Comparing Reduced Bitwidth Representations for Machine Learning - Quantization vs Brain Float

Anson Tsai, Ryan Lund

University of California, Berkeley

***Abstract*—Machine learning (ML) has rapidly become an essential component of modern commercial products. As such, researchers have sought to optimize both software and hardware in order to support faster and more energy efficient training and inference, both on the warehouse scale and for mobile devices. A recent area of focus is to use reduced bit-width numerical representation for weights and activations. One technique to reduce bitwidth is quantization, or using narrow integers along with a scaling factor in the place of a wider float value. An alternative technique is to use a novel numerical representation, such as the 16 bit bFloat format (bf16). With 1 sign, 8 exponent, and 7 explicit mantissa bits, bf16 is able to preserve the approximate dynamic range and model accuracy [1] of a 32-bit floating point (fp32) network while consuming half the space. In this project, we will compare the impacts on accuracy, performance, and resource utilization of these reduced bitwidth methods by running a testbench of ML workloads on a Gemmini [2] generated systolic array coupled with Rocket Core in-order [3] and BOOM [4] out-of-order processors.**

I. Introduction

A. Current Constraints

Within the next five years, the global machine learning market is expected to reach nearly $100 billion in value. [5] This market is not confined to a single segment, and the range of applications for ML mirrors the size of the industry. On one side of the spectrum is warehouse scale computing with use cases such as big data analysis, recommendation systems, and user segmenting. Here, algorithms are run on massively parallel server farms, with a vast amount of computing resources at their disposal. In sharp contrast to this is the other side of the ML spectrum, mobile computing, where use cases such as virtual assistants and augmented reality programs must run on resource limited platforms. In spite of the enormous differences in scale, both sides of the ML market find themselves limited by similar constraints. These constraints can be succinctly broken down into three categories: latency, power, and accuracy.

For warehouse scale systems, load times, often driven by neural network latency, can have a significant impact on revenue. [6] Similarly on mobile and edge devices, decreased latency improves the user experience of a product. With regards to power, warehouse scale computers are limited by the amount of cooling capacity for a given server farm, which is often not improved as quickly as the underlying computer hardware. Mobile systems suffer from the same limitation, which is further exacerbated by the finite capacity of a battery. Finally, for both scales, accuracy is a primary goal for maximizing user experiences and revenue. Consider the case of Netflix, where 75% of views are driven by recommendations [7].

B. Existing Optimizations

With the issues of latency, power, and accuracy affecting all ML workloads, a significant amount of research has been performed to optimize these factors. This exploration led to the emergence of new network architectures, such as convolutional neural networks (CNNs) and long short-term memories (LSTMs). More recently, research has focused on mobile optimized networks, such as MobileNet, SqueezeNet, and DenseNet, which feature increased weight density and computational efficiency.

Today, many ML workloads are run on CPUs or GPUs. Yet, even as network architectures have evolved, the core computational kernels remain largely the same. This has lead to the rise of research into domain specific hardware accelerators for deep learning. These accelerators can be constructed via FPGA based designs such as those used for Microsoft Asure, or ASIC based designs such as Google's Tensor Processing Unit (TPU).

Generally speaking, these different forms of NN accelerators share a common base architecture designed around matrix multiplication: the systolic array. This architecture is composed of numerous multiply accumulate cells (MACs), which act as the unit level building block of the larger array. The array itself is laid out in a square, with each individual MAC receiving input from the MACs to the north and west of itself, and passing output to the MACs to the south and east of itself. On the highest level, activation data is passed to the north side of the array, weight data is passed to the west side of the array, and after propagation output is produced on the south side of the array for further accumulation. It has been shown that increasing the dimensions of the systolic array either by changing the number of MACs or the bitwidth of data can increase power consumption by up to 3.4x and area by up to 2.3x. [2] However, increasing the number of MACs in the array has also been shown to increase performance by up to 4x for ML inference.

With these metrics in mind, it seems that a logical path to increase performance without increasing resource consumption would be to increase the number of MACs in a systolic array while simultaneously decreasing the bitwidth of data. However, this approach requires the assumption that tech-

niques exist to achieve similar accuracy between fill bitwidth and reduced bitwidth computation, all else constant. Research has focused on two main approaches to achieve this result: quantization and new numerical representation formats.

C. Quantization

Quantization is the processing of taking a set of floating point values (typically fp32) and converting them into a lower bit-width representation. Typically, this process can be thought of as generating a step function that maps the range of possible values for fp32 to the range of possible values for the smaller representation. In practice, this mapping carried out by generating a scaling factor and then performing a clamp and round on the scaled values. Research has shown that neural networks can be quantized to 8-bit integers (int8) with almost no loss in accuracy [8] and is currently focused on achieving the same result with 4-bit integers [9].

There are dual benefits to quantization into small integers. First, by reducing the bitwidth of data values, less hardware is required for each MAC, which means that computation can run faster and consume lower power. This improvement is also shared by bf16, although to a lesser extent due to the larger size of bf16 when compared to int8. The second benefit to quantization is that MACs can implement integer only arithmetic [8]. As integer arithmetic hardware often consumes lower area than the corresponding floating point hardware, this results in additional resource savings. Note that these savings are not shared by bf16.

For these benefits, there are some downsides to this method of bitwidth reduction. First, enacting quantization requires that a programmer create quantization methods and calculate a chain of scaling factors across multiple layers of a network. This requires coding effort but should have minimal overhead from the perspective of the workload itself. More importantly, quantized networks can suffer from accuracy degradation due to the lost of expressible range for weights and activations. However, for most networks this loss can be minimized through careful consideration of outlier weights if converting a fp32 network to int8, or by training the entire network in int8 to begin with.

D. New Numerical Representation Formats - bFloat16

Another method that researchers have developed to exploit the advantages of reduced bitwidth is to use new numerical representation formats. These formats are designed to meet the specific needs of machine learning while minimizing the number of bits consumed. Of note is the brain floating point format pioneered by Google [1].

Similar to quantization, researchers realized that neural networks rarely need highly precise weight values to achieve high accuracy. However, instead of moving to small integers and giving up range of fp32, bf16 acts as a hybrid between 16 bit floating point and 32 bit floating point that preserves the range of the 32 bit value in the footprint of a 16 bit value at the expense of precision.

This is achieved by having the same number of exponent bits in bf16 as in fp32, obtaining the space for these extra bits by having less mantissa bits in bf16 than in traditional fp16. With fewer mantissa bits, the floating point multiplier is smaller, which saves additional hardware area beyond the savings from reduction in overall bitwidth. However, this also means that custom hardware is needed to support operations with a non-standard number of exponent and mantissa bits. We will go more into the costs and benefits of bf16 in the ”Comparisons of State of the Art” section of this paper.

II. Proposed Techniques

A. Background

Our experiments consist of running a series of testbench ML workloads using differing numerical formats and comparing the resulting model accuracy, performance, and resource utilization via memory usage, hardware area, and power consumption.

For the testbench ML workload, we will create a custom test based on the CIFAR-10 [10] dataset. The CIFAR-10 dataset consists of 60000 32x32 images that belong to one of ten possible categories (6000 images per class). The goal for a trained model is to maximize the number of correctly classified test images (1000 per class). For these tests, we will use a modified version of the LeNet-5 [11] network. To ensure consistency, all models will be trained for a set number of epochs before classification begins. Furthermore, if the need arises, we will extend our workload by utilizing pre-written gemmini-rocc tests [12].

In order to run this testbench, we will be utilizing the Rocket and Boom processing cores coupled with a Gemmini generated systolic array. These cores are all publicly available via Chipyard [13], an open source framework for agile development of Chisel-based SOCs. That being said, Gemmini and RISC-V currently lack support for the bf16 numerical format. This means that we will need to add support for bfloat operations into Gemminmi. We will also need to find a workaround to pass bfloat values into our systolic array, which will likely mean packing the bfloat values as a integers.

In order to gather data about the physically implemented design, we will be using Hammer [14], a framework for building physical design generators, along with the ASAP7-PDK.

B. Data Collection

For the purposes of our paper, we will be measuring model accuracy, performance, and resource utilization in terms of both power and area. Here, accuracy is a measurement of the classification accuracy of each model. The performance metric will be largely based on latency, or the time the trained model takes for a single classification. To best visualize performance, we will create an accuracy vs latency graph. Several models will be trained to make predictions within $X, 2X, 3X...$ units of time, allowing us to understand the impact of latency goals on the models. This is similar to the model accuracy metric

explained above with the exception that the model accuracy tests are not latency bounded.

To measure resource utilization, we will analyse several metrics collected by Hammer during a synthesis level analysis of hardware area and power consumption of our designs. To calculate the memory usage (size) of our model, we will use a simple approximation:

$$(\text{weight count} + \text{activation count}) * \text{bitwidth}$$

C. Control and Experimental Tests

As a control, we will gather baseline values from the testbench workload on Gemmini array configured for fp32. 32-bit floating point has been widely used in industry and is the standard numerical format used in machine learning and deep learning processes.

Other than the differing numerical format, variables will be kept constant as a control in future tests. For the second experiment, we will run the same testbench workload on a Gemmini array configured for bf16. As mentioned above, this will require modifications to the processing cores to allow computations with the novel bf16 format. Then we will run the testbench on a Gemmini array configured for int8. This experiment will require that we add quantization to the software test.

D. Evaluation and Hypothesis

Our hypothesis is that both the bf16 and quantized int8 methods will perform better than the baseline fp32 in all measured areas with the exception of model accuracy. As mentioned previously, reduced bitwidths mean that values take less space in memory and use less hardware for computation - leading to decreases in power and area consumption. The drawback of this is that the resolution of each stored value is reduced, which may affect model accuracy.

Directly comparing bf16 and quantized int8, we expect that each will have advantages under different constraints. Looking strictly at the bitwidths, we hypothesize that quantized int8 will be advantageous in latency or bandwith bounded environments due to having 8 fewer bits per value as well as integer only computation, both of which reduce the amount of hardware required. However, in situations with a high latency or bandwidth limit, we speculate that using bf16 will give the better accuracy results. In both cases, we expect that the quantized int8 Gemmini array will consume less power and area due to the lack of floating point paths and 8 fewer bits per values.

III. Comparisons of the State of Art

In this section, we will further examine cutting edge research into each method of running neural networks with reduced bitwidth values. For each method, we will take a deeper look at the contributions and findings of a prominent paper as well as any potential gaps we find in the research.

A. bFloat16

In "A Study of BFLOAT16 for Deep Learning Training" [15], Kalamkar et al. present an extensive empirical study on the effectiveness of using bf16 in various deep learning processes such as image classification, language modeling, and generative networks. They adopted a training methodology that utilized a mixed precision data flow that emulated bf16 operations by using existing fp32 operations. By zeroing out the last 16 bits of a fp32 value and rounding the resulting value via round to nearest even (RNE), performing fp32 operations on these emulated bf16 values would result in the same behavior as expected from custom bf16 hardware. This conversion happens between layers and before specific operations such as general matrix multiply, batch-normalization, and activation functions, to fit their "bf16-input, fp32-accumulator" training schema.

The paper evaluated the efficacy of the bf16 methodology across a comprehensive spectrum of differing frameworks such as CNNs, recurrent neural networks (RNNs), and generative adversarial networks (GANs). For CNNs such as AlexNet and ResNet, similar accuracy was achieved between bf16 emulation and a fp32 baseline. Similar results were found in RNNs tests using the DeepSpeech2 topology and Google's Machine Translation model. These results, along with those gathered from other deep learning frameworks, support the overall finding that the bf16 datatype is able to achieve similar accuracies as fp32, even in the most demanding applications.

While the emulation strategy used in this paper does not lead to any concrete results regarding bf16 hardware, it does acknowledge that it can act as a fair approximation. It is expected that future chips, such as Intel Xeon CPUs utilizing the AVX512BF16 instruction set extension, will demonstrate results similar to those found in this paper while supporting true bf16 operations. The study that we aim to conduct should also provide insights on bf16 specific hardware, as it requires the implementation of custom hardware that operates directly on the bf16 datatype.

In summary, this paper demonstrates that using bf16 within ML training processes results in accuracies that are near those of fp32, making bf16 an excellent half-precision alternative. Moreover, by testing bf16 across a wide range of compatible domains, the report demonstrates that bf16 is a versatile datatype that can be implemented in various existing ML frameworks. As the paper states, industry-wide adoption of bf16 across growing ML domains should be expected in the near future.

B. Quantization

In "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference" [8], Jacob et al. propose a quantization scheme that allows for inference to be carried out using 8 bit integer-only arithmetic while preserving accuracy from the pre-quantized model. The proposed quantization scheme allows both training and inference to be done using only integers. This reduces error from networks that train using floats and then quantize to integers post-training.

The error for these networks typically arises from the loss of precision when moving from a wide range of weights of a smaller range of quantized values. Furthermore, by using integer-only operations in both training and inference, this method allows for simple and commonly available integer-only hardware to be used for neural network workloads.

The paper evaluated the effectiveness of this quantization scheme by demonstrating the improvement it made to the latency-vs-accuracy trade-off on mobile cores. On the effectiveness side, 8-bit integer quantized ResNets were within 2% accuracy of the floating point equivalent network. Likewise, the quantized Inception v3 model was within 4.5% of the floating point equivalent. These results demonstrate that the proposed quantization scheme can achieve accuracy near that of floating models.

With regards to latency-vs-accuracy, quantized MobileNets were shown to enable a 25% face detection network to operate in real time when running on a mobile device, a feat that their float point counterparts could not achieve. When given limited latency budgets, the quantized MobileNets also achieved better accuracy than the non-quantized networks. When viewed from the perspective of static accuracy goals, quantization resulted in a 2x latency reduction without significant accuracy degradation.

While this paper clearly demonstrates the advantages of using int8 quantization for neural networks, it is important to note that the magnitude of the findings can be skewed by the relative speed between the integer and floating point hardware used in the study. Since different chips dedicate different amount of resources to optimizing floating point paths, it is possible that there is a chip where quantization has less of an impact due to significantly more effort being put into float point hardware when compared to integer hardware. Our own study will attempt to remove this potential complication by running all simulations on identical underlying hardware and only making changes to the content of MACs in the Gemmini array.

In brief, this paper shows that integer-quantization results in near-unnoticeable accuracy degradation for both large networks as well as mobile networks. It also establishes that quantization is a robust method of bitwidth reduction that can be integrated cleanly with current ML procedures. Finally, the significant latency reductions within MobileNets demonstrates that integer-quantization will be extremely advantageous in latency-bounded environments.

C. Applications to This Project

While there have been papers published on the effectiveness of reduced bitwidths in ML workloads, either through bf16 or integer quantization, there have been none so far that directly compare the benefits of these two methods. Our work will be able to fill this gap by providing insights into the trade-offs between bf16 and quantized int8. Our experiments will provide data for not only model accuracy, but also resource utilization statistics such as area, memory usage, and power consumption. This will enable an objective comparison of the effectiveness and efficiency of bitwidth reduction methods.

IV. Conclusion

As we have shown in this paper, quantization and novel numerical representation formats are two promising frontiers for reducing the latency and resource consumption of domain specific neural network architectures. To date, research has shown the effectiveness of these two methods with respect to 32-bit floating point, but little to no direct comparison has been done between the two methods of bitwidth reduction. In our work, we aim to fill this void through experimentation by running a ML workload across three different Gemmini generated systolic arrays. Through this, we expect to demonstrate that bf16 achieves better accuracy than quantized int8 when not bounded by latency, but that int8 has superior performance when latency goals are introduced or if resource utilization is a priority.

References

[1] S. Wang, and P. Kanwar. "BFloat16: The Secret to High Performance on Cloud TPUs" Google, Google, 23 Aug. 2019, cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

[2] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister, Y. S. Shao, B. Nikolić, I. Stoica, and K. Asanovíc. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. arXiv:1911.09925 [cs.DC], 2019.

[3] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, 2015.

[4] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," Tech. Rep. UCB/EECS-2016-17, EECS Department, University of California, Berkeley, Apr 2016.

[5] Grand View Research, Inc. Machine Learning Market Size Worth $96.7 Billion by 2025 — CAGR: 43.8%: Grand View Research, Inc. PR Newswire, 13 Jan. 2020. https://www.prnewswire.com/news-releases/machine-learning-market-size-worth-96-7-billion-by-2025–cagr-43-8-grand-view-research-inc-300985428.html

[6] J, Brutlag. Speed Matters. Google AI Blog, Google, 23 Jun. 2009, https://ai.googleblog.com/2009/06/speed-matters.html

[7] X. Amatriain, J. Basilico. Netflix Recommendations: Beyond the 5 stars (Part 1). Netflix Technology Blog, Netflix, 6 Apr. 2012, https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429

[8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. arXiv:1712.05877 [cs.DC] 2017

[9] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. arXiv:1810.05723 [cs.DC] 2019

[10] A. Krizhevsky, V. Nair, G. Hinton. CIFAR-10. https://www.cs.toronto.edu/ kriz/cifar.html

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, November 1998.

[12] Gemmini-rocc-tests. https://github.com/ucb-bar/gemmini-rocc-tests

[13] Chipyard. https://github.com/ucb-bar/chipyard

[14] Hammer. https://github.com/ucb-bar/hammer

[15] D. Kalamkar, D. Mudigere, et al. A Study of BFLOAT16 for Deep Learning Training. arXiv:1905.12322 [cs.DC] 2019