Full Approximation of Deep Neural Networks through Efficient Optimization

Cecilia De la Parra
Robert Bosch GmbH
Renningen, Germany
cecilia.delaparra@de.bosch.com

Andre Guntoro
Robert Bosch GmbH
Renningen, Germany
andre.guntoro@de.bosch.com

Akash Kumar
Technological University of Dresden
Dresden, Germany
akash.kumar@tu-dresden.de

Abstract—Approximate Computing is a promising paradigm for mitigating computational requirements of Deep Neural Networks (DNN), by taking advantage of their inherent error resilience. Specifically, the use of approximate multipliers in DNN inference can lead to significant improvements in power consumption of embedded DNN applications. This paper presents a methodology for efficient approximate multiplier selection and for full and uniform approximation of large DNNs, through retraining and minimization of the approximation error. We evaluate our methodology using 422 approximate multipliers from the EvoApprox library, with three different Residual architectures trained with Cifar10, and achieve energy savings of up to 18% surpassing the original floating-point accuracy, and of up to 58% with an accuracy loss of 0.73%.

I. INTRODUCTION

Through approximate computing, hardware requirements in error-resilient applications such as DNNs can be reduced. Approximations at software level are e.g. precision scaling or quantization [1], [2], [3]. At hardware level, approximation of arithmetic units such as multipliers [4] can lead to significant energy savings. In this context, DNNs can be partially or fully approximated. Techniques for partial approximation include retraining [4], [5], and more recently, authors in [6] proposed ALWANN, a framework for partial DNN approximation without retraining. Drawbacks of partial approximation are the energy savings, limited by the amount of approximated elements, and its challenging implementation in generic hardware accelerators. Furthermore, the search for optimal designs is highly time-consuming in traditional simulation frameworks [6]. On the other hand, through specialized methodologies for full approximation of DNNs, a more generic implementation and more energy savings can be achieved. In [7], authors investigate fully approximated and retrained DNNs with quantized weights and FP activations. Methods for optimizing the Approximate Multiplier (AM) itself were introduced in [8], [9], which proved to be effective but time-consuming and with low flexibility, and authors in [10], [11] demonstrated the viability of accuracy recovery through hardware-aware retraining in DNNs for digit recognition. Specialized simulation frameworks such as Concrete [12], based on Caffe [13], have been recently proposed to further accelerate approximate DNN computation in Graphic Processing Units (GPUs) with small time overhead, making the optimization of larger approximate DNNs more attractive.

Considering all currently available approximate hardware designs, selecting the appropriate AM for uniform DNN inference without accuracy loss is a non trivial task, which depends mainly on the availability of resources for DNN retraining. For some applications, such as those with proprietary DNNs, retraining with the original dataset is not possible as it is not available to the public, and therefore, other approaches to overcome this are needed. On the other hand, retraining with all possible design points given all existing AMs is a time exhaustive task even with specialized simulation frameworks. For example, we would need around 1.5 days just to train a ResNet20 [14] with every multiplier from [15] for only one epoch with our specialized simulation framework. With this motivation, we present a methodology for efficient multiplier selection and full approximation of large DNNs without accuracy loss. We focus on designs with the same multiplier for all convolutional and fully-connected (FC) layers, which is desirable for hardware accelerators with uniform processing elements. For this, given an initial set of AMs, we first select the Pareto set of power consumption and Mean Relative Error (MRE), focused on convolutional and FC layers. Through this, we restrict our search to 18% of the initial design space. Then, we retrain only the approximate DNNs with multipliers from this Pareto set. Our methodology allows retraining even without the original dataset, as we can switch to minimizing the propagated approximation error instead of optimizing the DNN accuracy itself.

The novel contributions presented in this work are:

- A data-driven analysis for multiplier selection based on MRE, focused on convolutional and FC layers.
- A methodology for efficient uniform and full approximation of DNNs.
- An alternative to DNN retraining when the original training dataset is not available.
- Significant power improvements without accuracy loss of three residual architectures [14] trained with Cifar10, through simulation of 422 AMs from the EvoApprox library [15].

II. PRELIMINARIES

A. Deep Neural Networks

DNNs are characterized by the use of convolutional and FC layers. These operations can be generalized as follows: \( y = \)
QuantizeApproximateApproximate retrainingAvailable training dataMinimization of propagatedapproximation errorTrain/Test DatasetAMs-Behavioral simulationPre-trained DNNnoPareto-optimal approximate DNNsMREpowerPareto-frontyesAccuracy optimization

\[ \Psi(g(x, w) + b), \] where \( y \) is the layer output, \( g \) is the weighted operation between inputs \( x \) and weights \( w, b \) is the layer bias, and \( \Psi \) is a non-linear activation function, usually a Rectified Linear Unit (ReLU) [16]. To reduce energy consumption, we quantize all weights and activations to int8 precision. In this work, we employ linear, layer-wise quantization [3].

B. Approximate Multipliers

We implement the behavioral model of 422 different 8-bit AMs from the EvoApprox library [15]. As reference for 100% energy consumption, we use the accurate multiplier with the lowest power consumption (mul8_433), and we discard all multipliers with higher power. We define \( f(x, y)_{\text{acc}} \) as the accurate multiplication between inputs \( x \) and \( y \). For an AM, the operation is defined as \( f(x, y)_{\text{approx}} = x \times y + \epsilon \), where \( \epsilon \) is the approximation error, and the corresponding error metrics are formally computed within this range for each possible value. To compare the approximation error of DNN layers. The WMRE is defined as:

\[
WMRE = \frac{1}{n} \sum_{i=0}^{n} \sum_{j=0}^{n} |f(i, j)_{\text{approx}} - f(i, j)_{\text{acc}}| w_i, \tag{2}
\]

In [9], weights \( w_i \) are defined individually for each DNN layer, according to the corresponding weights distribution. To obtain weights which generalize to all DNN layers and also to activations distributions, we instead define our goal as follows: To give more relevance to more probable combinations of \( x = i \) and \( y = j \). Then, \( w_{i,j} \) can be derived from the multivariant distribution \( \mathcal{N}(x, y; \mu, \Sigma) \). We make the following generalizations:

- Weights have Gaussian distribution with mean \( \mu = 0 \) and std. deviation \( \sigma_x \). If the weights mean is not 0, we add an offset to the DNN weights as follows: \( \mu + \text{offset} = 0 \rightarrow \tilde{\mu} = 0 \) to perform this analysis.
- All post-ReLU activations have rectified Gaussian distribution with standard deviation \( \sigma_y \) and probability density functions as follows:

\[
f(y; \mu, \sigma^2) = \Phi \left( -\frac{\mu}{\sigma} \right) \delta(y) + \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y-\mu)^2}{2\sigma^2}} U(y), \tag{3}
\]

where \( \Phi \left( -\frac{\mu}{\sigma} \right) \) is the cumulative distribution function of the normal distribution, \( \delta(y) \) is a Dirac impulse and \( U(y) \) is a unit step [19]. Thus, for \( y > 0 \), the distribution is Gaussian. For the case of \( y = 0 \), we implicitly integrate a formal guarantee of maximum error penalization.

For a parameter-invariant case, we approximate \( \sigma_y \approx \sigma_x = \sigma \). The resulting probability density function is:

\[
f(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4}
\]

Eliminating constants outside the exponent, and substituting (4) in (2), we obtain:

\[
WMRE = \frac{1}{n} \sum_{i=1}^{n} |z_{\text{approx},i} - z_{\text{acc},i}| \exp^{-\frac{1}{2\sigma^2} \beta x^2 + y^2} \tag{5}
\]

Where \( \beta = 0 \) if \( x \times y = 0 \) and 1 otherwise. Although this introduces a strong non-linearity, this also guarantees a maximum error penalization when \( x \times y = 0 \). For our case studies, the Pearson’s correlation coefficient \( \rho \) of WMRE and MRE of all EvoApprox multipliers indicates a
Fig. 3: MRE vs WMRE vs power of EvoApprox Multipliers

linear relationship (see Fig. 3). The ρ of WMRE and ResNet accuracies are between 0.74 and 0.85, indicating that WMRE as well as MRE are indeed affine towards evaluating DNN approximation errors. As the ρ of WMRE and Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) is very low (0.05 and 0.04 respectively), we conclude that MRE is the most suitable metric to generate our Pareto-front of AMs, shown in Fig. 3. The advantage of using MRE against WMRE is that we do not need the distribution values of the examined DNN for its computation.

B. Definition of loss function

After the given pre-trained DNN model is quantized and approximated, we retrain the DNN. For this, we first determine the loss function. In our case studies, we use the cross-entropy DNN for its computation.

Through approximate DNN retraining we impose soft constraints, as the use of weights that lead to approximation errors is not prohibited but has a well-defined penalization, and thus, this steers the weights to their error-free neighborhoods (determined by their MRE), if any, which maintains the proportionality between MRE and DNN accuracy after retraining.

C. Approximate retraining

The corresponding training loss is hereby minimized through stochastic gradient descent (SGD). The gradient update is computed using Straight-Through-Estimators (STE) as in [11], except for the quantization function:

\[ w_{t+1} = w_t + \Delta w = w_t - \nabla C(p), \quad \text{where:} \]

where \( W_q \) are the quantized weights. In the STE the gradient is computed with respect to the accurate operation, as the derivative of approximated operands is undefined. Through this, we compensate simultaneously the quantization and the approximation error.

IV. EVALUATION

A. Simulation framework

For analysis and evaluation we use ProxSim [20] based on CUDA [21], a library for GPU parallel programming, and Tensorflow [22]. For efficient computation of approximate DNNs, we perform dynamic allocation in the GPU to load the behavioral simulation of the corresponding AM. All experiments were performed with an Nvidia GeForce GTX 1080Ti. We implement three residual architectures [14] initially trained with Cifar-10, during 200 epochs with an initial learning rate (lr) of 1e-3, with FP precision and without approximations. The characteristics of these DNNs are described in Table I.

<table>
<thead>
<tr>
<th>DNN</th>
<th>FP Acc.</th>
<th>8b Acc.</th>
<th>#Params</th>
<th>#MAC</th>
<th>Val.time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet8</td>
<td>85.68%</td>
<td>84.61%</td>
<td>78,666</td>
<td>12.5M</td>
<td>12.9 s</td>
</tr>
<tr>
<td>ResNet14</td>
<td>89.41%</td>
<td>89.20%</td>
<td>176,554</td>
<td>26.7M</td>
<td>25.7 s</td>
</tr>
<tr>
<td>ResNet20</td>
<td>91.04%</td>
<td>90.34%</td>
<td>274,442</td>
<td>40.8M</td>
<td>37.9 s</td>
</tr>
</tbody>
</table>

B. Overall results

We test all multipliers from Section II-B in all DNNs from Table I. As in [6], we propose an accuracy tolerance of 1% with respect to the 8b accuracy. The results are plotted in Fig. 4. Note that we only plot the multipliers with accuracy above 80% for clear visualization with respect to the accuracy tolerance, and that the FP accuracy is never surpassed. The Pareto set of MRE vs. power has a mean distance of 5% to the Pareto-front of DNN accuracy vs. power, and thus presents a suboptimal solution of the design space. This Pareto set contains 75 AMs, which is only 18% of the original set. In this work, we keep the whole Pareto set for retraining to validate our approach, but after performing DNN inference
TABLE II: Comparison with ALWANN

<table>
<thead>
<tr>
<th>DNN</th>
<th>#AMS</th>
<th>Train/epoch time</th>
<th>Total time</th>
<th>Energy savings</th>
<th>Acc. [%]</th>
<th>Acc. loss [%]</th>
<th>#AMS</th>
<th>Total time</th>
<th>Energy savings</th>
<th>Acc. [%]</th>
<th>Acc. loss [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet8</td>
<td>75</td>
<td>72 sec.</td>
<td>1.91 hrs</td>
<td>35 %</td>
<td>85.59</td>
<td>0.35</td>
<td>36</td>
<td>2.3 days</td>
<td>30 %</td>
<td>81.56</td>
<td>0.95</td>
</tr>
<tr>
<td>ResNet14</td>
<td>182</td>
<td>260 sec.</td>
<td>5.42 hrs</td>
<td>57 %</td>
<td>88.34</td>
<td>0.86</td>
<td>260</td>
<td>0.7 days</td>
<td>30 %</td>
<td>89.6</td>
<td>1.7</td>
</tr>
<tr>
<td>ResNet20</td>
<td>260</td>
<td>573 sec.</td>
<td>9.91 hrs</td>
<td>53 %</td>
<td>89.6</td>
<td>–</td>
<td>36</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

For ResNet8 and ResNet14, we perform approximate retraining with accuracy optimization for one epoch, for all AMs, to corroborate that our selected Pareto set is still suboptimal after DNN optimization. We use SGD with momentum, lr of 1e-3 and batch size of 512. We find that one epoch is enough to compensate for at least 85% of the lost accuracy due to approximations, making our proposal for approximate retraining less time-consuming than other approaches. As we demonstrate that the initial Pareto front is also suboptimal after retraining, we retrain ResNet20 only with the AMs of the Pareto front. The results are plotted in Fig. 5. With our method, the original FP accuracy can be reached or even surpassed in all DNNs with different multipliers. With a better accuracy compared to the original FP accuracy, we reach energy savings of up to 15%, 18% and 9% in the case of ResNet8, ResNet14 and ResNet20 respectively.

We compare retraining with accuracy optimization and MinPropAE by retraining ResNet8 with the latter approach. The results are presented in Fig. 6. We observe that this method is also effective to recover DNN accuracy and thus is a valid alternative when the original dataset is not available. However, the FP accuracy can not be surpassed.

The comparison of our work with ALWANN [6], with respect to DNN accuracy and retraining/excecution times with approximate multipliers, is presented in Table II. With this results, we demonstrate that retraining with arbitrary AMs is feasible for large approximate DNNs given the proper methodology and simulation tools, contrary to the stated in [6].

V. CONCLUSIONS

In this paper, we propose a novel methodology for efficient multiplier selection and full and uniform approximation of DNNs through retraining, for reducing the energy consumption of multiplications in convolutional and FC layers. Additionally, we present an alternative method for retraining without the original training dataset, specially helpful when approximating proprietary DNNs. Although we are limited by not considering partial approximation, we demonstrate that through effective DNN retraining and specialized tools, we achieve better energy savings in shorter execution times, compared to state-of-the-art frameworks for partial DNN approximation.
REFERENCES