to the nodes in the following layer. A loss function is used to supervise this entire mechanism by applying differentiation to generate parameters, or weights, that best estimate a solution for the problem being solved.

Layers, nodes, and parameters are essentially matrices, and the operations taking place are summations and multiplications. There are so many matrices multiply operations that it serves to measure the complexity of a neural network by simply counting the number of matrix multiplications. Technically the result of the multiplication is added to an accumulator, therefore a MAC, or multiply-accumulate, constitutes a unit of operation.

MAC is used to measure the efficiency, expressed in pico-Joules per MAC (pJ/MAC), and throughput, expressed in Giga-MAC per second (GMAC/sec), of our modified algorithms. We also measure the same for IEEE 754-2008, see Fig. 4 and Fig. 5.

In our experiments we design tensors and implement the tensor manipulation algorithms discussed in Section 2.5. These algorithms are used to write functions common in deep learning. Since many libraries have different implementations of the same functions, we benchmarked our results against the open-source *OpenBLAS* commonly associated with tensor manipulations. The functions are: (i) 2D Convolution, (ii) Element-wise addition, (iii) absolute, and (iv) product, (v) Softmax, (vi) Batch normalization, (vii) ReLU and (viii) the fast-Fourier transform (FFT).

We are interested to find out how efficient and performant our algorithms are when subjected to hardware acceleration on 3 different architecture types. DSP acceleration requires the least energy at the expense of limited applications.

Table 2 Multiply accumulate count, area and delay savings for some algorithms evaluated with a digital signal accelerator on AM572x board: Energy savings (picojoules per MAC), Latency savings (MAC per second), % Gains (comparison to IEEE754-2008)

Algorithm	Energy savings (pJ/MAC)	Latency savings (GMAC/sec)	Energy Gains (%)	Latency gains (%)
Absolute	1.33	3	11	14
FFT	1.9	1.8	24	28
BatchNorm	1.67	2	27	25
2D Conv	1.9	2	21	17
Softmax	1.2	2	16	16

Abbreviations: ABS (Absolute), FFT (fast Fourier transform), BatchNorm (batch normalization), 2D Conv (2-dimensional convolution, Softmax (Softmax)

4.1 CUDA acceleration

Writing kernels or low-level instructions is a recommended method for targeting hardware acceleration on NVIDIA graphic accelerators. Using the proprietary CUDA language with its custom compiler, we applied the Q-32 notation on several algorithms.

The results show how the same algorithms perform using conventional 32-bit floating numbers. Auto-differentiation has no further effects as expected.

4.2 MKL acceleration

Intel hardware acceleration takes place in the central processing unit, giving it a notable difference over the other options we observe in our experiments. Given this particularity, we used proprietary libraries to better take advantage of this architecture.

4.3 OpenCL acceleration

DSP acceleration is limited to signal processing and other related functions. Incidentally, convolutions are particularly suited for this type of acceleration, refer Table 2 and 3. Using the open-source OpenCL kernel library we can target computations that best utilize this architecture. Of all the devices used for experimentation, DSP is the

Table 3 *FPBench* aggregate precision error comparative results on AM572x with C66x DSP (with arbitrary rounding modes)

Operation	Q-32 round-up (*e-23*)	Q-32 truncate (*e-23*)	Float-32 round-to-nearest (e-23)
Sum	6.9	7.1	7.2
Logexp	6	6.3	6.2
matrixDet1	11.7	13.7	13.9
matrixDet2	11.8	14.1	14.4
Sqrt_add	9.9	10.3	11

Abbreviations: Logexp (exponential logarithm), matrixDet1, 2 (determinant matrix 1, 2), Sqrt_add (addition of square roots)

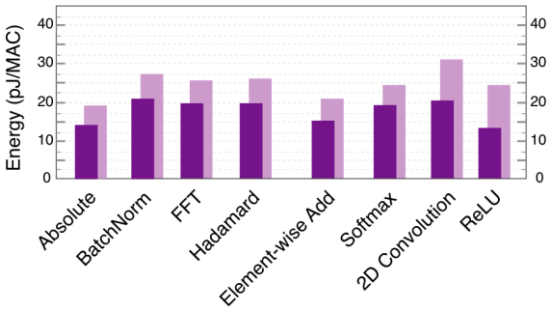

Fig. 6 CPU acceleration for the same custom notation still shows better performance for Q-32 (dark purple) over IEEE 754-2008 (light purple) on Intel NCS2

Fig. 7 DSP accelerated algorithms show best overall results because of domain-specific environment for convolutions: In dark purple: Q-format with auto-differentiation. In light purple: the IEEE 754-2008 results under similar conditions.

least power-consuming platform and therefore presents an obvious advantage.

Refer to Fig. 6 and 7 for more. The percentage gains compared our method to their floating-point alternative.

4.4 Advantages of using DSP hardware accelerator

As explained in section 3 *OpenBLAS* is an open-source library for scientific computations and it contains several algorithms for linear algebra like matrix multiplications. The right-most column in Table 2 shows how algorithms adapted to our proposed method compare to their *OpenBLAS* counterpart in terms of energy gains. This library makes extensive use of multiply-accumulate operations, giving us an ideal environment for evaluation.

We apply our evaluations on 3 devices with the use of Q-format notation for auto-differentiation. For similar tasks, we notice that domain-specific tasks that target DSP acceleration have a clear opportunity to spend less resources without sacrificing performance. Q-format introduces the missing feature for low-powered intelligent assistants unless IEEE 754-2008 addresses the design gaps highlighted in previous sections. But as Table 3 shows, Q-format aggregate precision error (or adjustment error) is slightly reduced from IEEE754-2008 regardless of host architecture. Rounding-up or truncating is a common operation in digital arithmetic. *FPBench* is a benchmark used to evaluate precision errors in floating-point digital arithmetic.

We also show the quality gains on replacing floating numbers with Q-format notations. To check for errors, we manually re-write some arbitrary algorithms to handle our proposed notation, which means we define the operands logic for add and multiply.

We divided our experiments into multiple phases for as many devices as we have. Each phase measures area and delay for several algorithms and the energy gains compared to floating numbers.

Our sample implementation uses a 32-bit Q-format equivalent to IEEE 754-2008 single floating number. Other bit schemes can be used. Conventionally switching from a single floating number to a double floating number is used to further increase the precision of operations at the expense of computing resources. When representing images as 2D matrix with Q-format notation, convolution operations benefit from the reduced errors caused by rounding and a few other known limitations in the IEEE 754-2008 standard.

5. Conclusion

In this work we propose an alternative notation that mimics integer format. We are interested in analyzing the gains this replacement has on embedded devices in terms of power consumption and latency.

Our evaluation shows gains in throughput and latency on a few common algorithms used in deep learning. We show up to 21% gain in energy savings and up to 22% reduced latency while maintaining optimal output error (2.1% average relative error). Our measurements are based on multiply-accumulate (MAC) with throughput, or performance expressed in the number of MACs per second, and energy in pico-Joules per MAC. Processor manufacturers publish theoretical numbers for speed, usually expressed in FLOPS (floating operations per second) which our estimated latency reduction is based on.

Assistive intelligence is a growing field provides interfaces where continuous input is expected throughout execution. The foundations of computing architecture can be revisited without stalling the progress achieved. Hybrid hardware-software tinkering is open to improvement and better numerical approximations.

Q-format notation demands a manual overhaul in the number notation design, which require advanced skills in several fields in computer science and artificial intelligence. Researchers are encouraged to spend more of their skills in lowering the barrier of entry for all people to join this field. That is a much better endeavor to making deep learning an affordable problem-solving toolkit.

Acknowledgments

The authors acknowledge the experimentation support and resources provided by the CIS Corporation Japan.

References

[1] J. Johnson: Rethinking floating point for deep learning, arXivpreprint arXiv:1811.01721, 2018.

[2] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko: Quantization and training of neural networks for efficient integer-arithmetic-only inference, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp.2704-2713, 2018.

[3] D. Lin, S. Talathi and S. Annapureddy: Fixed point quantization of deep convolutional networks, International Conference on Machine Learning, pp.2849–2858, PMLR, 2016.

[4] U. Köster et al.: Flexpoint: An adaptive numerical format for efficient training of deep neural networks, arXiv:1711.02213, 2017.

[5] D. Roel, S. Dean, T. Gilbert and N. Kohli: A broader view on bias in automated decision-making: Reflecting on Epistemology and Dynamics, arXiv preprint arXiv:1807.00553, 2018.

[6] 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.

[7] Precision & Performance: Floating point and IEEE 754 compliance for nvidia graphical processing units (GPUs).
https://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf, 2020. Retrieved: 2020-11-08.

[8] K. Manolopoulos and D. Reisis: An efficient dual-mode floating-point multiply-add fused unit, 17th IEEE International Solid-State Circuits Conference (ISSCC), pp. 82-83, 2016.

[9] N. S. Babu, V. Himaja and B. Srinu: High performance advanced signed multiplier for DSP applications, Proceedings of the Indian J. Sci. Res, Vol. 17, No.12, pp. 329-331, 2018.

[10] V. Sze, Y.H. Chen, T.J. Yang and J.S. Emer: Efficient processing of deep neural networks: A tutorial and survey, Proceedings of the IEEE, Vol. 105, No.12, pp. 2295-2329, 2017.

[11] H. Fan, C. Luo, C. Zeng, M. Ferianc, Z. Que, S. Liu, X. Niu and W. Luk: F-E3D: FPGA-based acceleration of an efficient 3D convolutional neural network for human action recognition, 30th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP), Vol. 2160, pp. 1-8, 2019.

[12] C. Yang, T. Kurth and S. Williams: Hierarchical roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system, Concurrency and Computation: Practice and Experience, Vol.32, No.20, p.e5547, 2020.

[13] D. Silver, A. Huang, C.J. Maddison et al.: Mastering the game of Go with deep neural networks and tree search, Nature, Vol. 529, No.7587, pp.484-489, 2016.

[14] S. Wijeratne, S. Jayaweera, M. Dananjaya and A. Pasqual: Reconfigurable co-processor architecture with limited numerical precision to accelerate deep convolutional neural networks, 29th IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP), pp. 1-7, 2018.

[15] M. Rastegari, V. Ordonez, J. Redmond and A. Farhadi: XNOR-Net: ImageNet classification using binary convolutional neural networks, arXiv 1603.05279, 2016.

[16] D. Lin, S. Talathi and S. Annapureddy: Fixed point quantization of deep convolutional networks, International Conference on Machine Learning, pp. 2849-2858. PMLR, 2016.

[17] Overcoming challenges in fixed point training of deep convolutional networks, ICML Workshop, 2016.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko: Quantization and training of neural networks for efficient integer-arithmetic-only inference, arXiv: 1712.05877, 2017.

[19] R. Zhao and W. Luk: Optimising convolutional neural networks for reconfigurable acceleration, doi 10.13140/RG.2.2.29524.30083, 2017.

[20] M. Mathieu, M. Henaff and Y. LeCun: Fast training of convolutional networks through FFTs, arXiv preprint arXiv:1312.5851, 2013.

[21] G.N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino and Y. LeCun: Fast convolutional nets with fbfft: A GPU performance evaluation, arXiv:1412.7580, 2015.

[22] A. Lavin, and S. Gray: Fast algorithms for convolutional neural networks, Proceedings of the IEEE Conference on Computer Vsion and Pattern Recognition, pp. 4013-4021, 2016.

[23] Y. Kochura, Y. Gordienko, V. Taran, N. Gordienko, A. Rokovyi, O. Alienin and S. Stirenko: Batch size influence on performance of graphic and tensor processing units during training and inference phases, International Conference on Computer Science, Engineering and Education Applications, pp. 658-668, 2019.

[24] S. Park, S. Kim, S. Lee, H. Bae and S. Yoon: Quantized memory-augmented neural networks, Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, No.1, pp. 3909-3916, 2018.

[25] A. G. Baydin, B. A. Pearlmutter, A. A. Radul and J. M. Siskind: Automatic differentiation in machine learning: A survey, Journal of Machine Learning Research, Vol.18, No.153, pp.1-43, 2018.

[26] L. Deng and G. E. Hinton: New types of deep neural network learning for speech recognition and related applications: An overview , IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8599-8603, 2013.

[27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin et al.: TensorFlow: A system for large-scale machine learning, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265-283. 2016.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen et al.: Pytorch: An imperative style, high-performance deep learning library, Advances in Neural Information Processing Systems, Vol. 32, pp.8026-8037, 2019.

[29] B. van Merriënboer, O. Breuleux, A. Bergeron and P. Lamblin: Automatic differentiation in ML: Where we are and where we should be going, arXiv:1810.11530, 2018.

[30] J. Alman and V.V. Williams: A refined laser method and faster matrix multiplication, 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 522-539, 2021.

[31] Y. LeCun, Y. Bengio and G. Hinton: Deep learning, Nature 7553:436, 2015.

[32] J. Johnson: Rethinking floating point for deep learning, arXiv preprint arXiv:1811.01721, 2018.

[33] G. Tagliavini and S. Mach: A transprecision floating-point platform for ultra-low power computing, IEEE Design, Automation & Test in Europe Conference & Exhibition, pp. 1051-1056, 2018.

[34] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang and D. Qian: The deep learning compiler: A comprehensive survey, IEEE Transactions on Parallel and Distributed Systems, Vol. 32, No.3, pp. 708-727, 2020.

[35] M. Gautschi and P. D. Schiavone: Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 25, No.10, pp. 2700-2713, 2017.

Robert Nsinga received his Master's degree in Information Science from the University of Tokushima in 2019. He is currently a Ph.D. graduand in Advanced Intelligent Systems at the University of Tokushima. His research focuses on edge AI, neural networks, optimization, com-puter vision and AI fairness.

Stephen Karungaru received his Ph.D. degree in Information System Design from the Graduate School of Information Science and Intelligent Systems, University of Tokushima in 2004. He is currently an Associate Professor in the University of Tokushima. His research interests include pattern recognition, neural networks, evolutionary computation, image processing and robotics.

Kenji Terada received his doctor degree from Keio University in 1995. In 2009 he became a Professor in the Department of Information Science and Intelligent Systems, University of Tokushima. His research interests are in computer vision and image processing. He is a member of the IEEE, IEEJ, and IEICE.

(Received October 28, 2021; revised February 8, 2022)