Accuracy Booster: Enabling 4-bit Fixed-point Arithmetic for DNN Training

Simla Burcu Harma
EcoCloud, EPFL
simla.harma@epfl.ch

Ayan Chakraborty
EcoCloud, EPFL
ayan.chakraborty@epfl.ch

Nicholas Sperry
EPFL
nicholas.sperry
grandhomme@epfl.ch

Babak Falsafi
EcoCloud, EPFL
babak.falsafi@epfl.ch

Martin Jaggi
EcoCloud, EPFL
martin.jaggi@epfl.ch

Yunho Oh
Korea University
yunho_oh@korea.ac.kr

Abstract

The unprecedented demand for computing resources to train DNN models has led to a search for minimal numerical encoding. Recent state-of-the-art (SOTA) proposals advocate for multi-level scaled narrow bitwidth numerical formats. In this paper, we show that single-level scaling is sufficient to maintain training accuracy while maximizing arithmetic density. We identify a previously proposed single-level scaled format for 8-bit training, Hybrid Block Floating Point (HBFP), as the optimal candidate to minimize. We perform a full-scale exploration of the HBFP design space using mathematical tools to study the interplay among various parameters and identify opportunities for even smaller encodings across layers and epochs. Based on our findings, we propose Accuracy Booster, a mixed-mantissa HBFP technique that uses 4-bit mantissas for over 99% of all arithmetic operations in training and 6-bit mantissas only in the last epoch and first/last layers. We show Accuracy Booster enables increasing arithmetic density over all other SOTA formats by at least $2.3 \times$ while achieving state-of-the-art accuracies in 4-bit training.

1 Introduction

Over the past decade, improvements in Deep Neural Network (DNN) algorithms have led to unprecedented growth in model complexity and dataset sizes. This growth has led to an exponential increase in the demand for computational resources to train DNN models which surpasses the conventional growth rate (according to Moore’s Law) in computational resources in the underlying hardware. Researchers and vendors have begun to search for alternate ways to improve the arithmetic density (operations per second per unit area of silicon) and operand storage in the underlying hardware platforms. Narrow bitwidth, low-precision numerical formats \cite{30,47,41,42,31} have emerged as a promising approach to overcome these limitations allowing for both high arithmetic density.

An ideal narrow bitwidth numerical format should maximize arithmetic density and minimize the storage requirements while maintaining FP32-level accuracy during training. Current state-of-the-art formats fail to optimize for all three constraints below 8 bits. Floating-point formats such as FP16 or BFloat16 \cite{47} maintain FP32-level accuracy but suffer from low arithmetic density. Likewise, fixed-point formats (such as INT8) do not work well for training because gradients have widely varying magnitudes.

Recent work has explored the use of scaled numerical formats to improve arithmetic density while maintaining high accuracy. Element-wise formats such as FP8 manage scaling factors in software at a coarse granularity ($\approx 1000$) and have not been show to reach FP32-level accuracy with 4-bit
fixed-point elements. In contrast, block-wise formats such as Block Floating Point (BFP) manage scaling factors in hardware to strike a balance between floating point and fixed point. BFP organizes floating-point numbers into blocks of mantissas, enabling fixed-point arithmetic by sharing a single exponent (as a scaling factor) within a block of fixed-point mantissas. Because the exponent management overhead in hardware gets amortized over the block size, BFP’s arithmetic density reaches that of fixed point even with fine-grain blocks (< 100 elements). Unfortunately, block-based formats by themselves have not been shown to maintain FP32-level accuracy for 4-bit training.

Similarly, in recent years, a consortium of vendors under the umbrella of the Open Compute Project has proposed block-wise scaled numerical formats based on shared micro-exponents (MX) allowing for two levels of fine-grained scaling on a block of mantissas. While micro-exponents optimize for storage density and have been shown to reach high accuracies with 6 bits per element, two-level exponent management in hardware fundamentally limits their arithmetic density as compared to 4-bit fixed point.

In this paper, we ask the question of whether scaled numerical formats can reach 4-bit fixed-point arithmetic density. Without loss of generality, we use Hybrid Block Floating Point (HBFP) as a mixed-precision recipe using BFP for dot products and higher precision (FP32/BFloat16) for the remaining operations to achieve FP32-level accuracy. We perform a full-scale empirical parameter exploration of HBFP with mathematical tools to derive insights about the impact of BFP’s parameter space on accuracy across layers and epochs. We show for the first time that with a fixed mantissa size across all epochs and layers, the minimum mantissa requirement for HBFP is 6 bits.

We then present Accuracy Booster, a novel mixed-mantissa training recipe that uses 4-bit mantissas in over 99% of training operations, and 6-bit mantissas in just the last epoch and first and last layers of DNN models. We use analytic models to estimate the silicon area of arithmetic units and show that Accuracy Booster increases the arithmetic density of a training accelerator by up to 25× over FP32, 5× over BFloat16, 4× over FP8, 2.3× over MXFP6, and 2.8× over MX9 while achieving state-of-the-art accuracies. Moreover, Accuracy Booster reduces the per-element storage requirement by 33% from 6 bits to 4 bits for all but the last epoch.

2 Arithmetic & storage density

In this section, we present a model to compare and contrast the arithmetic and storage density of a systolic-array-based training accelerator for state-of-the-art scaled numerical formats. Our metrics of interest are arithmetic density in operations per second per unit area of silicon, and operand storage in average number of bits required to store an element. We model a canonical systolic array rather than just multiply-accumulate units to estimate the area more precisely. The area estimate is more precise because, besides the basic logic to perform training operations, block-wise scaled numerical formats also require conversion units to and back from higher precision (e.g., FP32/BFloat16) to maintain FP32-level accuracy.

Figure 1 depicts the anatomy of a systolic array training accelerator of \( N \times N \) processing elements (PEs). Each PE computes the dot product of a group of \( W \) mantissas per cycle for each HBFP block, and includes an adder to compute the resulting exponent. The systolic array can compute the product
of two tiles of size \((N \times W) \times N\). We choose \(N = 8\) for all our analyses to ensure that the number of arithmetic units \((= N^2 \times W)\) does not exceed typical values found in modern accelerators. The array includes F2B conversion units along the edge and B2F conversion units integrated with each PE’s accumulator logic needed by block-based scaled numerical formats. Assuming all number formats take the same number of clock cycles for matrix multiplication, arithmetic density can be expressed as \((N^2 \times W) / \text{(Systolic Array Area)}\).

Much like prior work \[44\], we estimate silicon area as the total number of basic logic gates present in each systolic array. We assume the same clock rate for all configurations so that the number of operations per unit area becomes a proxy for arithmetic density. This approach \[2\] allows us to quickly iterate over various systolic array configurations without manually designing and synthesizing hardware for each numerical format. When evaluating element-wise formats, we estimate the number of logic gates for a dot product of tile \(W\) elements, equal to the block size in HBFP (or MX) or first scaling level for MXFP. The element-wise formats also include exponent management logic and an accumulator per element.

### Table 1: SOTA block-wise numerical formats, their arithmetic density and storage requirements.

|                | HBFP8 | HBFP6 | HBFP4 | MX9  | MX6\(^1\) | MX4\(^1\) | MXFP8 | MXFP6 | MXFP4 |
|----------------|-------|-------|-------|------|-----------|-----------|-------|-------|-------|
| **Scaling Level 1** |
| Block size     | 64    | 64    | 64    | 16   | 16        | 16        | 32    | 32    | 32    |
| Exponent bits  | 8     | 8     | 8     | 8    | 8         | 8         | 8     | 8     | 8     |
| **Scaling Level 2** |
| Block size     | N/A   | N/A   | N/A   | 2    | 2         | 2         | 1     | 1     | 1     |
| Exponent bits  | N/A   | N/A   | N/A   | 1    | 1         | 1         | 4     | 3     | 2     |
| Mantissa bits  | 7     | 5     | 3     | 2    | 3         | 1         |       |       |       |
| Normalized arithmetic density | 1.0   | 1.5   | 2.3   | 0.8  | 1.4       | 1.9       | 0.9   | 1.0   | 1.1   |
| Average number of bits/element | 8.1   | 6.1   | 4.1   | 9.0  | 6.0       | 4.0       | 8.2   | 6.2   | 4.2   |

\(^1\) not used for training.

In the rest of this paper, we refer to an HBFP configuration with \(m\) mantissa bits as HBFP\(_m\). We corroborate prior work \[47, 38\] indicating that 8-bit exponents are sufficient for maintaining high accuracy for block-wise numerical formats. Because the exponent’s silicon footprint is negligible relative to the block sizes we consider in this study, we use 8-bit exponents in the first scaling level in all block-wise formats without loss of generality.

Figure 2 compares the arithmetic density of HBFP with varying number of mantissa bits, and state-of-the-art element-wise formats, BFloat16 and FP8, normalized to that of FP32. The figure indicates HBFP’s superior arithmetic density over element-wise and coarse-grain scaled numerical formats thanks to its fixed-point mantissas. Moreover, the exponent management overhead quickly becomes negligible, with over 100 elements per block, allowing for the systolic array to reach fixed-point density. While the PE logic complexity is quadratic in the number of mantissa bits \[13\], we only observe a slightly superlinear improvement in asymptotic density between HBFP\(_8\) and HBFP\(_4\) because the conversion units do not scale commensurately.

The figure also indicates that with a block size of 64, an accelerator achieves \(\geq 90\%\) of asymptotic arithmetic density improvement. Sections 3.2 & 4 use mathematical analysis to show that a block size of 64 allows our numerical formats to achieve state-of-the-art accuracies. The figure illustrates that HBFP\(_4\) with a block size of 64 achieves 25\(\times\), 5\(\times\), 4\(\times\) higher density than FP32, BFloat16, and FP8, respectively. Not surprisingly, these floating-point formats are fundamentally limited by the inefficiencies of floating-point multipliers and accumulators. Unlike HBFP, MX and MXFP formats opt for smaller mantissas to allow for second-level scaling factors, and as such need smaller block sizes (e.g., 16-32) than HBFP to maintain high accuracy.

Table 2 compares the parameter space, the arithmetic density normalized to a baseline HBFP\(_8\), and per-element storage requirement for various state-of-the-art block-wise numerical formats. The results show that among the formats with \(\approx 4\) bits per element, HBFP achieves the best arithmetic density. MX formats suffer a slight degradation in density due to their two-level scaling. Nevertheless, MX formats are only used for inference because they can not reach sufficient accuracy in training \[38\]. In contrast, MXFP formats result in much lower arithmetic density due to two-level exponent management and a second-level scaling with a block size of one. Finally, even though MXFP\(_4\)
achieves a slightly better arithmetic density than MXFP6, it has only been used in practice for weights and still requires 6-bit arithmetic [39].

3 Minimizing HBFP

In this section, we focus on minimizing the HBFP encoding while achieving state-of-the-art accuracies. Unlike arithmetic density and storage requirements, which depend only on the HBFP parameter space, accuracy is also a function of the DNN model and the training algorithm. We elaborate on each of these factors impacting accuracy and use mathematical tools to derive insights about their impact. Finally, we explain our proposed technique for mixed-mantissa training to achieve 4-bit fixed-point arithmetic density.

3.1 What factors affect accuracy?

The primary factors impacting accuracy are the HBFP parameters including the exponent bitwidth, the mantissa bitwidth and the block size. We corroborate prior results (see Section 2) that there is no sensitivity to exponent bitwidth beyond 8 bits as we vary the block size and the mantissa bitwidth. As such, we keep the exponent bitwidth fixed at 8 and explore the space to minimize HBFP including techniques that exploit mixed-mantissa training.

The interplay between mantissa bitwidth and block size and its impact on accuracy has been reported empirically in many prior studies [7, 39, 13]. To the best of our knowledge, this paper is the first to evaluate meticulously this interplay and use rigorous mathematical tools to derive insights about how to minimize block-wise numerical formats for training state-of-the-art models.

The interplay between mantissa bitwidth and block size is driven by the quantization process of elements into a block of mantissa sharing an exponent. The process precludes normalization because the latter requires shifting the block of mantissas and adjusting the exponent which would negatively impact the arithmetic density (i.e., the elements are calculated as $2^{\text{exponent}} \times 0.\text{mantissa}$ rather than $2^{\text{exponent}} \times 1.\text{mantissa}$). As such, the precision of elements within a block is highly dependent on the largest element’s exponent.

The largest exponent in a block and the mantissa bitwidth dictate the magnitude of the interval defined by two representable consecutive numbers. This magnitude indicates that as the block size increases, the likelihood of disparity in the magnitude of elements also increases, leading to a precision loss for smaller elements in the block. Similarly, as the mantissa bitwidth decreases, the model’s sensitivity to the block size increases with a corresponding increase in the interval, leading to a higher quantization error. A larger mantissa bitwidth improves resilience to quantization errors due to larger block sizes, as each element can be represented more precisely.

Besides the HBFP parameters, other key factors impacting accuracy are the DNN model’s layers and the training epochs. The HBFP parameters can be tailored across layers and epochs with mixed-precision training. As a block-wise numerical format, HBFP naturally lends itself to mixed-mantissa training, keeping a fixed exponent for all blocks, and varying the mantissa bitwidth with little impact on the accelerator datapath (e.g., with bitwidths that are multiples of two). Prior work shows that keeping the first convolution layer and the last fully connected layer in high precision is crucial for CNN models [5, 29, 46]. Similarly, in Transformers, the first layer maps input tokens to dense word vectors in the embedding layer. The last layer in DNN models generates a probability distribution for the possible outcomes of the task. These layers require a boost in precision to maintain high accuracy, allowing for reduced precision in other layers.

Likewise, prior work [36, 51, 17, 16] shows that DNNs learn low-frequency, simple components in initial epochs and high-frequency, complex components in final epochs. In light of these findings, we hypothesize that high-frequency components are more sensitive to quantization errors, requiring larger mantissa bitwidths during the final training epochs. Thus, starting with small mantissa bitwidths for generalization and switching to larger ones in final epochs for optimization can boost accuracy.

3.2 Wasserstein distance & loss landscapes

To analyze the relationship between model accuracy, HBFP parameters, the DNN model and the training algorithm, we use two mathematical methods, Wasserstein distance and loss landscapes.
These techniques provide insights into HBFP’s optimization and generalization capabilities. The following subsections elaborate on our analysis and findings using these methods.

Preserving accuracy with narrow bitwidths depends on the numerical format’s representational power. If the format does not distort tensor distributions during training, accuracy is maintained. To study the distortion in distributions, we measure the similarity between BFP and FP32 tensors from various layers using Wasserstein distance, which is defined as:

$$W(P, Q) = \inf_{\gamma \in \Pi(P, Q)} \mathbb{E}_{(x,y) \sim \gamma}[||x - y||],$$

(1)

where $\Pi(P, Q)$ is the set of all joint distributions $\gamma(x, y)$ whose marginal distributions are equal to $P$ and $Q$. $\gamma(x, y)$ represents the amount of mass that must be transported from $x$ to $y$ to transform $P$ to $Q$. Unlike KL Divergence, which is commonly used to compare quantized tensors to their full-precision counterparts [7, 32], Wasserstein distance is symmetric, and thus a valid mathematical metric. Additionally, DNNs frequently deal with distributions where KL Divergence is not defined or infinite, necessitating the addition of a noise term that can cause disturbances. Our findings show that R-squared values between accuracy and various Wasserstein distances are approximately 0.99, validating the robustness of Wasserstein distance.

Figure 3 compares Wasserstein distance various HBFP configurations to that of FP32 for four layers of ResNet20 trained on CIFAR10. These layers include the first (conv1.weight), the last (fc.weight), and two representative convolutional layers in the middle of the model. We report power-of-two block sizes that are square numbers (i.e., 16, 64, 256) as blocks are formed as tiles. We also evaluate a block size of 576 as the baseline from [13]. We observe that HBFP4 has approximately a $3.5 \times$ higher distance than HBFP6, indicating more distorted tensor distributions. As a smaller Wasserstein distance indicates a closer match to FP32 accuracy, we hypothesize that HBFP6 is more likely to achieve accuracies comparable to FP32 than HBFP4.

The figure also shows that HBFP6 maintains consistent Wasserstein distances across various block sizes, while HBFP4 distances increase rapidly with block size. This observation indicates that HBFP6 accuracy will be invariant across various block sizes, whereas HBFP4 accuracy will degrade with larger block sizes. Based on these observations, we hypothesize HBFP6 is the smallest standalone HBFP format that will achieve competitive accuracies with FP32, with minimum sensitivity to block size.

Figure 3 also provides important insights into the behavior of the first and last layers of a CNN model. These layers show higher Wasserstein distances, indicating they are less resilient to quantization and should be kept at higher precision. Additionally, the increase in Wasserstein distance for these layers is significantly greater for HBFP4 compared to HBFP6, suggesting that these layers are more sensitive to HBFP4 and should be maintained at a precision of at least HBFP6.

To gain deeper insights into the interplay between generalization and optimization power of HBFP, we analyze log-scale loss landscapes for various configurations [27]. Visualizing loss landscapes highlights two key features: (1) the lower the curve’s minimum, the higher the accuracy of that configuration, and (2) the flatter the curve’s minimum (assuming it is sufficiently low) the better its ability to generalize. Ideally, we seek an HBFP configuration that matches FP32’s minimum and has a flatter curve.
The landscapes are generated across two axes, each representing a random direction. These two random directions are orthogonal due to the high-dimensional vector space, and they form a grid. For each training run, the loss value corresponding to the minimizer’s current state is positioned at the grid’s center \((0, 0)\). As the loss landscape shows similar behavior in both directions, we plot the landscape sliced along a single axis for clarity in Figure 4 (3D versions are available in Appendix E).

The figure indicates that the minimum of the HBFP6 curve is fairly close to that of FP32. In contrast, the HBFP4 curve’s minimum is significantly higher than that of FP32 even though its relatively flat shape may suggest better generalization. These findings are consistent with the Wasserstein distance analysis above.

Finally, to test the effect of keeping the first and last layers in higher precision, we use HBFP6 for these layers and HBFP4 for the remaining layers. This configuration, referred to as "HBFP4+Layers" in Figure 4, shows a sharper and lower curve compared to HBFP4. However, the model’s generalization and optimization powers are still imbalanced, resulting in convergence to another suboptimal local minimum.

To bridge the gap between generalization and optimization powers, we leverage the insight that the final epochs require higher precision. We introduce Accuracy Booster, a mixed-mantissa HBFP technique that uses HBFP6 only in the last epoch and first/last layers, and HBFP4 for the rest. We hypothesize that employing HBFP6 for the last epoch is sufficient to enhance the model accuracy, while the preceding epochs use HBFP4 to help the model generalize and reach a sufficiently low loss value. Switching to HBFP6 for the final epoch optimizes and fine-tunes the model, boosting accuracy close to FP32 levels.

The loss landscape for Accuracy Booster (referred to as "Booster" in Figure 4 for brevity) supports our hypothesis. We observe that the Booster curve approaches the HBFP6 and FP32 curves, and the flatness at the minima due to initial HBFP4 training indicates higher generalization power than FP32. This demonstrates that Accuracy Booster effectively uses minimal HBFP encoding and is the recipe to reach the sweet spot between generalization and optimization.

4 Experimental results

Our analysis in Section 3 suggest the following two hypotheses: (1) HBFP6 is the minimum single-mantissa encoding to achieve FP32-level accuracies in training, and (2) Accuracy Booster allows for 4-bit training for all but the last epoch while achieving FP32-level accuracies.

We test these hypotheses on state-of-the-art models and datasets for vision and language tasks. We train ResNet20/50/74 \([20]\), and DenseNet40 \([21]\) on CIFAR10 and CIFAR100 \([25]\), and ResNet50 on ImageNet \([9]\) for image classification. For Transformers, we train DeiT-Tiny \([43]\) on CIFAR-10, Transformer-Base \([45]\) on IWSLT’14 German-English \([3]\), BERT-Base \([12]\) on En-

| Number Format | Block Size | Normalized Arithmetic Density | CIFAR10 ResNet20 | CIFAR10 ResNet74 | CIFAR100 ResNet50 | CIFAR100 ResNet74 | CIFAR100 DenseNet40 |
|---------------|------------|-------------------------------|-----------------|-----------------|-------------------|-------------------|-------------------|
| FP32          | -          | 1                            | 91.7            | 93.6            | 74.1              | 74.6              | 72.4              |
| HBFP8         | 576        | 13                           | 91.5            | 93.4            | 73.8              | 74.3              | 73.7              |
| HBFP6         | 16         | 15                           | 91.1            | 93.4            | 73.1              | 73.5              | 72.1              |
|               | 64         | 18                           | 91.1            | 92.9            | 72.1              | 72.4              | 71.8              |
|               | 256        | 19                           | 91.4            | 92.8            | 72.0              | 72.5              | 71.5              |
|               | 576        | 20                           | 90.7            | 92.2            | 70.8              | 72.5              | 71.1              |
| HBFP4         | 16         | 20                           | 82.6            | 76.9            | -                 | -                 | 63.7              |
|               | 64         | 25                           | 80.2            | 74.4            | -                 | -                 | 62.4              |
|               | 256        | 27                           | 77.0            | 60.7            | -                 | -                 | 60.0              |
|               | 576        | 27                           | 75.3            | 66.7            | -                 | -                 | 59.8              |
| Total Number of FLOPs required to train the model | 41M | 174M | 119M | 326M | 542M |
We report all the hyperparameters we use for our experiments in Appendix B. Moreover, we tune the FP32 models’ hyperparameters and use the same hyperparameters for HBFP training, showing that our technique can be used out of the box without further tuning.

### 4.1 Baseline Single-Mantissa HBFP

Table 2 shows the Top1 validation accuracies for the CNN models with varying HBFP parameters and their arithmetic density normalized to FP32. The table corroborates our hypothesis that HBFP6 is the minimal single-mantissa HBFP configuration achieving accuracies within 2% of FP32 across all models and datasets for block sizes up to 256. Larger block sizes (e.g., 576 reported in prior work) reach diminishing returns in arithmetic density, and more importantly negatively impact accuracy for ResNet20 and ResNet50. In Section 4.2, we also present results for Transformer-based models with HBFP6 achieving FP32-level accuracy.

In contrast to HBFP6, HBFP4 offers higher arithmetic density but falls short of reaching FP32 accuracies for any block size. As the accuracy loss for ResNet50 and ResNet74 on CIFAR100 is considerably high, we did not train these models with HBFP4. Even for small models like ResNet20, the accuracy drops more than 9% with a block size of 16. We observe that for HBFP4, the sensitivity to block size increases for all the models because of higher distortion levels in tensor distributions.

### 4.2 Accuracy Booster

In this section, we evaluate Accuracy Booster on various CNN and Transformer-based models, presented in Table 3 and Table 4, respectively. We choose a block size of 64 as the sweet spot for Accuracy Booster, considering the HBFP hardware model. A block size of 64 is within 90% of the maximum arithmetic density gain while achieving accuracies with less than 1% degradation for standalone HBFP6. We use HBFP6 in the last epoch of training and HBFP4 for the rest of the epochs in all CNNs, DeiT-Tiny, and Transformer-Base. For the language modeling experiments, we use HBFP6 in the last 5k iterations for BERT-Base and the last 2k iterations for GPT-2. As an ablation study, we also used HBFP6 in the last 10 epochs for CNNs to test for further accuracy improvements.

We observe that for all CNN models, Accuracy Booster performs similarly to FP32, showing a huge jump in accuracies compared to standalone HBFP4 training. Similarly, for BERT-Base and Transformer-Base, Accuracy Booster outperforms FP32. For GPT-2 and DeiT-Tiny, Accuracy Booster incurs a minor accuracy loss, but it is still within the 1% error margin of FP32 accuracies. When we keep the last 10 epochs in HBFP6, we observe that the accuracies slightly increase, which may not be worth the extra computation depending on application needs.

Figure 5 shows the validation accuracy curves of ResNet74 and DenseNet40 on CIFAR100 with FP32, HBFP6, and Accuracy Booster (the curves for the other models are in Appendix D). For the Booster curves, we observe a noticeable sharp increase in accuracy towards the end for all models, demonstrating the significance of increasing precision in the final epoch. Overall, the training curves for Accuracy Boosters follow similar trends across all epochs compared to FP32 and HBFP6, albeit a bit noisier due to the reduced precision.

Our method achieves only 2% accuracy degradation with ResNet50 on ImageNet by using only 4-bit fixed-point arithmetic. Sun et al. propose 4-bit training by using INT4xINT4 arithmetic for

---

Table 3: Top-1 validation accuracies of various CNN models for Accuracy Booster

|               | CIFAR100 | ImageNet | CIFAR100 |
|---------------|----------|----------|----------|
| ResNet20      | 91.7     | 93.6     | 76.3     | 74.6     | 72.4     |
| ResNet74      | 91.4     | 93.0     | 74.2     | 73.7     | 73.6     |

---

1. We report HBFP5 in Appendix A to corroborate our conclusion about HBFP6 being the baseline single-mantissa encoding with FP32-level accuracies.
Table 4: Evaluation of various Transformer-based models for Accuracy Booster and HBFP6.

|          | BERT-Base (Perplexity) | GPT-2 (Loss) | DeiT-Tiny (Accuracy) | Transformer-Base (BLEU) |
|----------|------------------------|--------------|----------------------|-------------------------|
| FP32     | 3.3                    | 3.1          | 93.8                 | 34.8                    |
| HBFP6    | 3.3                    | 3.2          | 93.7                 | 34.5                    |
| Booster  | 3.2                    | 3.2          | 92.4                 | 36.1                    |
| Number of Model Parameters | 110M | 120M | 5M | 65M |

Figure 5: Top1 Accuracy training curves for ResNet74 and DenseNet40 on CIFAR100

forward pass and INT4xFP4 arithmetic for backward pass. However, their design requires support for both multiplication units, thus incurs a higher silicon area. Similarly, a recent paper by Chmiel et al. [4] applies the same regime but propose a more efficient way for INT4xFP4 multiplication and the resulting MAC unit is equivalent to FP7. These two papers report 2.3% and 1.1% degradation on ImageNet respectively with roughly 2.5× less arithmetic density compared to our method.

5 Related work

In recent years, there has been a significant amount of research on inference and training [47, 30, 37, 54, 41, 28, 6, 29, 42, 10] with narrow numerical encodings. The most recent work on FP8 [31] empirically shows accurate training and inference for various models including Transformers. However, all of these methods suffer from the intrinsically low arithmetic density of floating-point arithmetic. Additionally, these methods usually cannot go lower than 8 bits or require careful tuning. Recent research advocates the use of BFP for DNN training [14, 26, 8, 15] and inference [7] to overcome the limitations of floating-point formats. Drumond et al. [13] combine BFP with floating point to propose HBFP. Recently, Rouhani et al. [38] have proposed a new format based on shared micro-exponents (MX), which uses two levels of fine-grained scaling on blocks of fixed point numbers. A follow-up work [39] introduces MXFP6, where a block of FP6 elements share an 8-bit exponent. In this paper, we argue that further reducing the mantissa bitwidth in HBFP results in a significant improvement in arithmetic density for DNN training, and there is a wide design space left unexplored.

Mixed-precision training has emerged as a popular technique to recover the information loss caused by quantization. Several techniques [24, 52, 40, 5, 54, 29, 46] vary precision layer-wise, using higher precision for more important layers. Fu et al. [16] employ fixed-point arithmetic with varying bitwidths across epochs during training. Some techniques [17, 53, 55] combine layer-wise and epoch-wise approaches and vary the precision adaptively per epoch and layer at the same time using various control mechanisms. Recently proposed techniques for 4-bit training [42, 4] utilize different arithmetic units for the forward and backward passes. Although these aforementioned studies use leaner arithmetic for a fraction of the training process, they fail to make leaner arithmetic the common case of training, requiring additional arithmetic units.

Recent work [11] suggests that the optimizer states during training can be reduced to 8 bits by using block-wise quantization. This observation is in line with our work, applying quantization by extracting the largest exponent per block. Similarly, FAST [55] uses a BFP-based layer-wise
mixed-precision approach using 2 and 4-bit mantissas. Unlike our work, FAST requires fine-tuning several additional hyperparameters, making it difficult to apply to other DNN models. Another BFP-based work, FlexBlock [33], uses 4 and 8-bit mantissa with various block sizes but also needs higher-precision BFP formats for weight gradient calculations.

6 Discussion

Hardware Implications. Accuracy Booster primarily employs HBFP4 with a limited number of HBFP6 operations and can be implemented efficiently without requiring reconfigurable hardware. The 6-bit multiplications required for HBFP6 can be emulated on 4-bit HBFP processing elements in four steps (cycles). This process involves separating the four lower bits and the two higher bits that constitute the 6-bit mantissa. Let \( A \) and \( B \) be two mantissa vectors of two HBFP6 blocks. \( A_L \) denotes the four lower bits and \( A_H \) denotes the two higher bits of mantissas, such that \( A = 2^4 \cdot A_H + A_L \). Multiplication between two HBFP6 blocks of mantissas \( A \times B \) can be calculated in four cycles as follows:

\[
A \times B = 2^8 \cdot A_H \times B_H + 2^4 \cdot A_H \times B_L + 2^4 \cdot A_L \times B_H + A_L \times B_L
\]

Each of these four multiplications can be executed using the 4-bit arithmetic units and accumulated by shifting units with a minimal area cost, albeit at a latency increase of 4× per operation. However, as previously discussed, these 6-bit operations occur infrequently (< 1%), so the overall impact of throughput degradation due to these operations is negligible. Hence, we do not explicitly target reconfigurable systems; instead, we focus on fixed arithmetic units for 4-bit operations.

Limitations. One notable constraint is the dependency on specific hardware configurations that can support mixed-mantissa arithmetic efficiently, especially for operand storage. While the HBFP format enhances arithmetic density, it requires hardware support to handle the separation and combination of different bitwidth mantissas seamlessly. This could limit the applicability of Accuracy Booster to environments where such hardware is available or feasible to implement. Additionally, although we report our results on an extensive set of models, we do not experiment with large language models due to their high training overhead. Prior work [11] suggests that the outlier features in large models may hurt model accuracies of leaner formats. There are several proposals to handle outliers in large models, and we leave this study on how to leverage these proposals to handle outliers using HBFP for future work.

Impact. By enabling 4-bit fixed-point arithmetic, Accuracy Booster dramatically increases arithmetic density, reducing the computational resources and power consumption required for training DNN models. This enhancement can lead to more energy-efficient training processes, making it feasible to train complex models on resource-constrained devices such as edge computing hardware and mobile devices. Additionally, the insights gained from the mixed-mantissa approach may inspire further innovations in numerical formats and training algorithms, contributing to the ongoing advancement of efficient AI technology.

7 Conclusion

In this paper, we explore the extensive parameter space of BFP within the context of HBFP, delving into the interplay between block size and mantissa bitwidth. Leveraging mathematical tools such as Wasserstein distance and loss landscapes, we gain unique insights into the effects of training with block-wise formats on accuracy. Building on these insights, we introduce Accuracy Booster, a mixed-mantissa training recipe using HBFP6 only in the last epoch and first/last layers and HBFP4 for the rest. A significant contribution of our work lies in identifying opportunities for mixed-mantissa BFP encodings across layers and epochs during training, a previously unexplored dimension.

Our work not only expands the understanding of BFP but also paves the way for storage- and area-efficient DNN training. By providing a path to achieve 4-bit training with fixed-point arithmetic, we bridge the gap between storage optimization and arithmetic density. Our method uses 4 bits per element on average and increases the arithmetic density by up to 4× compared to FP8 and up to 2.3× compared to MXFP6 while achieving SOTA accuracies. This research contributes to the ongoing efforts in developing high-performance and resource-efficient DNN models, providing valuable insights that could shape future hardware-software co-design strategies for DNN training.
Acknowledgments and Disclosure of Funding

We thank Amir Yazdanbakhsh, Antoine Bosselut, Bita Darvish Rouhani, Shashwat Shrivastava, and the anonymous reviewers for their feedback and support. This work was partially supported by the SNSF project “Unified Accelerators for Post-Moore Machine Learning” (200021_212757) and a Microsoft Research PhD Fellowship.
References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. CoRR, abs/1701.07875, 2017.

[2] K. M. Büyüksahin and F. N. Najm. High-level area estimation. In V. De, M. J. Irwin, I. Verbauwhede, and C. Pigoet, editors, Proceedings of the 2002 International Symposium on Low Power Electronics and Design, 2002, Monterey, California, USA, August 12-14, 2002, pages 271–274. ACM, 2002.

[3] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico. Report on the 11th IWSLT evaluation campaign. In M. Federico, S. Stüker, and F. Yvon, editors, Proceedings of the 11th International Workshop on Spoken Language Translation: Evaluation Campaign@IWSLT 2014, Lake Tahoe, CA, USA, December 4-5, 2014, 2014.

[4] B. Chmiel, R. Banner, E. Hofer, H. Ben-Yaacov, and D. Soudry. Accurate neural training with 4-bit matrix multiplications at standard formats. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023.

[5] J. Choi, Z. Wang, S. Venkataramani, P. I. Chuang, V. Srinivasan, and K. Gopalakrishnan. PACT: parameterized clipping activation for quantized neural networks. CoRR, abs/1805.06085, 2018.

[6] M. Courbariaux, Y. Bengio, and J. David. Low precision arithmetic for deep learning. In Y. Bengio and Y. LeCun, editors, 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings, 2015.

[7] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, A. Forin, H. Zhu, T. Na, P. Patel, S. Che, L. Chand Koppaka, X. SONG, S. Som, K. Das, S. T. S. Reinhardt, S. Lanka, E. Chung, and D. Burger. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 10271–10281. Curran Associates, Inc., 2020.

[8] D. Das, N. Mellempudi, D. Mudigere, D. D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. O. Pirogov. Mixed precision training of convolutional neural networks using integer operations. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[10] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. CoRR, abs/2208.07339, 2022.

[11] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer. 8-bit optimizers via block-wise quantization. In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net, 2022.

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4171–4186. Association for Computational Linguistics, 2019.

[13] M. Drumond, T. Lin, M. Jaggi, and B. Falsafi. Training DNNs with Hybrid Block Floating Point. arXiv:1804.01526 [cs, stat], Dec. 2018. arXiv: 1804.01526.

[14] M. P. Drumond. Coltrain: Co-located dnn training and inference, 2020.

[15] S. Fox, S. Rasoulinezhad, J. Faraone, D. Boland, and P. H. W. Leong. A block minifloat representation for training deep neural networks. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[16] Y. Fu, H. Guo, M. Li, X. Yang, Y. Ding, V. Chandra, and Y. Lin. CPT: efficient deep neural network training via cyclic precision. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.
[17] Y. Fu, H. You, Y. Zhao, Y. Wang, C. Li, K. Gopalakrishnan, Z. Wang, and Y. Lin. Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient DNN training. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[18] A. Gokaslan and V. Cohen. Openwebtext corpus. [http://Skylion007.github.io/OpenWebTextCorpus](http://Skylion007.github.io/OpenWebTextCorpus), 2019.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[21] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[22] karpathy. nanogpt. [https://github.com/karpathy/nanoGPT](https://github.com/karpathy/nanoGPT).

[23] S. Khoram and J. Li. Adaptive quantization of neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[24] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[25] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. J. Pai, and N. Rao. Flexpoint: An Adaptive Numerical Format for Efficient Training of Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[26] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6391–6401, 2018.

[27] D. D. Lin, S. S. Talathi, and V. S. Annapureddy. Fixed point quantization of deep convolutional networks. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2849–2858. JMLR.org, 2016.

[28] N. Mellempudi, S. Srinivasan, D. Das, and B. Kaul. Mixed Precision Training With 8-bit Floating Point. [arXiv:1905.12334][cs, stat], May 2019. arXiv: 1905.12334.

[29] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed Precision Training. [arXiv:1710.03740][cs, stat], Feb. 2018. arXiv: 1710.03740.

[30] S. Migacz. 8-bit Inference with TensorRT, May 2017.

[31] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.
[36] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. C. Courville. On the spectral bias of neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, 9-15 June 2019, Long Beach, California, USA, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 2019.

[37] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 525–542. Springer, 2016.

[38] B. D. Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar, L. Shao, G. Kolhe, D. Melts, J. Klar, R. L’Heureux, M. Perry, D. Burger, E. S. Chung, Z. S. Deng, S. Naghshineh, J. Park, and M. Naumov. With shared microexponents, a little shifting goes a long way. In Y. Solihin and M. A. Heinrich, editors, *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA 2023*, Orlando, FL, USA, June 17-21, 2023, pages 83:1–83:13. ACM, 2023.

[39] B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, S. Dusan, V. Elango, M. Golub, A. Heinecke, P. James-Roxby, D. Jani, G. Kolhe, M. Langhammer, A. Li, L. Melnick, M. Mesmakhosroshahi, A. Rodriguez, M. Schulte, R. Shafipour, L. Shao, M. Siu, P. Dubey, P. Micickevicius, M. Naumov, C. Verrilli, R. Wittig, D. Burger, and E. Chung. Microscaling data formats for deep learning, 2023.

[40] J. Shen, Y. Wang, P. Xu, Y. Fu, Z. Wang, and Y. Lin. Fractional skipping: Towards finer-grained dynamic CNN inference. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5700–5708. AAAI Press, 2020.

[41] X. Sun, J. Choi, C.-Y. Chen, N. Wang, S. Venkataramani, V. V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan. Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[42] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan. Ultra-Low Precision 4-bit Training of Deep Neural Networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1796–1807. Curran Associates, Inc., 2020.

[43] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers and distillation through attention, 2021.

[44] M. van Baalen, A. Kuzmin, S. S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soria, and T. Blankevoort. FP8 versus INT8 for efficient deep learning inference. *CoRR*, abs/2303.17951, 2023.

[45] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[46] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18*, page 7686–7695, Red Hook, NY, USA, 2018. Curran Associates Inc.

[47] S. Wang and P. Kanwar. BFloat16: The secret to high performance on Cloud TPUs, Aug. 2019.

[48] R. Wightman. Pytorch image models. [https://github.com/rwightman/pytorch-image-models](https://github.com/rwightman/pytorch-image-models), 2019.

[49] Wikimedia. Wikimedia downloads.
[50] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[51] Z. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. CoRR, abs/1901.06523, 2019.

[52] L. Yang and Q. Jin. Fracbits: Mixed precision quantization via fractional bit-widths. In Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021, pages 10612–10620. AAAI Press, 2021.

[53] S. Q. Zhang, B. McDanel, and H. Kung. Fast: Dnn training under variable precision block floating point with stochastic rounding. In 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pages 846–860. IEEE, 2022.

[54] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. [arXiv:1606.06160 [cs]], Feb. 2018. arXiv: 1606.06160.

[55] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In The IEEE International Conference on Computer Vision (ICCV), December 2015.
A  HBFP5 accuracies

We report the top-1 validation accuracies of various CNN models for HBFP5 (along with the other configurations for ease of comparison) to corroborate our conclusion about HBFP6 being the baseline single-mantissa encoding with FP32-level accuracies.

Table 5: Top-1 validation accuracies of various CNN models for various HBFP configurations.

| Number Format | Block Size | Normalized Arithmetic Density | CIFAR10 ResNet20 | CIFAR100 ResNet74 | CIFAR100 ResNet50 | CIFAR100 ResNet74 | CIFAR100 DenseNet40 |
|---------------|------------|--------------------------------|------------------|------------------|------------------|------------------|--------------------|
| FP32          | -          | 1                              | 91.7             | 93.6             | 74.1             | 74.6             | 72.4               |
| HBFP8         | 576        | 13                             | 91.5             | 93.4             | 73.8             | 74.3             | 73.7               |
| HBFP6         | 16         | 15                             | 91.1             | 93.4             | 73.1             | 73.5             | 72.1               |
|               | 64         | 18                             | 91.1             | 92.9             | 72.1             | 72.4             | 71.8               |
|               | 256        | 19                             | 91.4             | 92.8             | 72.0             | 72.5             | 71.5               |
|               | 576        | 20                             | 90.7             | 92.2             | 70.8             | 72.5             | 71.1               |
| HBFP5         | 16         | 17                             | 90.2             | 92.2             | 67.2             | 70.5             | 69.4               |
|               | 64         | 21                             | 89.7             | 91.3             | 66.8             | 69.3             | 68.9               |
|               | 256        | 23                             | 89.1             | 91.1             | 63.0             | 66.6             | 67.8               |
|               | 576        | 23                             | 88.9             | 91.0             | 64.6             | 64.4             | 67.6               |
| HBFP4         | 16         | 20                             | 82.6             | 76.9             | -                | -                | 63.7               |
|               | 64         | 25                             | 80.2             | 74.4             | -                | -                | 62.4               |
|               | 256        | 27                             | 77.0             | 60.7             | -                | -                | 60.0               |
|               | 576        | 27                             | 75.3             | 66.7             | -                | -                | 59.8               |

Total Number of FLOPs required to train the model: 41M, 174M, 119M, 326M, 542M

B  Hyperparameters for training

The CNN models for image classification are trained for 160 epochs each on CIFAR10, and 300 epochs each on CIFAR100. DeiT-Tiny is trained for 300 epochs, and Transformer-Base for 70 epochs. We used the HuggingFace Transformers library [50] for BERT-Base training and nanoGPT [23] for GPT2 training, and we trained these models for 500k and 12k iterations, respectively. We use NVIDIA V100 32GB and A100 80GB GPUs in our evaluations. For large models and datasets, we train on multiple GPUs by using data parallelism. We report the total amount of computations required to train all CNN models in Table 2 and the number of parameters of Transformer-based models in Table 4.

We use FP32 precision as the baseline for model accuracies. For the image classification experiments, we report the Top1 validation accuracies; for machine translation, we report the BLEU scores, for language modeling we report the perplexities. Moreover, to show the impact of our method, we tune the hyperparameters of FP32 models and then train the same models from scratch with the same hyperparameters in HBFP, showing that our method can be used out of the box without further hyperparameter tuning.

B.1 Image classification experiments

We report the hyperparameters we use for all our CNN experiments in Table 6. All convolutional layer weights are initialized using a zero-mean Gaussian distribution whose standard deviation (std) is \( \sqrt{2/n_{l}} \) \( (n_{l} \) is the number of activations in the output), a method introduced in [19]. All the Batch Normalization layer weights are initialized to 1 and are kept in FP32 as HBFP suggests [13].

B.2 Neural machine translation experiments

Our implementation is based on the encoder-decoder (seq2seq) Transformer implementation of Fairseq [35]. We report the hyperparameters we use for all our experiments in Table 7. For the ones we do not specify, we use the default hyperparameters.
Table 6: Hyperparameters for various sizes of ResNet on CIFAR10 and CIFAR100

| Hyperparameter          | ResNet20 & ResNet74 on CIFAR10 | DenseNet40 & ResNet74 on CIFAR100 |
|-------------------------|---------------------------------|-----------------------------------|
| Epochs                  | 160                             | 300                               |
| Batch size              | 128                             | 128                               |
| Optimizer               | SGD                             | SGD                               |
| Nesterov                | True                            | True                              |
| Learning rate           | 0.1                             | 0.1                               |
| Learning rate decay epochs | 82, 122                         | 150, 225                          |
| Learning rate warmup    | True                            | True                              |
| Weight decay            | 1e-04                           | 1e-04                             |
| Momentum                | 0.9                             | 0.9                               |

Table 7: Hyperparameters for Transformer on IWSLT’14 German to English translation task

| Transformer architecture hyperparameters                  |
|-----------------------------------------------------------|
| Dataset                                                   | iwslt14.tokenized.de-en                |
| Architecture                                              | transformer_iwslt_de_en               |
| Number of layers                                         | 6                                  |
| Number of heads                                          | 4                                  |
| Number of hidden dimensions                               | 512                                 |
| Embedding dimension for FFNs                              | 1024                                 |
| Optimizer                                                | Adam                                |
| Adam-betas                                               | (0.9, 0.98)                         |
| Learning rate                                            | 0.0005                              |
| Learning rate scheduler                                   | Inverse Square Root                  |
| Warmup updates                                           | 4000                                 |
| Warmup initial learning rate                              | 1e-07                               |
| Label smoothing                                          | 0.1                                 |
| Maximum tokens                                           | 4096                                 |
| Criterion                                                | label_smoothed_cross_entropy         |
| Clip-norm                                                | 0.0                                 |
| Weight decay                                             | 0.0001                              |
| Dropout                                                  | 0.3                                 |

B.3 Experiments with other transformer models

As mentioned in Section 4, we use the HuggingFace Transformers library [50] for BERT-Base training and nanoGPT [23] for GPT2 training, and Huggingface Timm [48] for DeiT-Tiny. We use the original hyperparameters mentioned in the respective papers for all these models.

C Error bars

![Error bars for ResNet20 on CIFAR10 with 5 different seeds for various configurations.](image_url)
In this section, we report the error bars for ResNet20 on CIFAR10 ran on 5 different seeds for FP32, HBFP6, and Accuracy Booster (see Figure 6). Columns show the average over the results for the multiple seeds, and the lines indicate the variability of the results (standard deviation). We observe that the variation in results is not more than 0.4%. We do not report larger models because multiple runs with such models are computationally expensive.

D  CNN training curves

In this section, we report the training curves of all CNN models trained on CIFAR10 (see Figure 7).

![Figure 7: Top1 Accuracy training curves for ResNet20 and ResNet74 on CIFAR10](image)

E  Loss landscapes

In this section, we report the 3D versions of the loss landscapes explained in Section 3.2. Figure 8 shows loss landscapes for ResNet20 trained on CIFAR10 with various configurations.
Figure 8: Loss landscapes in 3D for ResNet20 on CIFAR10 with various configurations.