Combining Gradients and Probabilities for Heterogeneous Approximation of Neural Networks

Elias Trommer, Bernd Waschneck, and Akash Kumar. 2022. Combining Gradients and Probabilities for Heterogeneous Approximation of Neural Networks. In IEEE/ACM International Conference on Computer-Aided Design (ICCAD ’22), October 30-November 3, 2022, San Diego, CA, USA. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3508352.3549329

ABSTRACT

This work explores the search for heterogeneous approximate multiplier configurations for neural networks that produce high accuracy and low energy consumption. We discuss the validity of additive Gaussian noise added to accurate neural network computations as a surrogate model for behavioral simulation of approximate multipliers. The continuous and differentiable properties of the solution space spanned by the additive Gaussian noise model are used as a heuristic that generates meaningful estimates of layer robustness without the need for combinatorial optimization techniques. Instead, the amount of noise injected into the accurate computations is learned during network training using backpropagation. A probabilistic model of the multiplier error is presented to bridge the gap between the domains; the model estimates the standard deviation of the approximate multiplier error, connecting solutions in the additive Gaussian noise space to actual hardware instances. Our experiments show that the combination of heterogeneous approximation and neural network retraining reduces the energy consumption for multiplications by 70% to 79% for different ResNet variants on the CIFAR-10 dataset with a Top-1 accuracy loss below one percentage point. For the more complex Tiny ImageNet task, our VGG16 model achieves a 53% reduction in energy consumption with a drop in Top-5 accuracy of 0.5 percentage points. We further demonstrate that our error model can predict the parameters of an approximate multiplier in the context of the commonly used additive Gaussian noise (AGN) model with high accuracy. Our software implementation is available under https://github.com/etrommer/agn-approx.

KEYWORDS

neural networks, approximate computing, energy efficiency

ACM Reference Format:

Elias Trommer, Bernd Waschneck, and Akash Kumar. 2022. Combining Gradients and Probabilities for Heterogeneous Approximation of Neural Networks. In IEEE/ACM International Conference on Computer-Aided Design (ICCAD ’22), October 30-November 3, 2022, San Diego, CA, USA. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3508352.3549329

1 INTRODUCTION

The power consumption of neural networks (NNs) has long been a major obstacle for their deployment inside power-constrained edge devices [37]. Particularly their computational complexity is a concern. Approximate arithmetic units have been proposed by the research community to address these issues [20, 24, 40]: relaxing the constraints imposed on the accuracy of operations enables new optimizations of arithmetic hardware, allowing for improved latency, energy consumption and area usage. Because of the dominant impact of multiplications, most efforts have been focused on improving the performance of multipliers using approximation—a rationale that this work builds upon as well.

Recent findings in the field of quantization demonstrate that optimizing quantization bit widths individually for each layer can provide higher accuracy compared to solutions that uniformly quantize the entire network. While some of these approaches rely on reinforcement learning [7, 36], an alternative route is the optimization of quantization parameters using gradient-based methods during network training [19, 38, 39]. To the best of our knowledge, the only work that investigates a non-uniform (i.e. heterogeneous) approach to approximation of NNs is Mrázek et al. [25], which uses a multi-objective genetic algorithm. By relying on behavioral simulation to evaluate a large number of candidate solutions, this method can not perform retraining, as it would render the already lengthy search procedure computationally intractable. Recent work by De la Parra et al. [3] demonstrates that solutions employing a single approximate multiplier (AM) throughout the entire network can outperform heterogeneous solutions if lost accuracy is recovered using retraining. Naturally, this raises the question whether performance could be improved even further if heterogeneous approximation and retraining were combined. It is obvious that this requires a different approach to finding a heterogeneous multiplier configuration in order for the search procedure to be feasible. To address this problem, we propose a gradient-based search algorithm that is capable of finding high-quality heterogeneous AM configurations. The search results in only a single candidate solution (or a small set acquired by varying a hyperparameter) that can easily be retrained. The key novel contributions of this work are:

- An efficient method for jointly determining the robustness to approximate multiplications for each NN layer during training. Through transformation into a more favorable solution space, our algorithm allows for a fast traversal of the heterogeneous multiplier assignment problem in a NN.
- A probabilistic error model that can give a precise estimate of the performance of an AM in a NN layer in terms of recoverable and non-recoverable error. Besides knowledge of
the difference between the accurate and approximate multiplication results for all operand combinations, no behavioral simulation is required. The model is data-driven and does not assume any particular distribution of input operands. This makes its performance agnostic to methods that impact these distributions such as pruning [10, 33], quantization [41], etc.

Our evaluation of several convolutional neural network (CNN) models on the CIFAR-10 and Tiny ImageNet datasets shows that our method consistently manages to push the boundary of energy efficiency and network performance for various networks. We also improve upon existing models that express AM properties as additive Gaussian noise (AGN) parameters. This boosts the accuracy of AGN as a faster and simpler replacement for behavioral simulation of AMs.

2 BACKGROUND AND RELATED WORK
Due to the significant complexity of accurately simulating AMs during the training procedure of NNs [4, 35], several works propose the use of random noise as a replacement for the inaccurate computations. Hammad et al. [9] use a model based on the multiplier’s Mean Relative Error (MRE) to generate random data which perturbs the output of an accurate computation. To enable retraining without the need for hardware simulation, De la Parra et al. [5] propose a data-driven noise model with higher granularity. The model constructs a noise tensor individually for each neuron by observing the approximation error on sample data. Similar to behavioral simulation, this model is not generalizable across AM instances because it only captures the dynamics of the AM it was constructed for.

Noisy intermediate results have also been used to demonstrate that the robustness of a NN is not uniform but varies for different layers [2]. These findings are corroborated by Hanif et al. [11], who determine the individual robustness of layers to approximation by injecting AGN into individual layers and observing the change in accuracy. By optimizing one layer at a time, the works do not take into account interdependencies between the robustness of layers. It also lacks a method that connects the robustness of a given layer to a concrete hardware instance. A similar method in the context of Capsule Networks is discussed by Marchisio et al. [21]. The proposed model for connecting a layer’s robustness to noise with AM instances requires Monte Carlo (MC) simulations to be carried out for each combination of layer and AM. Other models that describe the error incurred by a single approximate multiplication were put forward by Mazahir et al. [22] and Ullah et al. [34]. These models, however, do not consider the compounding effects of multiple operations in a neural network.

To leverage the varying robustness of layers Mrazek et al. [25] demonstrate the use of a multi-objective evolutionary algorithm as a means to tackle the large search space of heterogeneous multiplier assignment in a NN. This solution relies on the evaluation of numerous candidate solutions, requiring the use of a weight remapping scheme rather than retraining to recover the degraded accuracy. Even without retraining, the vast number of simulations makes the method prohibitively slow for non-trivial networks. The same is true for more recent approaches that optimize the network architecture itself for use with AMs [27].

3 PROPOSED METHODOLOGY
We provide an analysis of the aggregate error at a neuron’s output and conclude that AGN is a meaningful surrogate model for behavioral simulation of AMs. Using the properties of this model, we demonstrate how to simultaneously optimize the amount of AGN across the entire network, considering the complex interactions between perturbations in different layers. An error model is developed in order to make the abstract AGN parameter comparable with the computation errors exhibited by concrete AM instances. Using the learned robustness parameters and the error model, we can determine which AMs will produce the required accuracy and match appropriate AM instances based on each layer’s individual sensitivity.

3.1 Modeling approximate multiplication as noise
The error imposed by a single approximate multiplication can be considered additive to the output of an accurate multiplication

\[ f(x, w) = x \cdot w + e(x, w) \]  

where \( e(x, w) \) is an error function that is unique to each AM instance. For a NN application we are, however, not concerned with the error of each individual operation, but with the aggregate error over several multiplications. With the definition of the pre-activation output of an accurate neuron

\[ y = \sum_{i=1}^{n} x_i w_i + b \]  

Equation (1) can be substituted to obtain the output of the same neuron using approximate multiplication as:

\[ \hat{y} = \sum_{i=1}^{n} x_i w_i + e(x_i, w_i) + b \]  

\[ = \sum_{i=1}^{n} x_i w_i + b + \sum_{i=1}^{n} e(x_i, w_i) \]  

Assuming that \( x \) and \( w \) exhibit sufficiently random properties, we conjecture that, as \( n \) grows, the distribution of the aggregate error will converge to a normal distribution, thus:

\[ \hat{y} \approx \sum_{i=1}^{n} x_i w_i + b + \mathcal{N}(\mu_e, \sigma_e) \]  

\[ = \sum_{i=1}^{n} x_i w_i + b + \mu_e + \sqrt{\sigma_e} \cdot N(0, 1) \]  

Furthermore, the systematic portion of the error \( \mu_e \) will be absorbed by the bias or subsequent batch normalization for any non-degenerate case when the network is retrained to match the approximate configuration [31] s.t. \( b' = b - \mu_e \). To simulate the effect of approximation on a fully retrained network, we can therefore assume that \( \mu_e = 0 \).

Using this surrogate model instead of a behavioral simulation of the approximation error has two important benefits for the search procedure: First, AGN can easily be constructed using primitives that are available in most common deep learning toolkits and does
Because this formulation is continuous and differentiable, it is possible to optimize all layer perturbations simultaneously—just like all other network parameters—using backpropagation. Computation errors in an early layer can impact the overall accuracy disproportionately as they might alter the results in all subsequent layers; these complex dynamics of propagating errors are already captured by this model because of the chained computation of gradients. The loss function is evaluated based on layers being perturbed by a certain amount of AGN, resulting in model convergence towards higher robustness to small perturbations. Through the added noise, the output of each individual neuron will be less reliable. Information relevant to the model’s task will therefore have to be spread out over more neurons when propagating through the network, similar to the popular Dropout regularization [30]. We assume that this increased robustness will also be beneficial when deploying AMs: because the model has learned to be less reliant on the precise output of individual neurons, it will be able to handle slight deviations in the intermediate results caused by AMs better.

Only using the task loss to drive model optimization is not sufficient, however, since \( \sigma_l \) could always be driven towards zero. To avoid this, we introduce an additional noise loss \( L_N \) that incentivizes the optimizer to explore solutions with higher perturbation. The noise loss takes into account the current values of \( \sigma_l \) as well as the relative cost of each layer \( \epsilon_l = \epsilon(l)/\sum_{l \in L} \epsilon(l) \). The relative cost scales the importance of the amount of perturbation in each layer; it is clear that we care most about high values of \( \sigma_l \) in layers with a high complexity, while it is preferable to allow for higher relative accuracy in layers that do not contribute much to overall resource consumption. We choose the amount of multiplications in a layer as an easy to implement cost function \( \epsilon(l) \). The additional noise loss solves the problem of the optimizer converging to solutions with no AGN, but it creates a new one: The optimizer could now decrease the total loss indefinitely by adding ever-growing amounts of noise to the intermediate results, making the task loss insignificant. We avoid this by upper-bounding the maximum allowable noise loss to some fixed value \( \sigma_{\text{max}} \). This gives us the total noise loss as

\[
L_N = - \sum_{l \in L} \min \{|\sigma_l|, \sigma_{\text{max}}\} \cdot \epsilon_l \tag{10}
\]

The overall loss \( L \) is then simply the weighted sum of task and noise loss

\[
L = L_T + \lambda \cdot L_N \tag{11}
\]

where \( \lambda \) is a hyperparameter that determines the relative importance of network accuracy and perturbation. Equation (10) can be differentiated w.r.t. \( \sigma_l \) as

\[
\frac{\partial L_N}{\partial \sigma_l} = \begin{cases} 
-\epsilon_l, & |\sigma_l| \leq \sigma_{\text{max}} \\
0, & \text{otherwise}
\end{cases} \tag{12}
\]

After optimizing the amount of injected noise per layer we arrive at a configuration where each layer’s learned robustness to noise \( \sigma_l \) has been tuned to maximize the amount of overall AGN throughout the network while minimizing the degradation of accuracy. Each layer’s \( \sigma_l \) can be considered a proxy for the actual (but much harder) optimization task: determining how sensitive the network’s overall performance is to inaccurate computation results in each layer.
3.3 Probabilistic Multi-Distribution Error Model

An important question that arises is how the abstract optimization factor $\sigma_l$ in the AGN model relates to the error produced by a given AM. In order to link both, we propose the use of a probabilistic model that treats each error as a random event with certain probabilities; instead of truthfully simulating the error function for each pair of input values, its output can be considered a discrete random variable $Z = e(x, w)$ that maps the outcome of randomly selecting $x$ and $w$ to the error produced by these operands. Determining the standard deviation of $Z$ makes it comparable with the learned perturbation factor $\sigma_l$ that simulates the non-recoverable portion of the approximation error in the AGN space. Assuming the commonly used 8-bit Integer multipliers, both $x$ and $w$ can take $2^8 = 256$ distinct values for their sample spaces $\Omega_x$ and $\Omega_w$. In total, there are $256^2$ possible combinations of input values in the joint sample space $\Omega_Z = \Omega_x \times \Omega_w$. For any multiplier of interest, we need to know the error (i.e. the difference between accurate and approximate output) for each of these input combinations, from hereon referred to as the multiplier’s error map. This is the only part of the method that requires at least a high-level model of the hardware to be simulated. If simulating the hardware is expensive, this can be done once and the resulting error map stored for later use.

Next, we need to consider that not all 256 values that $x$ and $w$ might take are equally likely to appear. The relative frequencies of values in the weights tensor and an input sample can be used to model their probability distributions $p_x(x)$ and $p_w(w)$. In practice, this means building a histogram with the count of each possible 8-bit Integer value for both tensors and normalizing it to one. The data-driven construction of operand probabilities means that no assumptions have to be made with regards to their underlying distribution. Given the error for each combination of values and their respective likelihoods, mean and standard deviation of the error can be estimated as

$$\mu_Z = \sum_{x \in \Omega_x} \sum_{w \in \Omega_w} p_x(x) \cdot p_w(w) \cdot e(x, w)$$

$$\sigma_Z^2 = \sum_{x \in \Omega_x} \sum_{w \in \Omega_w} p_x(x) \cdot p_w(w) \cdot (e(x, w) - \mu_Z)^2$$

which describes the mean and standard deviation of the error for a single multiplication. From Equation (2), it is clear that the neuron output is the sum over $n$ multiplications. All other factors being equal.

So far, we have assumed $x$ and $w$ to be independent and identically distributed. Empirically validating this assumption shows that it holds well for weights, but not for activations. Intuitively, this can be explained by a higher amount of local correlations in the activations. To make this clearer, we can imagine a black and white image passing through a convolutional layer: because pixels of the same color tend to be grouped together, the individual patches are more likely to be all-black or all-white than the global distribution of pixels in the image would suggest. More concisely, the local distribution of feature values can deviate strongly from their global distribution.

The problem of diverging local and global distributions of operand values can be addressed through sampling: mean and standard deviation are first calculated for several samples of the local distribution. The sample is drawn from the receptive field of a neuron in the target layer, i.e. a sample is either a randomly selected input feature vector for Fully-Connected (FC) layers or a patch for convolutional layers. These individual observations are then integrated into an estimate of the global distribution’s mean and standard deviation as shown in Figure 2. We randomly sample $k$ vectors from the layer’s input activations. For each input vector, a frequency distribution $p_x(x)$ is generated. Equations (13) and (14) are then used to calculate its mean and standard deviation. As an additional benefit, this removes the need to build a global histogram of the input value distribution in favor of building multiple distributions on very small input samples. Calculating the combined mean of the local distributions is simple. Special care needs to be taken when combining group standard deviations: Our method must account for the effect of different means in each sample on the combined
standard deviation [17].

\[
\mu_Z = \frac{1}{k} \sum_{i=1}^{k} \mu_{Z_i} \\
\sigma_Z^2 = \frac{1}{k} \left( \sum_{i=1}^{k} \left( \sigma_{Z_i}^2 + \mu_{Z_i}^2 \right) - \frac{1}{k} \left( \sum_{i=1}^{k} \mu_{Z_i} \right)^2 \right)
\]

The degree with which this method is applicable is determined by the fan-in \( n \) of the respective layer, as convergence of the error towards a normal distribution depends primarily on the fan-in. If layers with a very low fan-in were present, some care would have to be taken to confirm that the AGN model is still valid.

3.4 Multiplier Matching

Our method focuses on Integer AMs with a low bit width which are typically used in low-energy inference settings like edge devices and accelerators. Because of the small amount of operand combinations, a full error map should normally be easy to obtain. To increase the degrees of freedom of the solution, having a large set of different AMs that cover a wide range of accuracy and resource consumption is desirable. As long as these conditions are met, our method is applicable to many different AMs designs.

With an optimized configuration of \( \sigma_l \) as a measure of a layer’s sensitivity, we can match an appropriate AM to each layer. For every AM in our search space, we calculate an estimate of the error it produces using the method described in detail in Section 3.3. The estimate produced by the error model is directly comparable to the learned \( \sigma_l \). The multipliers for which the error lies above the accuracy threshold \( \sigma_l \) are discarded since they do not produce the required accuracy. From the remaining set of multipliers that produce sufficiently accurate results, we can pick the one that optimizes another metric of interest.

4 RESULTS AND DISCUSSION

Unless stated otherwise, all experiments were carried out on a host system equipped with an AMD Ryzen 9 3900X CPU and a single nVidia RTX 2080Ti GPU. We implement our search algorithm [32] as an extension of the popular deep learning framework PyTorch [26].

4.1 Multi-Distribution Error Model

In order to assess the accuracy with which our error model can infer the standard deviation of the approximation error on the layer’s output, we evaluate its performance on the layers of a ResNet8 model. The model is trained using the parameters laid out in Section 4.2. We do not explicitly evaluate the estimate of the error mean, as it is less relevant to our method for the reasons discussed in Section 3.1. From Equation (14), it should be clear that an accurate estimate of the error mean is a necessary condition for an accurate estimate of the standard deviation. The ground truth for the approximation error is obtained through a behavioral simulation of all 13 unsigned multipliers from the EvoApprox library [23]. The simulation generates the approximated pre-activation output of each layer, given its weights and activations. We compare the measured error’s standard deviation to the estimate of the standard deviation generated by our probabilistic multi-distriution error model for \( k = 512 \) input samples. The results are also compared to the single-distribution MC method discussed in [21]. We further include the respective multiplier’s MRE in the evaluation as it is a commonly proposed [3, 9] single-value metric, used as an indication of multiplier performance. The results of our evaluation can be found in Table 1.

We find that the results of the Single-Distribution MC method are very similar to the results produced by using our probabilistic method without accounting for differing local distributions, i.e. based on the global frequency distribution of activation values alone. Single-Distribution MC and our probabilistic model converging to similar results in this scenario is very plausible, given that the former method is a MC simulation of exactly the same process that is analytically described in the latter. This suggests that taking local divergence of the input activations into account provides a significant boost in the predictive performance of our model, compared to methods that assume the global distribution to be present for all operations. The observed values for the standard deviation occupy a very wide numerical range of approximately 5 orders of magnitude. On such a large range, a median relative error of 4.6 % is negligible, which is reflected in the almost perfect correlation of 0.997.

Our evaluation shows that the predictive performance of the MRE for the error’s standard deviation is very limited. This can be explained by the fact that the MRE is only a metric of a multiplier performance over the entire numerical range and for operands with equal probabilities. It does not distinguish between systematic and non-systematic portions of the error and can include neither knowledge about the different likelihoods of operands in NN models nor the impact of a layer’s fan-in on the aggregate error at the layer output.

4.2 ResNet on CIFAR-10

In line with previous works [3, 25], our method is evaluated on several sizes of the ResNet [12] architecture, trained on the CIFAR-10 dataset [15] with 36 8-bit unsigned multipliers from the EvoApprox library [23] as the AM search space. Relative power numbers were generated using the pdk45_pwr property of the EvoApprox multipliers, normalized to the power consumption of the reference accurate multiplier and weighted with the number of multiplications in each layer.

The training and augmentation procedure laid out in the original work is used to generate a floating-point reference model. Next, Quantization-aware Training (QAT) is carried out to obtain a baseline model that is quantized to 8 Bit. On the quantized baseline,
Table 2: Comparison of energy reduction and accuracy loss for different methods

| Model  | Method       | Energy Reduction | Top-1 Accuracy Loss [p.p.] |
|--------|--------------|------------------|----------------------------|
| ResNet8| ALWANN [25]  | 30%              | 1.7                        |
|        | Uniform Retraining [3] | 58%              | 0.9                        |
|        | Gradient Search (ours) | 70%              | 0.5                        |
| ResNet14| ALWANN [25] | 30%              | 0.9                        |
|        | Uniform Retraining [3] | 57%              | 0.9                        |
|        | Gradient Search (ours) | 75%              | 0.9                        |
| ResNet20| LVRM [31]   | 17%              | 1.0                        |
|        | Uniform Retraining [3] | 53%              | 0.7                        |
|        | Gradient Search (ours) | 71%              | 0.9                        |
| ResNet32| Gradient Search (ours) | 79%              | 0.8                        |

we uniformly initialize each layer’s perturbation factor $\sigma_l$ to 0.1 with a cap at $\sigma_{\text{max}} = 0.5$. Layer perturbation factors and other network parameters are then jointly optimized for 30 epochs using the Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of $1 \times 10^{-2}$ and a decay of 0.9 after every 10 epochs. After matching AMs with appropriate accuracy to each layer, we retrain the entire network using behavioral simulation of the selected AMs for another 5 epochs with an initial learning rate of $1 \times 10^{-3}$ and a decay of 0.9 after every other epoch. During this phase, the Straight-Through Estimator (STE) [1, 13] is used to derive valid gradients for the AMs. We repeat the Gradient Search and retraining several times while varying the $\lambda$ parameter between 0 and 0.6 in steps of 0.05. By adjusting the value of $\lambda$, we can generate several solutions, each of which strikes a different balance between accuracy and perturbation (and thus, by extension, energy consumption).

In the comparison of results in Figure 3 it can be seen that the accuracy is above the baseline for all ResNet variants for an energy reduction of up to 45%. We attribute this to the perturbation acting as a form of learnable regularization; since all parameters are trained in the Gradient Search phase, the network converges towards a configuration that is both more resilient to approximation and generalizes better to unseen data. The drop in accuracy for higher degrees of approximation becomes increasingly steep for deeper models, most likely due to the accumulating effect of the approximation error.

The comparison of the accuracy in the AGN space (‘AGN Model’) with the accuracy that is achieved after retraining using behavioral simulation (‘Approx., Gradient Search Weights’) in Figure 4 reveals another interesting property: the accuracy of a retrained model under approximation is very similar to the accuracy exhibited by the model that is perturbed using AGN up to energy savings of around 60%. For more aggressive approximation, the relationship between both values deteriorates up to a gap of several percentage points. These models suffer from a significant degradation of accuracy under approximation, while the same is not true for the same model perturbed with comparable amounts of AGN. This can be interpreted as a shortcoming of the AGN model: AMs with very low energy consumption are more likely to produce localized error patterns which are insufficiently captured by the AGN model. The AGN model assumes an even spread of errors over all neurons. It is likely that a structured error has a more adverse effect on the propagation of information for some specific connections, widening the gap between predicted and achieved accuracy.

Figure 4 also provides some insight into the question of whether training in the AGN space has a positive carryover to model accuracy when deploying AMs. To make the impact of training with AGN on the achieved accuracy visible, we repeat the retraining using behavioral simulation based on the weights and biases of the baseline model (‘Approx., Baseline Weights’) instead of the weights and biases learned during the Gradient Search phase. We can see that the models trained using AGN consistently achieve higher accuracies after retraining using behavioral simulation compared to models that directly attempt approximate retraining on the baseline model. This indicates that models trained using AGN have converged to a configuration that is more robust to the errors produced by approximation.

A comparison of the loss of accuracy and reduction in energy consumption to other state-of-the-art methods can be found in Table 2. In addition to improved energy efficiency, our algorithm adds little
To demonstrate the scalability of our method, we apply it to the popular ADAM optimizer [14] does not produce satisfactory results when trading off energy savings and loss of accuracy. Given the more complex classification task, we do not find an improvement over the accuracy of the baseline model anymore, both in the AGN space as well as after deployment of AMs and retraining. The much lower reduction of energy consumption when using signed multipliers can be attributed their lower overall energy reduction for similar performance as well as the smaller search space of only 13 available signed 8-bit multipliers.

For the unsolved solution, the search procedure identifies 13 different AMs from the search space. The comparison of layer complexity and the respective reduction in energy consumption per layer in Figure 5 shows that the Gradient Search and subsequent multiplier assignment deploys the highest degrees of approximation on the network’s inner layers with high amounts of multiplications. In contrast, particularly the first and last layers are assigned highly accurate hardware instances. A relatively high sensitivity of these layers is in line with common heuristics in non-uniform quantization schemes [36].

We confirm that this effect is consistent by repeating the experiment with other CNN models. Performing the same optimization procedure on AlexNet [16] and MobileNetV2 [28] models yields similar results: Inner layers with higher computational complexity are assigned less accurate hardware instances with up to 84.4% reduction in energy consumption. In contrast, highly accurate AMs with a reduction between 1.3% and 5.4% are mapped to the first and last layers of each model.

5 CONCLUSION

In this work, we have demonstrated the feasibility of combining heterogeneous approximation and retraining for NNs. The use of AGN as a surrogate model for behavioral simulation is motivated from the mathematical formulation of NNs. Our findings suggest that there is a strong connection between perturbation of neural networks with AGN and low to medium degrees of approximation. This connection, however, weakens for higher degrees of approximation. Whether the AGN model can be adapted to capture more structured errors without sacrificing generalizability remains to be answered by further research. Despite this, we show that the trade-off between energy and accuracy can be improved consistently and significantly by considering the varying robustness of different parts of a NN during the deployment of AMs. As a means to tackle the enormous search space of heterogeneous AM assignment, the flexibility of the AGN model is used to abstract over many different AMs with only a single parameter. This parameter is differentiable, so it can be learned using backpropagation. Learning the different degrees of robustness to errors during network training is the key to efficiently exploring different solutions, especially for deeper networks. To connect AGN to concrete hardware, we introduced a probabilistic model of the approximation error. This model allows for an accurate prediction of the noise properties which the AM error will exhibit on any given layer without having to rely on time-consuming simulations. Through combining these methods, we provide a path towards NN hardware that makes inference both accurate and power-efficient.
REFERENCES

[1] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. CoRR abs/1308.3432 (2013).

[2] Nicholas Cheney, Martin Schirmpf, and Gabriel Kreiman. 2017. On the Robustness of Convolutional Neural Networks to Internal Architecture and Weight Perturbations. CoRR abs/1703.08245 (2017).

[3] Cecilia De la Parra, Andrej Guntoro, and Akash Kumar. 2020. Full Approximation of Deep Neural Networks through Efficient Optimization. In IEEE International Symposium on Circuits and Systems, ISCAS 2020, Sevilla, Spain, October 10-21, 2020.

[4] Cecilia De la Parra, Andrej Guntoro, and Akash Kumar. 2020. ProcSim: GPU-based Simulation Framework for Cross-Layer Approximate DNN Optimization. In 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020.

[5] Cecilia De la Parra, Andrej Guntoro, and Akash Kumar. 2021. Efficient Accuracy Recovery in Approximate Neural Networks by Systematic Error Modelling. In ASPDAC’21: 21st Asia and South Pacific Design Automation Conference, Tokyo, Japan, January 18-21, 2021.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA.

[7] Ahmed T. Elbakhel, Prannoy Pilligundla, Fatemehsadt Mireghallahri, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. 2020. ReLoQ: A Reinforcement Learning Approach for Automatic Deep Quantization of Neural Networks. IEEE Micro 40, 3 (2020), 37–46.

[8] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. Deep Learning. MIT Press.

[9] Issam Hammad, Kamal El-Sankary, and Jason Gu. 2019. Deep Learning Training and Benchmarking of Approximation Methods. In EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD 2016, Austin, TX, USA, November 7-10, 2016.

[10] Vojtech Mravec, Zdenek Vasichek, and Lukas Sekanina. 2021. Learnable Companding Quantization for Accurate Low-Precision Neural Network Inference. In ACM International Conference on Computer-Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021.

[11] Mohammad Abbah, Rehan Hafiz, and Mohammad Shafique. 2018. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada.

[13] Muhammad Abdullah Hanif, Rehan Hafiz, and Muhammad Shafique. 2018. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018.

[14] Xin He, Liu Ke, Wenyan Lu, Guihai Yan, and Xuan Zhang. 2018. AxTrain: Hardware-Oriented Neural Network Training for Approximate Inference. In Proceedings of the International Symposium on Low Power Electronics and Design, ISLPED 2018, Seattle, WA, USA, July 23-25, 2018.

[15] Diedrek P. Kangna and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[17] Alex Krizhevsky, Geofrrey Hinton, et al. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems.

[18] Ingo Lange. 2018. Standardabweichungen und Kovarianzen über mehrere Mittelungsintervalle. https://wettermast.uni-hamburg.de/Downloads/StdAbwKovIntervalle.pdf.

[19] Ya Le and Xuan S. Yang. 2015. Tiny ImageNet Visual Recognition Challenge.

[20] Uros Lotric and Patricio Bulic. 2012. Applicability of approximate multipliers in computers.

[21] Ya Le and Xuan S. Yang. 2015. Tiny ImageNet Visual Recognition Challenge.

[22] Sana Mazahir, Osman Hasan, Rehan Hafiz, and Muhammad Shafique. 2020. ReD-CaNe: A Systematic Methodology for Resilience Analysis and Propagating Gradients Through Stochastic Neurons for Conditional Computation. In 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020.

[23] Kohei Yamamoto. 2021. Learnable Companding Quantization for Accurate Low-Bit Neural Networks. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021.

[24] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. LQ-Nets: A Provably Cycling Gradient Network. IEEE Transactions on Automatic Control, 63(9): 3141-3153.

[25] Lingfei Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2017. EvaproXpress: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017.

[26] Vojtech Mravec, Radek Hribacek, Zdenek Vasichek, and Lukas Sekanina. 2017. EvaproXpress: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017.