Tunable Floating-Point for Artificial Neural Networks

Franceschi, Marta; Nannarelli, Alberto; Valle, Maurizio

Published in:
Proceedings of 25th IEEE International Conference on Electronics Circuits and Systems

Link to article, DOI:
10.1109/ICECS.2018.8617900

Publication date:
2018

Document Version
Early version, also known as pre-print

Link back to DTU Orbit

Citation (APA):
Tunable Floating-Point for Artificial Neural Networks

Marta Franceschi¹, Alberto Nannarelli², Maurizio Valle¹,
¹Cosmic Lab, DITEN, University of Genova, Italy
²DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark

Abstract—Approximate computing has emerged as a promising approach to energy-efficient design of digital systems in many domains such as digital signal processing, robotics, and machine learning. Numerous studies report that employing different data formats in Deep Neural Networks (DNNs), the dominant Machine Learning approach, could allow substantial improvements in power efficiency considering an acceptable quality for results. In this work, the application of Tunable Floating-Point (TFP) precision to DNN is presented. In TFP different precisions for different operations can be set by selecting a specific number of bits for significand and exponent in the floating-point representation. Flexibility in tuning the precision of given layers of the neural network may result in a more power efficient computation.

Index Terms—Floating-point, power efficiency, neural networks

I. INTRODUCTION

In the last years, approximate computing has been undergoing a rapid growth as a promising approach to energy-efficient design of digital systems in a wide spectrum of domains ranging from digital signal processing, to robotics and machine learning. By relying on the ability of many systems and applications to tolerate some loss of accuracy, approximate computing techniques achieve improved energy efficiency.

Approximate computing encompasses a broad spectrum of techniques that relax accuracy to improve efficiency: both at hardware and software level (e.g. skipping computations, voltage overscaling, loop perforation, and quality-scalable or approximate circuits [1], [2]). Moreover, there are some studies that combine multiple approximation techniques [3].

Deep Neural Networks (DNNs) has emerged as the dominant Machine Learning (ML) algorithm showing remarkable success in many challenging application domains as image processing, speech recognition, and machine translation [4], [5]. As high classification accuracy comes at expenses of significant computation cost (area/time/energy) in DNNs [6], it may be advantageous to trade computational throughput for accuracy/quality.

The precision and the data format requirements depend on the computation phases/workloads in DNNs. The training phase presents numerical challenges as it typically contains millions of parameters that are usually trained iteratively over a vast amount of data. In contrast, in the inference phase, input data have to go only once through the network and the required precision is usually lower than in the training phase [6].

Traditionally, neural networks (NNs) are trained in double or single-precision, a common practice in general scientific computing. However, to reduce the execution time (especially in memory transfers) and to increase the energy efficiency, the computation is migrated from double-precision (binary64: 53-bit significand¹, 11-bit exponent) in IEEE 754-2008 standard [7], to single (binary32: 24-bit significand, 8-bit exponent) and half (binary16: 11-bit significand, 5-bit exponent) precision.

Recent developments suggest that more efficient data formats could allow savings in power consumption at the cost of reduced precision in the results. Google has already made its way into production hardware with the Tensor Processing Unit (TPU) [8] designed for inference. The TPU supports the Brain-FP format consisting of 8-bit for the significand and 8-bit for the exponent. Moreover, some studies show that it is possible to achieve dramatic reductions in bit width from 32-bit all the way down to one bit (i.e., binary networks [9]).

Nvidia included Tensor cores specifically designed for ML in their latest generation of GPUs [10]. Each Tensor core can perform fully parallel 4×4 matrix multiplication on binary16 operands to produce binary32 results.

Intel introduced the “Flexpoint” format for deep learning with the aim to replace the training done in binary32. The Flexpoint format is a blocked fixed-point format for tensors (matrices) consisting in a block of 16-bit significands sharing a 5-bit exponent [11].

In this paper, a deep neural network has been exploited to evaluate a novel approximate computing technique, Tunable Floating-Point (TFP), reducing power consumption while still producing acceptable accuracy of results both in the training and inference phases. In fact, the main TFP advantage is flexibility: the precision of operands and results can be chosen for single operations by selecting a specific number of bits for significand and exponent in the floating-point representation. By tuning the precision of the operands and results in a given layer of the DNN, computations can be more power efficient.

II. TUNABLE FLOATING-POINT

The Tunable Floating-Point (TFP) representation is a floating-point (FP) representation with arbitrary number of significand’s and exponent’s bits: m and e, respectively. The main advantage of FP over the fixed-point (FXP) representation is that the dynamic range is much larger than the FXP for similar bits of storage. This is true especially for multiplication where the dynamic range increases quadratically. In TFP, dynamic ranges

¹It includes the integer bit. Significands are normalized in [1.0, 2.0] in IEEE 754.
are only considered from and below the binary32 representation. These ranges are suitable for ML applications.

The TFP representation is normalized to have the conversions compatible with the IEEE 754-2008 standard [7]. Subnormals are flushed-to-zero in TFP.

The architecture of the TFP multiplier (TFP-mul) is sketched in Fig. 1. The significand and exponent bit-widths $m$ and $e$ can be selected for the single operation by setting a 7-bit value in a control register.

The significand path consists of a radix-4 multiplier array with product in carry-save format ($P_3, P_C$) followed by rounding and normalization blocks. The rounding is performed speculatively and in a variable position by selecting a rounding word (RW) depending on the precision $m$ required. The RW consists in a bit vector of zero except the rounding bit set to “1”. It is generated by a decoder based on the precision $m$ required. The decoder also produces the mask necessary to zero the result bits of the significand beyond position $m$. The exponent path is depicted at left in Fig. 1. More detail is given in [12].

The architecture of the TFP adder (TFP-add) is derived from the “double-path” scheme of [13], and it is sketched for the significand path in Fig. 2. Depending on the effective operation (addition or subtraction) and the exponent difference the operation is performed either in the “close” (at left) or the “far” (at right) path. Similarly to the TFP-mul, a decoder provides the rounding word and the mask to implement the TFP operation.

The TFP-mul and the TFP-add are pipelined in two stages to reach a target throughput of 1 GFLOPS. The position of the pipeline registers is indicated by the horizontal blue lines in Fig. 1 and Fig. 2.

For the implementation of the TFP-mul and the TFP-add a 45 nm CMOS library of standard cells by using commercial synthesis tools (Synopsys) has been chosen.

### III. TFP AND NEURAL NETWORKS

The Neural Network (NN) is a mathematical model that consists in a set of layers: input, hidden and output layers. An input layer represents a training or inference data set, a hidden layer generates computations and transfers information to the output layer that produces the results. If a NN consists in many hidden layers, it is called Deep Neural Network (DNN).

These models typically contain a very large number of parameters (weights $w_i$ and bias terms $b_i$) and are usually trained iteratively over vast amounts of data. The NN structure can be represented as in Fig. 3. The training phase consists of feeding data to the network, forward propagating through the whole network, estimating whenever incorrect predictions are made by comparing predictions with ground truth (i.e. a target for the NN), computing and back-propagating weights through the network to minimize the error of incorrect predictions.

An epoch is defined as a full pass over the entire training data set. While the NN training can require hundreds of epochs before reaching the final parameter values, the inference phase consists of a single pass over the entire network. After training, the optimal parameter values are computed and the model is ready to classify new input data.

Fig. 4 shows the architecture for the two-hidden-layers NN, which has been considered as reference example to illustrate the properties of TFP. Moreover, a data set corresponding to a cosine-trend curve with 200 points have been chosen (Fig. 5). The goal is to interpolate the function approximating the distribution of the points.

A TFP simulator consisting in a library of C functions and
implementing TFP operations has been developed in [12]. Each operation is implemented with a standard FP algorithm by (1) limiting the computation of the significand bits to $m$ and (2) applying the specified rounding mode. The simulator executes the algorithm under test in both double-precision and TFP providing the error in key points. The simulator also generates TFP vectors to be used to test the hardware implementation.

A. Training in TFP

The conducted study shows the approximation errors, that depend on selecting a specific precision for the significand in the floating-point representation, for the training and inference stage of a NN. Results for NN training are presented.

Table I reports the training for the test case $\text{cosine}$ for several TFP precisions. The table lists the approximation relative error ($\epsilon_{\text{ave}}$) obtained at the given epoch, the number of the TFP operations executed, the average power dissipation (at 1 GHz for addition, multiplication and total), and the ratio among the total dissipated power for all the considered TFP precisions. The $\epsilon_{\text{ave}}$ has been defined as $|\hat{v} - v|$, where $\hat{v}$ and $v$ are the approximate and binary64 values of the NN computation, respectively.

The trends in Table I show that the power dissipation drops linearly as $m$ is scaled. Scaling $m$ in the NN allows to achieve a good power efficiency as a reduction up to 30% and 50% is reached comparing $m=24$ with $m=11$ and $m=6$, respectively. Table I also shows that the operands precision has an impact on the convergence rate. For $\text{cosine}$ training, the binary32 smallest error is obtained for a large epoch. From $m = 14$ the NN converges very rapidly at epoch 5, and for binary16 we obtain the lowest error. Fig. 5 shows the curves approximated for the case of binary32 (212 epochs) and binary16 (5 epochs). The reason for this behavior is that, when $m$ and $e$ are reduced, small numbers are flushed to zero causing a sort of pruning in the NN. A lower precision may lead to faster convergence, but also to an excessive pruning resulting in the NN not converging.

IV. ERROR CHARACTERIZATION

The neural network of Fig. 4 is used to interpolate the function describing the distribution of the training points. Since for the test case of Fig. 5 the function generating the training points is known, the error can easily be evaluated.

The approximation error is the error due to approximation done by the NN. Referring to Fig. 6, assume there are a number of points (dark “+” in the figure) and we want to detect which ones are within a given distance from the ideal function/curve. This region is delimited in Fig. 6 by the blue lines, and the points which fall in the region in the ideal case of the generating function are marked with a green “∗” (120 points). However, when repeating the inference with the NN interpolated function (curve in magenta), a different set of points, marked with a red “∗” in Fig. 6, lies within the given distance (129 points). By analyzing the results of the experiment, the points correctly identified by the NN lying in the region are 97, i.e., the intersection of the green and red sets. The misclassified points are:

- 23 green “∗” which are inside the region, but not detected by the NN;
- 32 red “∗” which are detected inside the region by the NN, but they are actually outside.

In addition to the approximation error, there is also the quantization error due to the reduction in the precision of
the operations. By repeating the experiment of points lying in the given region for different TFP precisions, we obtain the results reported in Table II. In this case the reference set is the one obtained for inference with binary32 precision.

In Table II, the column marked \((**\cap\ast)\) shows the points detected correctly for the given precision with respect to binary32; the column marked \((\ast)\) reports the points lying in the region for binary32, but not detected in the given precision; the column marked \((\ast\ast)\), vice versa, reports the points detected in the region for the given precision, but which are outside the region for binary32.

The results of the error evaluation done for this specific example, show that even a small variation in precision may lead to a sizeable increase in the overall classification error. Therefore, finely tuning the precision, as in TFP, may significantly improve the results of NN inference with respect to fixed-precision formats.

V. ADVANTAGES OF TFP FOR DEEP LEARNING

The novelty of TFP format is its flexibility. To prove the flexibility advantage studies have been conducted on the inference phase as the inference usually requires lower precision than the training phase [6].

A NN with two hidden layers needs three different parameter levels \((0, 1, 2)\). Consequently, weights and bias terms has been divided in \(w_j^0, w_j^1, w_j^2\) and \(b_j^0, b_j^1, b_j^2\) (Fig. 4). The precision of parameters has been reduced depending on the level belonging to. Each parameter level has been varied by selecting a specific number of bits for significand among 16 or 8 fixing \(e = 8\) in the floating-point representation. The flexibility of the employed TFP format for significand has been reported in the Table III together with results.

Results obtained by using binary32 representation have been also included for comparing with the chosen TFP precision. Table III depicts the quantization error with respect the different TFP precisions, and also shows which parameters configurations can be more power efficient.

The maximum results quality and minimum dissipated power are achieved when \(m^0, m^1, m^2 = 16, 8, 8\). These precisions allow to reduce power dissipation by 30\% (ratio = 0.70) and to detect correctly 98\% (127/129) points respect to binary32. Instead, the largest power reduction (40\%, ratio= 0.60) is reached by exploiting the lowest precisions (i.e. \(m^0, m^1, m^2 = 8, 8, 8\). For the given precisions only the 95\% (123/129) points respect to binary32 are correctly detected.

VI. CONCLUSIONS

Today power efficiency is probably one of the most relevant aspects in the design of embedded digital systems. Consequently, numerous methods have been studied to save power and increase performance by trading power for accuracy. In this paper, the approximate computing technique named Tunable Floating-Point has been proposed to tailor the necessary precision in the parts of a deep neural network to achieve a lower power consumption. The TFP-unit can work at different precisions in the two phases of training and inference of neural networks by simply setting the required precision of operations.

Experiments showed that TFP can be used in a novel flexible mode to save more power at cost of reasonable errors.

Future work will address into adjusting “on-the-fly” (during the execution of the algorithm) the precision of TFP operations when some specific criteria in a neural network are met.

REFERENCES


