Journal of Big Data

RESEARCH Open Access

Neural network training with limited precision and asymmetric exponent

Mariusz Pietrołaj* and Marek Blok

*Correspondence: mariusz.pietrolaj@pg.edu.pl

Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland

Abstract

Along with an extremely increasing number of mobile devices, sensors and other smart utilities, an unprecedented growth of data can be observed in today's world. In order to address multiple challenges facing the big data domain, machine learning techniques are often leveraged for data analysis, filtering and classification. Wide usage of artificial intelligence with large amounts of data creates growing demand not only for storage and operational memory, but also computational power. Increasing complexity and variety of neural network architectures are vivid examples of such trends in the modern data-driven industry. In response to this situation, focusing on less demanding operations for inference and training of neural networks became a popular approach among many researchers to overcome resources related issues. This work aims to investigate one of the paths associated with the mentioned efficiency problems and shows the impact of floating-point precision limitation on convolutional neural networks, including experiments on various exponent and mantissa sizes. Additionally, authors explore floating-point numbers utilization and optimization techniques in the scope of neural network training. Based on conducted research a novel method of asymmetric exponent utilization is presented achieving almost identical accuracy of 32-bit floating-point parameters while training a neural network with only 12-bit variables without additional rounding.

Keywords: Neural network, Deep learning, Machine learning, Big data, Precision limitation, Asymmetric exponent

Introduction

Broad availability of significant amounts of data created great opportunities for solving complex, so far untouchable problems. However, the huge scale of input information is often an obstacle for efficient data processing, especially if human interaction is required [1]. One of the fields in which artificial intelligence (AI) will surely soon outperform people and their senses is computer vision [2]. Nowadays, convolutional neural networks (CNN) are commonly used for image recognition tasks [3]. The classification accuracy presented by CNN architectures grew over time, starting with LeNet-5 appliance to zip codes reading in US postal offices [4]. Then over a dozen years later, the winner of the 2012 ImageNet competition AlexNet [5], initiated a rapid increase of new CNN architectures such as GoogleNet [6] and ResNet [7].

In recent years, there has been a growing impact of the big data field on artificial intelligence algorithms. Unprecedented growth of information produced by people every day, coming from smartphones and the expanding market of Internet of Things (IoT) devices [8], generates an enormous demand for data classification and analysis techniques [9]. This creates an inseparable relation between big data and machine learning as AI requires massive amounts of data to thrive. The growing volume of input data, that needs to be processed by computational devices, imposes not only technical but also financial challenges when it comes to available resources. Additionally, there is a strong tendency for productizing AI applications for various business use cases with the aims of generating company growth and cost reductions [10]. While storing the data itself might be cheap, processing it with complex algorithms requires a significant amount of memory and processing power. Taking these constraints into consideration, multiple methods such as processing parallelization, grid computing and hardware modifications have been proposed [11]. In order to enable a less computationally expensive neural network (NN) training approach, based on big data, authors of this paper focus on NN floating-point parameters representation in order to enable widely available and less computationally expensive data processing.

Access to sufficient amounts of training data is a key factor in most machine learning tasks. Currently we can observe two general obstacles when it comes to big data utilization. First is the lack of well classified data for specialized recognition tasks. This problem occurs especially in the case of supervised learning and opens an area for multiple data augmentation techniques to artificially increase the amount of accessible data [12]. Second problem is contradictory to the first one and revolves around the excess of input data that requires extensive analysis and cleanup before appliance of neural network models [13]. The growing need for energy and computational power is mutual for both mentioned cases, no matter how the input data have been prepared. This opens an interesting field for machine learning methods efficiency improvements, focusing on data and the power expensive training step.

The common factor, characteristic for the majority of neural network architectures, was a rapidly expanding consumption of computational power and operational memory. Moreover, training required a significant amount of input information, hence more storage capacity was mandatory [3]. Access to big data enforced both deeper architectures and more time required for processing data through NN [14]. Growing number of artificial neurons and layers in NN designs, encouraged researchers to leverage graphic processing unit (GPU) cores to speed up the time-consuming training process [15]. Such shift to deep NN (DNN) architectures, besides increased classification accuracy, brought also disadvantages in the form of huge number of power-consuming floating-point operations and memory indispensable to store data and network parameters, implying a need for introducing quantization techniques [16]. Not only creates it resources related issues, but also implies development difficulties such as increased time for design and difficult debugging.

NNs are commonly present on mobile and low power devices where computational limitations are still a significant factor. Dedicated AI chips are often provided externally or integrated into the newest hardware by the market leaders such as Google, Intel, Nvidia or Qualcomm in order to enhance device's computational capabilities [17]. Variety of applications such as phrase recognition, voice translation, image augmentation or data aggregation cause increased demand for AI solutions on mobile devices. Growing number of smart assistants, embedded Digital Signal Processing (DSP) units or IoT devices creates significant constraints on neural network models sizes or their power usage [18]. In most cases mobile devices are used for model inference, due to mentioned hardware limitations, or exchange data over network to leverage capabilities of an external infrastructure. Such limitations are extremely important in case of NNs that utilize user's private data or information from embedded sensors which often cannot be shared via network for NN retraining due to privacy reasons [19, 20]. Improving computational capacity of a device itself would enable both hardware producers and software developers to overcome these issues and adjust pre-trained NN models locally on the device.

Reducing complexity of operations executed during the inference process is already a well-known method to limit NN resource consumption [21]. Currently available machine learning frameworks provide specific tools for neural network inference on mobile devices which shows growing demand for low-power AI applications [22]. Such solutions focus on both quantization techniques and limiting bit count of models' parameters. In a similar way, limiting the size of network parameters gives a chance to save precious resources during the training phase and make it applicable to a wider range of low power devices.

This article is structured in the following way, the second chapter briefly presents theory concerning NN and floating-point numbers. In the third section authors familiarize readers with related study and other research results important to the subject. Chapter four describes limitation experiments conducted by the authors on floating-point based NN parameters. The fifth segment shows modifications introduced to the floating-point format in order to save original accuracy of a tested network despite conducted bit count limitations. Closing chapters depict promising opportunities for future work and conclusions derived from the presented research.

Theory

In general, a neural network is a mathematical structure that approximates the relation between input and output data. The mechanism aims, with a degree of simplification, to resemble processes that occur in a human brain [23]. In most cases, NN requires a significant amount of input data for the training procedure. Based on relations found in previously analyzed samples, a network shapes itself to classify or forecast unseen data. Figure 1 gives an example of a simple DNN structure with two hidden layers.

The network's layers consist of multiple artificial neurons, which store their current state. Under the typical scenario, each of them can be characterized by three features: weights, biases, and activations [25]. Once input is provided to a neuron, it calculates the weighted sum of given input values with the addition of its bias. This operation takes the shape of the following equation.

$$y = \left(\sum_{i=1}^{n} x_i \cdot w_i \right) + b \tag{1}$$

where y—output of the neuron, x_i—input of the neuron, w_i—weight assigned to the input of the neuron, b—bias.

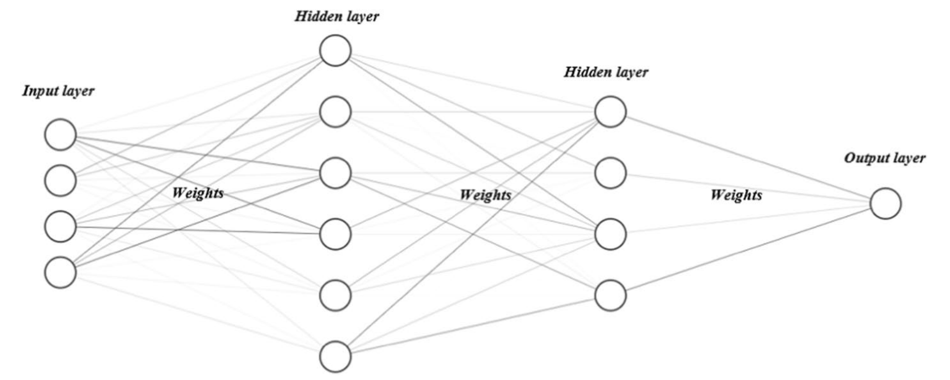

Fig. 1 Deep neural network with two hidden layers [24]

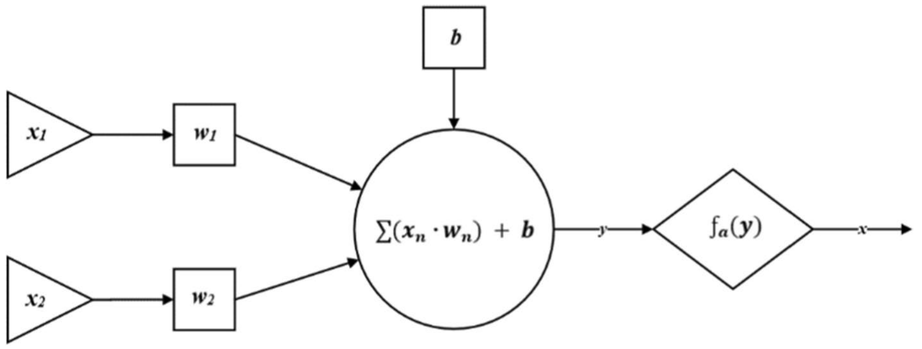

Fig. 2 Single artificial neuron with two inputs

When the output of the neuron is calculated, a special function is used for its activation. In such a way the network can adapt and respond to generalized input based on gathered data [26]. Likewise, output of previous layers is processed throughout the whole network. Figure 2 illustrates the structure of a single artificial neuron.

In order to achieve optimal neurons parameterization to a given problem, the network needs to be adjusted during the process of training. The key element of this scenario is a mechanism of backpropagation, which is an indispensable step in training of DNN [27]. In addition to feedforward pass through the network, backpropagation uses accuracy of the given output to reward or punish neurons to adapt their weights to a proper result. Unfortunately, this step, due to the significant number of floating-point operations required, makes training much more resource and time consuming in comparison to inference.

Floating-point multiplication is a key element of nearly all operations performed within NN. Unfortunately, due to binary representation in computers' memory there is a number of issues and limitations in terms of precision and range of such numbers [28]. Moreover, floating-point operation implies longer and more power consuming calculations for processing units [29]. To save such values in computers' memory a format with separation to exponent and mantissa is commonly used, representing the value with a method identical to the presented equation.

$$x = S \cdot M \cdot B^E \tag{2}$$

where x—floating-point number value, S—sign of the number, M—mantissa, B—base of the number system, two for binary, E—exponent.

There are two contradictory requirements with respect to the use of floating-point variables for NN. A broader range of numbers that can be represented by a given variable allows to achieve improved output accuracy. However, the number of bits used for parameters increases computational power required for multiplication and memory needed to store the result. The common variable used for NN is a 32-bit floating point with 23 bits assigned to mantissa, 8 bits to exponent and 1 bit to sign. Figure 3 provides the structure of such a variable.

Limiting the bit range of variables used by NN can speed up the training process and reduce demand for computational power. Furthermore, low bit range variables utilize less storage space in the device's memory. It needs to be stated that applicability of modified or limited parameters may impose hardware changes as standard floating-point implementation in general use hardware and most popular software AI frameworks do not provide flexible implementation of floating-point variables. In such cases use on-chip or external AI accelerators might be necessary to take full advantage of all benefits provided by limited variables representation. It is worth mentioning that such a shift to specialized hardware units [30], mainly for NN inference, is a common trend in recent years, examples of which are Google TPU, Habana Goya or Nvidia T4 [17]. It must be also remembered that reduction of floating-point variable bit count may impose an accuracy drop in the final training results. Hence, as this paper presents, sophisticated rounding and quantization techniques are often applied to both NN topologies and their parameters.

Related study

Multiple researchers have already investigated the subject of NN training with limited precision. Proposed methods can be divided into two general categories, software limitation and hardware designs. Based on the literature review conducted during this work, a significant number of papers applying to the categories of "neural network" and "precision limitation" after 2015 focus on hardware acceleration for NN training speed up. Combining results of both software modification of floating-point variable structure and usage of dedicated hardware creates a promising area for rapid improvements and reduction of NN's resource consumption.

Software or design changes to parameter types or NN architecture itself, aim to limit the number of time-consuming operations without compromising expected

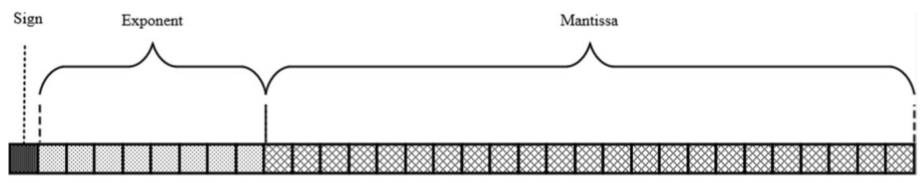

Fig. 3 32-bit floating-point number representation

accuracy. Gupta et al. [31] examined 12-bit and 14-bit fixed-point variables for NN training with the addition of a stochastic rounding algorithm. Their work shows nearly identical results in comparison to 32-bit floating point variables, tested on both the MNIST [32] and CIFAR10 [33] datasets. Moreover, the authors proposed an energy-efficient hardware accelerator for low-precision fixed-point arithmetic with stochastic rounding.

Ortiz et al. [34] presented interesting results on training CNN with CIFAR10 dataset. Their experiments showed inability to train CNN without accuracy degradation while using 12-bit fixed-point representation. Much more promising results were obtained for 12-bit floating-point variables with stochastic rounding, which showed network degradation of less than 2%. The best outcome was achieved with context-based float variables, surpassing base 32-bit results by approximately 2%. The authors also proposed a power of two network implementation that, as the name suggests, uses only bit level operations to limit computational complexity and memory consumption. All limitations were executed with use of 32-bit floating point as a base type.

Comprehensive solution proposed by Na and Mukhopadhyay [35] also combines proposals on the software and hardware side. It introduces a new mechanism of Dynamic Precision Scaling (DPS) which allows to dynamically adjust the precision of a network's parameters based on a stored value. Such approach was also investigated by Taras and Stuart [36] giving 98.8% accuracy on MNIST database with 14-bits precision for weights and 16-bits for activations. In addition to DPS Na and Mukhopadhyay [35] proposed multiplier–accumulator (MAC) design which was introduced to cover multiplication with flexible size of variables. According to the authors this solution has enabled them to train LeNet and AlexNet networks several times faster.

The solution proposed by Park et al. [37] implements stochastic gradient descent with Kahan summation for the low precision parameters update. The validation included MNIST, CIFAR10 and SVHN [38] datasets. Leveraging the lazy update technique allowed authors to achieve accuracy resembling the use of 32 bits floating-point variables on 8-bit signed integers.

Fuketa et al. [39] leveraged a 9-bit floating point format with 5 bits for exponent and 3 bits for mantissa with hidden sign and the most significant bit. The NN training results were verified on AlexNet and ResNet-50 with ImageNet ILSVRC2012 dataset. According to the authors, training results exceeded the accuracy of regular 16-bit floating point variables. In addition, estimated hardware size has been proposed for this method.

Although NN training with limited precision is still an open subject, there are plenty of solutions already using such a method for less resource-intensive inference. In this approach the network is trained on big data in full precision and then quantized to lower bit count types in order to save operational memory, storage and computational power. The aforementioned technique is already available in several NN related frameworks such as Tensorflow [40] or Pytorch [41].

Distinctive quantization-based approach presented by Onishi et al. [42] focuses on limiting memory and power consumption during NN training leveraging lookup tables (LUT). Their solution was validated on LeNet-5 network with MNIST dataset.

The results showed that LUT-based training technique allowed researchers to limit memory usage up to 22% during forward pass and 60% for backpropagation. The overall number of multiplication operations have been limited by 11.7%. The presented improvements were achieved with 1.41% NN accuracy loss.

In the field of hardware improvements Lee et al. [43] proposed Unified Neural Processing Unit (UNPU) which supports flexible precision from 1 to 16 bits for convolutional, fully connected, and recurrent layers of NN. Moreover, the proposed hardware limits the amount of off-chip memory access for additional speed up. According to authors such a design allows to reduce potential energy consumption and external memory access to about 50% for specific NN architectures.

Table 1 gives a concise summary of the presented limitation techniques, including the solution proposed by the authors. Provided comparison includes variable type limitation, specific technique used during research and topologies with datasets used for experiments. As stated at the beginning of this chapter, classification for software and hardware solutions is also highlighted in the table. Although most of the papers defined accuracy results in relation to baseline training, there is a difficulty in direct comparison between chosen techniques. There are major differences between NN topologies architecture, baseline accuracy of the model or number of epochs used during training. Taking that into consideration the authors decided to provide differences between given baseline accuracy and, where it was possible, the one achieved after the limitation step. Such a measure can be used for preliminary evaluation of the proposed solutions.

Floating-point limitation

As indicated in chapter 2, floating-point operations are one of the most demanding calculations from the perspective of modern computers. Consumed computational power and memory are correlated with the size of floating-point variables used during multiplication. The first stage of the conducted work aimed to investigate the decrease of CNN accuracy while limiting bit count available for network parameters. In order to achieve such results a benchmark LeNet CNN has been implemented and trained with various limited bit ranges for both exponent and mantissa. The detailed structure of the network is presented in Table 2.

Due to available hardware all limitation steps were carried out in software layer only, in a similar manner to the work of Ortiz et al. [34]. The experiments used generally available 32-bit hardware with full 32-bit floating-point variables as network parameters. The limitation of weights, biases and gradients has been simulated at each step of the network execution and stored in the full-size variables. The development environment leverages python with the Pytorch framework used for NN implementation. In order to overcome python limitation while dealing with floating-point variables bits manipulation a separate, custom C++ library has been created. Boost framework was used to link python to the aforementioned library. After every step of the network calculations, layer by layer, each parameter has been cut to fit the given bit count of exponent and mantissa. A similar approach has been executed for gradient parameters. The diagram presented in Fig. 4 gives an overview of the used environment.

The training process leverages MNIST dataset which consists of 60,000 training and 10,000 test examples of hand-written digits [32]. It is a common input data used across

Table 1 Detailed summary of the related study with comparison to the proposed technique of limitation with asymmetric exponent

Paper	Variable type	Technique	Category	Dataset	Topology	32-bit baseline accuracy	Accuracy after limitation	AI framework
Gupta et al. [31]	12-bit fixed point 14-bit fixed point	Stochastic rounding	Software limitation Hardware design	MNIST	Custom LeNet	99.23%	99.17% (14-bit fixed point) 99.11% (12-bit fixed point)	Not defined
				CIFAR10	3-layer CNN	75.4%	74.6% (14-bit fixed point) 71.2% (12-bit fixed point)	
Ortiz et al. [34]	12-bit floating point 12-bit fixed point	Stochastic rounding Context representation	Software limitation	CIFAR10	3-layer CNN	75.6%	63.03% (12-bit fixed point) 74.20% (12-bit floating point) 78.02% (12-bit context-float) 76.32% (12-bit context-fixed)	Caffe
Na and Mukhopadhyay [35]	16-bit fixed point 32-bit fixed point	Dynamic precision scaling (DPS) Flexible multiplier-accumulator (MAC)	Hardware design	MNIST	LeNet	Not given (only loss charts presented)	32-bit fixed point accuracy achieved on 16-bit fixed point with DPS	Caffe
				Flickr images	AlexNet (pre-trained)		64-bit fixed point accuracy achieved on 32-bit fixed point with DPS	
Taras and Stuart [36]	14-bit fixed point (weights) 16-bit fixed point (activations)	Stochastic rounding Dynamic precision scaling (DPS)	Software limitation	MNIST	LeNet	98.80%	98.80%	Caffe
Park et al. [37]	Combination of 8-bit and 16-bit integers	Stochastic gradient descent with Kahan summation Lazy update	Software limitation	MNIST	LeNet-like CNN	99.10%	99.24%	Caffe Tensorflow
				SVHN	4-layer CNN	97.06%	96.99%	
				CIFAR10	3-layer CNN	81.56%	81.17%	
					ResNet-20	90.16%	90.23%	
				ImageNet	AlexNet	80.81%	80.62%	

Table 1 (continued)

Paper	Variable type	Technique	Category	Dataset	Topology	32-bit baseline accuracy	Accuracy after limitation	AI framework
Fuketa et al. [39]	9-bit floating point format with hidden most significant bit and sign bit	Custom float representation Custom MAC unit	Software limitation Hardware design	ILSVRC	AlexNet ResNet-50	48.27% 68.84%	46.18% 67.55%	Not defined
Onishi et al. [42]	No strict parameters limitation, factorization based on LUT is used for limiting memory consumption and multi-adds operations	Lookup-Table (LUT) based quantization Cluster swap	Software limitation Hardware design	MNIST	LeNet	99.28%	97.87% Memory consumption reduced 22.2% (forward pass) 60% (backward pass)	PyTorch
Lee et al. [43]	Fully variable weight bit-precision from 1 to 16 b	Original hardware accelerator for CNN-RNN networks	Hardware design	Not applicable	AlexNet VGG-16	Not applicable	Operation based power savings presented	Not applicable
Our proposal	8-bit floating point 12-bit floating point 14-bit floating point	Asymmetric exponent No additional rounding	Software limitation	MNIST	LeNet	96.04%	75.89% (8-bit floating point) 95.01% (12-bit floating point) 97.13% (14-bit floating point)	PyTorch

Table 2 LeNet-5 structure used for experiments

Item	Parameters
1st convolutional layer	Input channels: 1 Output channels: 6 Kernel: 5
2nd convolutional layer	Input channels: 6 Output channels: 16 Kernel: 5
3rd convolutional layer	Input channels: 16 Output channels: 120 Kernel: 1
1st fully connected layer	Input: 120 Output: 84
2nd fully connected layer	Input: 84 Output: 10
Activation function (all layers)	TANH
Classifier	Log_softmax

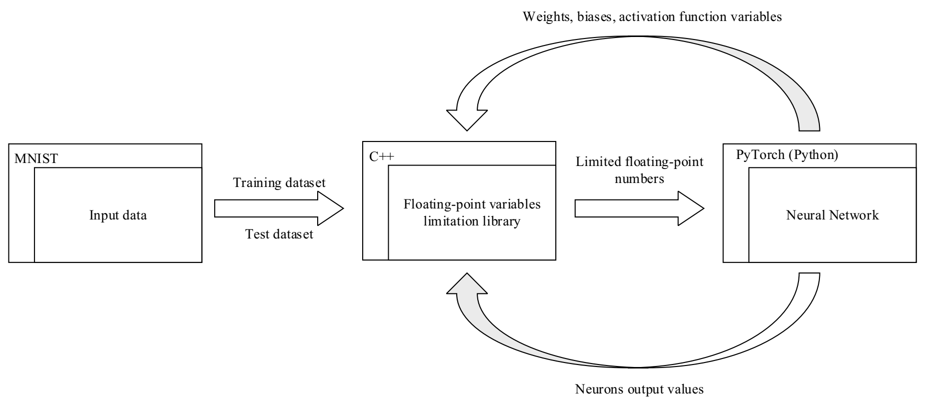
Weights, biases, activation function variables
MNIST
Input data
Training dataset
Test dataset
C++
Floating-point variables limitation library
Limited floating-point numbers
PyTorch (Python)
Neural Network
Neurons output values

Fig. 4 Overview of the environment used for NN parameters limitation

various CNN verification or testing tasks. Choice of the dataset and network architecture was dictated by its relative simplicity and commonness with various research and how-to examples, which should significantly help with the experiment results reproduction by other parties.

The network limitation phase covered various bit ranges of exponent and mantissa. The 32-bit floating-point variable, which was used as a base type, enforced exponent limitation to values between 1 and 8 bits, mantissa covered bit counts from 1 to 23. As previously stated, the limitation included weights, biases and gradients at each stage of NN execution. The baseline training phase was limited to 10 epochs which allowed us to achieve 96% accuracy on 32-bit floating point. Figure 5 presents accuracy of the trained NN across all examined bit counts for both exponent and mantissa.

Based on the results illustrated by Fig. 5, there is a clear increase of the network accuracy along with a growing bit range of the network's parameters. This behavior is expected because a higher number of bits makes it possible to store a wider range of values in each of the network's parameters. As in the previously discussed research, we can confirm that 16-bit floating point is enough for training the network (Fuketa

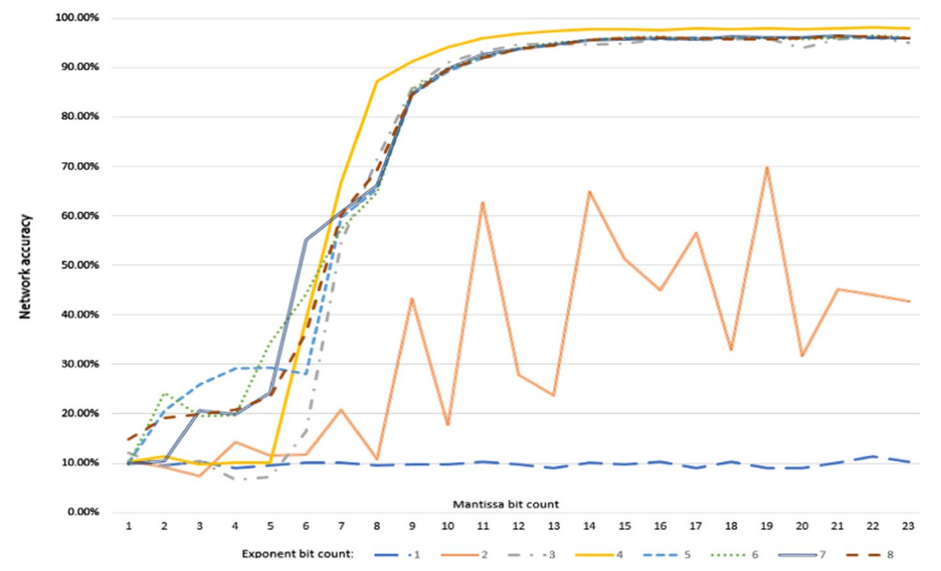

Fig. 5 Neural network accuracy in various exponent and mantissa bit count scenarios

et al. [39]). The decrease in accuracy with this number of bits is negligible while smaller coefficients size helps to significantly limit memory used for network storage.

The presented results show that setting exponent bit count below 3 bits is not enough to train the network without significant decrease of accuracy. It can be observed that a network with 1-bit exponent is unable to approximate data classification and maintains 10% of accuracy for all mantissas' bit counts. In case of 2-bit exponent, Fig. 5 shows an irregular behavior with inconsistent accuracy up to 70% for 19-bit mantissa with poor 42% for 23-bit mantissa. Such behavior suggests that 2-bit exponent is not sufficient for the training process and inconsistent results for increasing mantissa's bit counts may depend on model parameters initialization values. Interestingly, limitation to as low as 3-bit exponent provides satisfactory results achieving 95% accuracy starting from 12-bit mantissa. Even better outcome can be observed for 4-bit exponent, it is vivid that in this case limitation provides a regularization mechanism that not only helps to achieve the expected accuracy with lower mantissa bit count but also exceeds standard 32-bit accuracy by above 2% for 23-bit mantissa and 1% for 12-bit mantissa.

Apart from the accuracy of the trained network, an interesting observation can be spotted once we investigate bits utilization for both exponent and mantissa in each parameter type. In the case of exponent, it is clear that only negative values are used while training the analyzed NN. This means that half of them are not used, and these values are generally wasted. Naturally, this situation may vary depending on the network architecture or activation function used. Figure 6 depicts exponent utilization for all layers in the tested network. Each histogram presents exponent values gathered for every weight parameter in a particular layer across all training batches. The data has been logged and depicted with usage of Pytorch Tensorboard [44]. Axis X describes values of the parameter's exponent and axis Y stands for batch number in 10 epochs training process. Starting from the first layer to the last fully connected layer,

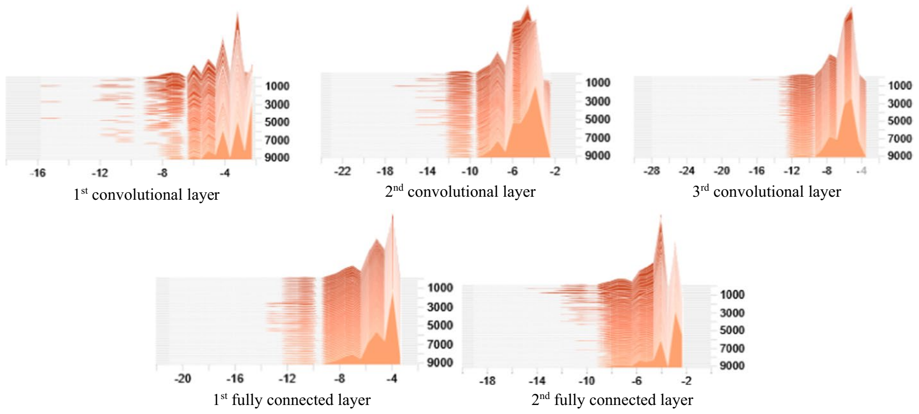

Fig. 6 Weights exponent values utilization during training of the network (per training batch) [44]

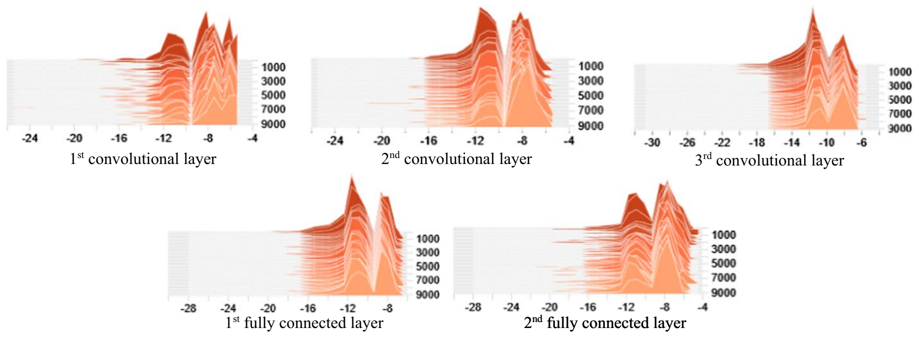

Fig. 7 Weights gradient exponent values utilization during training of the network (per training batch) [44]

we can observe that only negative values of exponent, in the range of − 22 to − 2 are stored in network parameters for each batch iteration. Taking into consideration that the 8-bit exponent allows for values ranging from − 128 to 127, there is an underutilization of available bits in the exponent domain. The same statement is also true for biases stored in the network. A similar situation is observed for weights' gradient values which are used for the backpropagation step, presented in Fig. 7. In this case only exponent values in the range of − 30 to − 4 are stored in gradient parameters.

As mentioned before, exponent utilization in the network omits its positive values. Taking into consideration that biased exponent splits its range in half to negative and positive values, there is a promising expectation that leveraging unused values of exponents may increase accuracy of a network trained with limited floating-point parameters.

Experiments

Several research [31, 34, 37] and results presented in the previous chapter show that limitation of neural network's parameters is, in many cases, followed by a rapid degradation of its final accuracy. Nevertheless, sophisticated rounding techniques or regularization methods may help to minimize this effect. Encouraged by previous experiments with parameters limitation, the authors focused on implementation of a method involving asymmetric exponent representation in order to fully utilize its bit count available during NN training.

The study of the use of exponent bits, carried out in the previous section, showed that a significant number of exponent values have been unused during the training process. The histograms presented in Figs. 6 and 7 clearly indicate that only negative exponent values are stored in the validated network's parameters. In this case it is possible to utilize twice as many exponent values for the same bit count. The authors decided to follow this path and change the exponent range in the specified bit count, creating a floating-point variable with an asymmetric exponent. Based on current bits utilization, a new type of floating-point variable has been implemented to use all available exponent bits for negative values only. The new type was applied to weights and biases in each layer of the network. In all instances, mantissa was cut to match the given bit range. The previously described software environment has been adjusted to implement the required limitation.

During initial research asymmetric exponent has been also applied to limited gradient values. However, further experiments have shown that usage of a regular symmetric exponent format in case of gradient values significantly improves network accuracy for lower mantissa bit counts. Considering this, the authors decided to not apply asymmetric exponent format to limited gradient values. Table 3 presents the comparison of accuracy for a network with the limitation, described in the chapter 3 of this paper, asymmetric exponent and asymmetric exponent excluding limited gradient values. All results are presented for a 3-bit exponent as it has given the best increase in accuracy after application of the asymmetric exponent method.

The use of the asymmetrical method on a 3-bit exponent gives a similar value range to the regular 4-bit exponent. Nevertheless, the comparison of NN accuracy results, especially on smaller mantissa's bit counts, shows significant advantage of asymmetric approach. Table 4 presents the comparison of 3-bit exponent with asymmetric approach and regular 4-bit exponent. In both cases mantissa's bit counts are the same.

Table 3 Comparison of neural network accuracies with 3-bit exponent

Bit counts			**Network accuracy**		
Bit count	**Exponent**	**Mantissa**	**Limitation only (%)**	**Limitation with asymmetric exponent (%)**	**Limitation with asymmetric exponent (excluding gradients) (%)**
5	3	1	12.08	10.09	9.80
6	3	2	9.39	8.92	14.29
7	3	3	10.42	9.58	57.31
8	3	4	6.63	10.10	75.89
9	3	5	7.17	22.42	85.97
10	3	6	16.37	41.32	89.31
11	3	7	54.61	66.52	90.27
12	3	8	71.45	86.98	95.01
13	3	9	85.77	91.34	94.07
14	3	10	91.06	94.54	97.13
15	3	11	93.26	95.93	97.76
16	3	12	94.77	97.29	97.32

Table 4 Neural network accuracy comparison between 4-bit exponent and 3-bit asymmetric exponent

Mantissa's bit count	4-bit regular exponent (%)	3-bit asymmetric exponent (%)	Accuracy improvement (%)
4	10.10	75.89	65.79
5	10.10	85.97	75.87
6	39.21	89.31	50.10
7	66.72	90.27	23.55
8	87.28	95.01	7.73
9	91.29	94.07	2.78
10	94.12	97.13	3.01
11	95.88	97.76	1.88

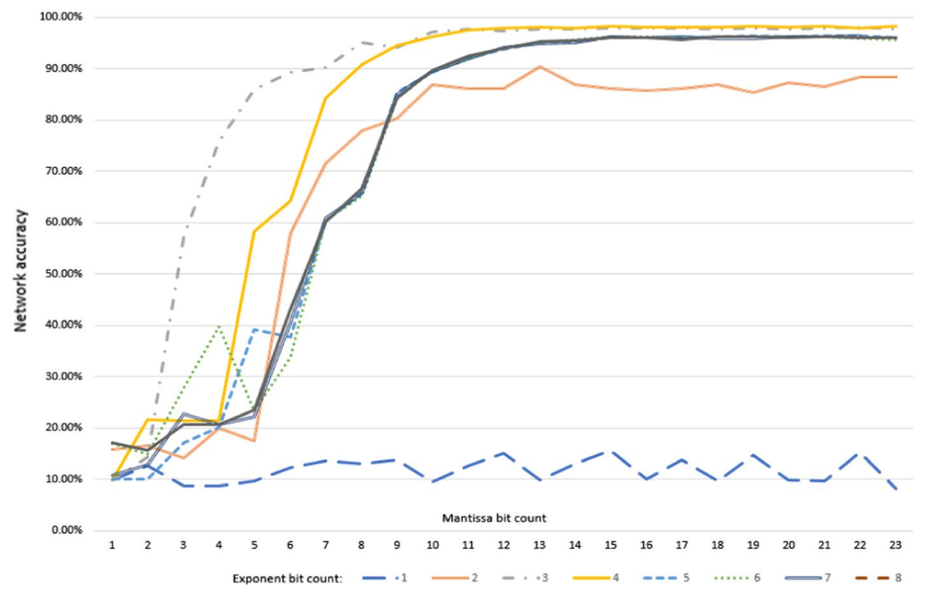

Fig. 8 Neural network accuracy with asymmetric exponent (excluding gradients) in various exponent and mantissa bit count scenarios

Based on the above results the authors decided to cross-check all available limitation bit counts. Figure 8 provides a summary of the network accuracy for the described method.

The presented results show that both 3-bit and 4-bit exponent values with asymmetric approach are sufficient to train the network. Accuracy equal to 32-bit floating-point number is achieved for much smaller mantissa ranges, 12-bit value is sufficient for achieving 95% network accuracy. In addition, this approach allows to train the network and reach 75.89% accuracy for as small parameters as 8-bit floating-point which was presented in Table 3. It should be noted that, in contrast to the previously presented research, no rounding algorithms have been applied for the asymmetric exponent method. This leaves an additional margin for possible method improvements leveraging quantization and rounding techniques.

Future work

Considering the promising results of the presented research, the authors are working on additional methods to verify and improve floating-point numbers limitation for NN training. Leveraging sophisticated rounding algorithms along with the asymmetric exponent method might be necessary to improve presented NN accuracy even further. A similar in-depth investigation is currently being conducted for AlexNet utilizing CIFAR10 dataset with preliminary results confirming applicability of the asymmetric exponent method to more complex, deeper networks. In addition to deeper NN topologies, the authors are verifying other use cases of limited CNN topologies in the audio domain. Key phrase detection or speech recognition are common functions implemented on embedded devices that may benefit from reduced computational complexity or power consumption. Another research direction pursued by the authors is verification of floating-point limitation with the asymmetric exponent for Recurrent NN (RNN) in order to confirm robustness of the proposed technique for other types of NN. As a final step, authors consider usage of custom FPGA implementation to estimate potential power savings resulting from the proposed calculation simplification.

Conclusion

Challenges created by a rapid increase of data and complexity of related problems require more and more computational power. In such cases, it is critical to ensure that provided resources are optimized and utilized efficiently. The presented work touches on this issue with a thorough study of floating-point variables utilization in NN parameters. Bit count usage for multiple exponent and mantissa configurations have been presented based on a LeNet network along with the data required to reproduce the experiments. A novel method of asymmetric exponent has been described giving extremely promising results of NN accuracy after its parameters limitation. The method achieved 95% accuracy with usage of 12-bit floating-point parameters without any additional rounding which is similar to results of 32-bit floating-point variables. Moreover, 8-bit floating-point variables were sufficient for reaching over 75% network's accuracy. Based on the asymmetric exponent method, this work presents new types of variables that could be utilized by specific hardware in order to save both memory and energy consumption during NN training and state a solid base for research continuation.

Abbreviations

AI: Artificial intelligence; CNN: Convolutional neural network; NN: Neural network; DNN: Deep neural network; GPU: Graphics processing unit; MAC: Multiplier–accumulator; DPS: Dynamic processing scaling; LUT: Lookup table; IoT: Internet of things; RNN: Recurrent neural network.

Acknowledgements

Not applicable.

Author contributions

Not applicable. All authors read and approved the final manuscript.

Funding

Not applicable.

Availability of data and materials

The MNIST dataset [32] used in this research is publicly available at: http://yann.lecun.com/exdb/mnist/.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 10 May 2021 Accepted: 6 April 2022
Published online: 12 May 2022

References

1. Najafabadi M, Villanustre F, Khoshgoftaar T, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015. https://doi.org/10.1186/s40537-014-0007-7.
2. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, et al. ImageNet large scale visual recognition challenge. Int J Comput Vis. 2015;115(3):211–52.
3. Rawat W, Wang Z. Deep convolutional neural networks for image classification: a comprehensive review. Neural Comput. 2017;29(9):2352–449.
4. LeCun Y, Boser B, Denker J, Henderson D, Howard R, Hubbard W, et al. Backpropagation applied to handwritten zip code recognition. Neural Comput. 1989;1(4):541–51.
5. Krizhevsky A, Sutskever I, Hinton G. ImageNet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84–90.
6. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR): Piscataway; 2015. p. 1–9.
7. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR): Piscataway; 2016. p. 770–8.
8. Al-Sarawi S, Anbar M, Abdullah R, Al Hawari A. Internet of things market analysis forecasts, 2020–2030. London: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4); 2020.
9. Khan N, Yaqoob I, Hashem I, Inayat Z, Mahmoud Ali W, Alam M, et al. Big data: survey, technologies, opportunities, and challenges. Sci World J. 2014;2014:1–18.
10. Ghimire A, Thapa S, Jha A, Adhikari S, Kumar A. Accelerating business growth with big data and artificial intelligence. Palladam: 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC); 2020.
11. Tsai C, Lai C, Chao H, Vasilakos A. Big data analytics: a survey. J Big Data. 2015. https://doi.org/10.1186/s40537-015-0030-3.
12. Shorten C, Khoshgoftaar T. A survey on image data augmentation for deep learning. J Big Data. 2019. https://doi.org/10.1186/s40537-019-0197-0.
13. Ma T, Garcia R, Danford F, Patrizi L, Galasso J, Loyd J. Big data actionable intelligence architecture. Journal of Big Data. 2020;7(1).
14. Chen X-W, Lin X. Big data deep learning: challenges and perspectives. IEEE Access. 2014;2:514–25.
15. Oh K, Jung K. GPU implementation of neural networks. Pattern Recogn. 2004;37(6):1311–4.
16. Choi Y, El-Khamy M, Lee J. Towards the limit of network quantization. Comput Vis Pattern Recognit. 2016. https://doi.org/10.48550/arXiv.1612.01543.
17. Reuther A, Michaleas P, Jones M, Gadepally V, Samsi S, Kepner J. AI accelerator survey and trends. Massachusetts: 2021 IEEE High Performance Extreme Computing Conference; 2021.
18. Lane N, Bhattacharya S, Mathur A, Georgiev P, Forlivesi C, Kawsar F. Squeezing deep learning into mobile and embedded devices. IEEE Pervasive Comput. 2017;16(3):82–8.
19. Tawalbeh L, Saldamli G. Reconsidering big data security and privacy in cloud and mobile cloud systems. J King Saud Univ Comput Inf Sci. 2021;33(7):810–9.
20. Sarker I, Hoque M, Uddin M, Alsanoosy T. Mobile data science and intelligent apps: concepts, AI-based modeling and research directions. Mob Networks Appl. 2020;26(1):285–303.
21. Wu J, Leng C, Wang Y, Hu Q, Cheng J. Quantized convolutional neural networks for mobile devices. USA: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. p. 4820–8.
22. David R, Duke J, Jain A, Janapa Reddi V, Jeffries N, Li J, Kreeger N, Nappier I, Natraj M, Wang T, Warden P, Rhodes R. TensorFlow Lite Micro: embedded machine learning for TinyML systems. Proc Mach Learn Syst. 2021;3:800–11.
23. Nwadiugwu MC. Neural networks, artificial intelligence and the computational brain. Neuron Cogn. 2020. https://doi.org/10.48550/arXiv.2101.08635.
24. LeNail A. NN-SVG: publication-ready neural network architecture schematics. J Open Source Softw. 2019;4(33):747.
25. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev. 1958;65(6):386–408.
26. Sharma S, Sharma S, Athaiya A. Activation functions in neural networks. Int J Eng Appl Sci Technol. 2020;4(12):310–6.
27. Hecht-Nielsen R. Theory of the backpropagation neural network. Neural Netw. 1988;1:445.
28. Goldberg D. What every computer scientist should know about floating-point arithmetic. ACM Comput Surv. 1991;23(1):5–48.
29. Salehi S, DeMara R. Energy and area analysis of a floating-point unit in 15 nm CMOS process technology. Florida: SoutheastCon 2015; 2015.

30. Baischer L, Wess M, TaheriNejad N. Learning on hardware: a tutorial on neural network accelerators and co-processors. Mach Learn. 2021. https://doi.org/10.48550/arXiv.2104.09252.
31. Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P. Deep learning with limited numerical precision. France: Proceedings of the 32nd International Conference on Machine Learning; 2021. p. 1737–46.
32. LeCun Y, Cortes C. MNIST handwritten digit database. 2010. http://yann.lecun.com/exdb/mnist/. Accessed 25 Apr 2021.
33. Krizhevsky A. Learning multiple layers of features from tiny images. Germany: BibSonomy; 2009.
34. Ortiz M, Cristal A, Ayguadé E, Casas M. Low-precision floating-point schemes for neural network training. Mach Learn. 2018. https://doi.org/10.48550/arXiv.1804.05267.
35. Na T, Mukhopadhyay S. Speeding up convolutional neural network training with dynamic precision scaling and flexible multiplier-accumulator. San Francisco: Proceedings of the 2016 International Symposium on Low Power Electronics and Design; 2016.
36. Taras I, Stuart DM. Quantization error as a metric for dynamic precision scaling in neural net training. Mach Learn. 2018. https://doi.org/10.48550/arXiv.1801.08621.
37. Park H, Lee JH, Oh Y, Ha S, Lee S. Training deep neural network in limited precision. Neural Evol Comput. 2021. https://doi.org/10.48550/arXiv.1810.05486.
38. Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng YA. Reading digits in natural images with unsupervised feature learning. New Orleans: NIPS Workshop on Deep Learning and Unsupervised Feature Learning; 2011.
39. O'uchi S, Fuketa H, Ikegami T, Nogami W, Matsukawa T, Kudoh T, et al. Image-classifier deep convolutional neural network training by 9-bit dedicated hardware to realize validation accuracy and energy efficiency superior to the half precision floating point format. IEEE International Symposium on Circuits and Systems (ISCAS): Piscataway; 2018.
40. TensorFlow. TensorFlow. 2021. https://www.tensorflow.org/. Accessed 25 Apr 2021.
41. PyTorch. Pytorch.org. 2021. https://pytorch.org/. Accessed 25 Apr 2021.
42. Onishi K, Yu J, Hashimoto M. Memory efficient training using lookup-table-based quantization for neural network. IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS): Piscataway; 2020.
43. Lee J, Kim C, Kang S, Shin D, Kim S, Yoo H. UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. 2018 IEEE International Solid—State Circuits Conference—(ISSCC): Piscataway; 2018.
44. TensorBoard|TensorFlow. TensorFlow. 2021. https://www.tensorflow.org/tensorboard. Accessed 25 Apr 2021.

Publisher's Note