FF-INT8: Efficient Forward-Forward DNN Training on Edge Devices with INT8 Precision

Jingxiao Ma
School of Engineering
Brown University
Providence, RI
jingxiao_ma@alumni.brown.edu

Priyadarshini Panda
Department of Electrical Engineering
Yale University
New Haven, CT
priya.panda@yale.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI
sherief_reda@brown.edu

***Abstract*—Backpropagation has been the cornerstone of neural network training for decades, yet its inefficiencies in time and energy consumption limit its suitability for resource-constrained edge devices. While low-precision neural network quantization has been extensively researched to speed up model inference, its application in training has been less explored. Recently, the Forward-Forward (FF) algorithm has emerged as a promising alternative to backpropagation, replacing the backward pass with an additional forward pass. By avoiding the need to store intermediate activations for backpropagation, FF can reduce memory footprint, making it well-suited for embedded devices. This paper presents an INT8 quantized training approach that leverages FF's layer-by-layer strategy to stabilize gradient quantization. Furthermore, we propose a novel "look-ahead" scheme to address limitations of FF and improve model accuracy. Experiments conducted on NVIDIA Jetson Orin Nano board demonstrate 4.6% faster training, 8.3% energy savings, and 27.0% reduction in memory usage, while maintaining competitive accuracy compared to the state-of-the-art.**

***Index Terms*—Neural network quantization, Low power, Low-precision training, Resource-constrained devices.**

I. Introduction

Deep neural networks (DNNs) have achieved state-of-the-art performance across many application domains. However, their growing complexity significantly raises the computational cost of training, doubling roughly every six months from 2010 to 2022 [1]. For example, ResNet-50, which is introduced in 2015, consists of 26 million parameters and takes about 14 days to train on an NVIDIA M40 GPU [2]. In contrast, GPT-3, a large language model released in 2020, which contains 175 billion parameters, requires 3,640 PF-days to train, equivalent to 355 years of single-processor computing time, consuming 284,000 kWh of energy and costing more than 4.6 million US dollars [3]. These trends pose even greater challenges for edge devices, where limited memory and power resources make efficient, real-time DNN training an urgent priority.

The last few years have seen various methodologies for efficient deep learning computing, which can be categorized into pruning, quantization, neural architecture search, *etc* [4]. Among these, quantization stands out for its clear impact on memory footprint and computational cost, achieved by reducing the precision of weights and activations without altering the number of learned features [5]. Moreover, many edge devices are equipped with specialized hardware optimized for low-precision computations, making quantization particularly well-suited for edge computing scenarios. Despite its benefits, most quantization techniques are limited to model inference [6]. Meanwhile, the Forward-Forward (FF) algorithm [7] offers an alternative to backpropagation, replacing the backward pass with an additional forward pass to address inefficiencies in gradient computation. This paper delves into yet another advantage of the FF algorithm, specifically its applicability in implementing INT8 quantized training algorithms. The contributions of this paper are outlined below.

- To the best of our knowledge, FF-INT8 is the first work to devise a low-precision training method leveraging the Forward-Forward algorithm. Our observation reveals that FF algorithm's layer-wise greedy approach effectively mitigates the accumulation of accuracy degradation, a common issue in quantized backpropagation techniques.
- We propose a novel training method based on the Forward-Forward algorithm, named FF-INT8. This approach adopts a greedy, layer-wise training strategy, where each layer of a deep neural network is trained independently using INT8 precision, making it especially well-suited for resource-constrained edge devices.
- We identify two key drawbacks of the Forward-Forward algorithm: low convergence accuracy and slow convergence speed. To address these limitations, we propose a novel "look-ahead" scheme that incorporates information from subsequent layers. This enhancement improves both training accuracy and convergence efficiency, particularly in low-precision training environments.
- Using FF-INT8, we train multiple DNNs on the Jetson Orin Nano board, a typical edge device. Our experimental results show that FF-INT8 significantly improves efficiency while preserving high accuracy. We achieve a speedup of 4.6% in DNN training, reduce the memory footprint by 27.0% and energy consumption by 8.3% compared to a state-of-the-art INT8 training algorithm.

The organization of this paper is as follows. In section II, we overview relevant previous works in DNN quantization. We then briefly introduce Forward-Forward algorithm as background in section III. In section IV, we introduce FF-INT8 training algorithm. We provide our experimental results in section V. Finally, we summarize our conclusion in Section VI.

This work was supported in part by the National Science Foundation (Grant #2350180, #2453413, #2312366, and #2318152).

II. Previous Work

A wide range of quantization methods have been proposed to enhance the efficiency of DNN inference, with post-training quantization (PTQ) being the most common approach. PTQD [8] proposes a PTQ framework for diffusion models that optimizes model efficiency and performance by disentangling and correcting quantization noise. To make the model more robust to the reduced precision, methodologies of quantization-aware training (QAT) are proposed to preserve end-to-end model accuracy post quantization [9], [10].

Compared to the large amount of studies on accelerating inference by model quantization, few works explore low-precision training. MPTraining [11] trains DNNs using 16-bit floating-point. Minifloat [12] also proposes to use 8-bit floating-point numbers to train DNNs. Floating-point quantization requires specific hardware platform to achieve acceleration, which is not conducive to the practical deployment on resource-constrained devices. On the other hand, compared to floating-point, quantizing into 8-bit integer has great potential in hardware acceleration, since INT8 operations are widely supported by recent GPUs. UI8 [13] utilizes direction sensitive gradient clipping and deviation counteractive learning rate scaling to implement a fully unified INT8 training for convolutional neural networks. DAI8 [14] achieves better gradient quantization by performing channel-by-channel gradient distribution perception, but it sets too many channel dimension quantization parameters, adding additional computational complexity. GDAI8 [15] also uses a data-aware dynamic quantization scheme to quantize various special gradient distributions. In summary, most INT8 training methods require extra effort to adapt to gradient distribution.

III. Background

The traditional backpropagation (BP) algorithm is inefficient in several ways. First, it requires substantial memory to store the full computational graph, leading to a large memory footprint. Additionally, the backward pass of derivative computation takes much longer time and consumes more energy. In contrast, the Forward-Forward (FF) training algorithm [7] offers a novel direction to DNN training by replacing the forward and backward passes of BP with two forward passes to improve efficiency. Unlike BP, the FF algorithm uses two distinct data sets with opposing objective functions and trains the DNN layer-by-layer in a greedy manner. Figure 1 compares training procedure between BP and FF. In FF (Figure 1b), the first step is to generate "positive" and "negative" datasets from input data. One way to generate datasets is to annotate the input data using one-hot encoding vector of the label [7]. More specifically, the one-hot vectors in positive samples are pointing to the true labels, while the ones in negative samples are pointing to the wrong labels. FF algorithm uses *goodness* function G to qualify how well each layer is trained. The positive pass operates on positive samples and adjusts the weights to increase the goodness in every hidden layer, while the negative pass operates on negative samples and adjusts the weights to decrease the goodness. One common measurement

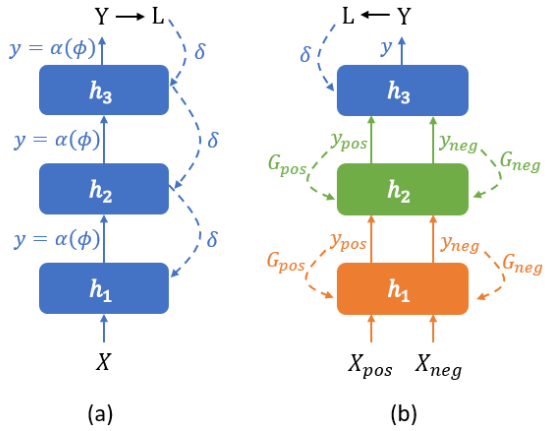

Fig. 1. (a) Backpropagation consists of a forward pass and a backward pass. (b) The Forward-Forward algorithm uses "positive" and "negative" datasets, and trains each layer individually using "goodness" function G.

of goodness function is the sum of squared neural activities. FF algorithm aims to make the goodness function well above a certain threshold θ for positive data (G_{pos}) and well below θ for negative data (G_{neg}), where θ can be considered as a hyperparameter that controls scale of the weights.

FF algorithm trains each layer once until convergence and then optimize its successor. The main idea behind such greedy scheme is to make each layer "excited" about positive samples and, at the same time, less excited about negative ones, so that positive samples can be trained to match the correct label later. FF algorithm improves efficiency in many aspects. First, instead of storing the entire computational graph, FF algorithm only stores current layer in memory, which significantly reduces memory footprint. Also, forward pass takes less time than backward pass. However, the heuristic property prevents earlier layers to learn from the later ones and final outputs. where training of earlier layers can be misleading. As a result, the FF algorithm often suffers from accuracy loss and struggles to scale effectively to modern, large-scale DNNs.

IV. Proposed Methodology

In this section, we describe our proposed methodology of INT8 Forward-Forward algorithm for efficient DNN training. Our FF-INT8 training method builds upon symmetric uniform quantization (SUQ), which is one of the most efficient quantization methods due to its hardware-friendly computation [6].

A. Network Depth and Gradient Quantization

As a preliminary experiment, we train ResNet-18 using CIFAR-10 dataset, with backpropagation and INT8 quantization including gradients. Figure 2 compares the changes of loss and accuracy per epoch between 32-bit floating-point (FP32) and INT8 training. The network is trained properly with FP32 precision. However, the loss of INT8 training increases dramatically as soon as we start training, while the accuracy drops to random level. We assume that INT8 training fails due to the accumulation of quantization error through backward propagation of derivatives. To further prove this hypothesis, we train three multilayer perceptrons (MLP) with different number of layers on MNIST dataset with different

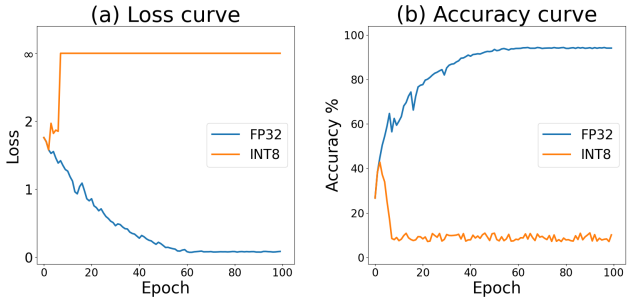

Fig. 2. Loss and accuracy of ResNet-18 on CIFAR-10 when gradients are directly quantized to INT8.

TABLE I
ACCURACY OF MULTILAYER PERCEPTRONS ON MNIST DATASET WITH DIFFERENT NUMBER OF HIDDEN LAYERS AND TRAINING PRECISION. EACH HIDDEN LAYER CONSISTS OF 500 NEURONS.

Number of Hidden Layers	FP32 Acc. (%)	INT8 Acc. (%)	Accuracy Difference (%)
0	89.5	88.7	-0.8
1	93.4	73.8	-19.6
2	94.5	62.4	-32.1
3	94.3	65.2	-29.1

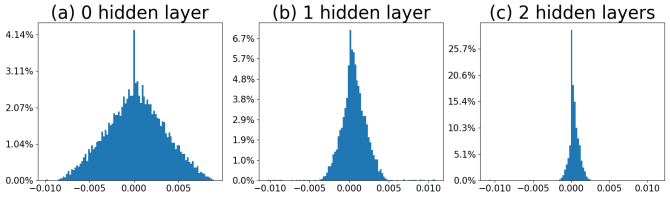

Fig. 3. Gradient distribution of first layer with different num. of hidden layers.

number of hidden layers, each of which is trained with FP32 and INT8 precision respectively. As Table I shows, as the number of hidden layers increases, the accuracy of FP32 training increases until the model overfits. On the contrary, the accuracy of INT8 training decreases dramatically as the network becomes deeper. The accuracy difference between FP32 and INT8 training is considerably small with a single-layer network, but increases significantly as we include the first hidden layer. We may conclude that quantization error accumulates as network becomes deeper. Instead of deeper networks, INT8 training can be directly executed on a single-layer network. Finally, we plot gradient distribution of the *first* layers for each MLP in Table I, which are trained using FP32. As Figure 3 shows, for deeper networks, gradient distribution of earlier layers are sharper with larger extreme values, while distribution of single-layer network is more even. Thus, direct quantization in deeper network leads to large quantization error, since most gradients gather in a small range and we cannot tell the differences in such small range after quantization. On the other hand, direct quantization in single-layer network is less error-prone. FF algorithm proposes to train each layer individually in a greedy manner, making it particularly compatible with INT8 training.

B. INT8 Forward-Forward Algorithm

FF-INT8 algorithm trains each layer individually in a greedy manner using both positive and negative datasets, with only one layer being trained at a time. For each dataset (positive or

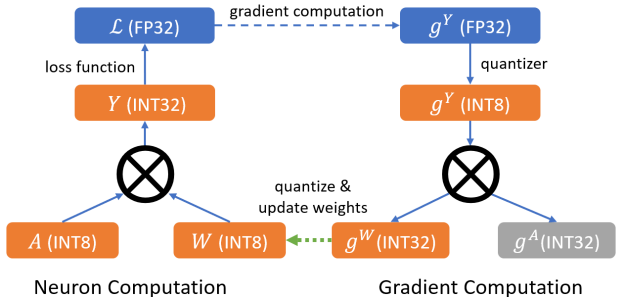

Fig. 4. Dataflow of FF-INT8 on a single layer.

negative), our FF-INT8 training workflow for a single layer is illustrated in Figure 4. Initially, we apply SUQ with stochastic rounding [16] to quantize the input data A. INT8 MAC operations are then employed to compute the dot product in either dense or convolution layers. Specifically, integer matrix multiplication with INT8 inputs and INT32 accumulation is used. The loss function is based on the negative log-likelihood of the goodness function G. For the positive dataset, the loss function $\mathcal{L}_{pos}$ is computed as follows:

$$\mathcal{L}_{pos} = -\log p(positive) = \log\left(1 + e^{-(G_{pos}-\theta)}\right) \quad (1)$$

For negative dataset, loss function $\mathcal{L}_{neg}$ is computed as

$$\mathcal{L}_{neg} = -\log p(negative) = \log\left(1 + e^{(G_{neg}-\theta)}\right) \quad (2)$$

where G represents the goodness function, and θ is the hyper-parameter that represents threshold as discussed in Section III.

By optimizing this loss function, we encourage outputs of positive samples to be large and outputs of negative samples to be small. The gradient g^Y is computed and quantized into INT8. Since gradients are not back-propagated to previous layers, there is no need to compute the gradients of input data g^A. Only gradients of weights g^W are computed and updated to current weights. Thus, for computation of neuron activity Y and gradient g_W, we replace floating-point computations with INT8, which saves a large amount of computation.

C. FF-INT8 Algorithm with "Look-Ahead"

As discussed in Section III, FF algorithm offers efficiency but imposes limitations by restricting each layer to learn solely from its immediate input, preventing communication with subsequent layers, particularly the final outputs. Consequently, earlier layers are primarily trained to distinguish between positive and negative data, leaving the task of specific label classification to the final layer. This lack of feedback from final outputs can lead to suboptimal training of earlier layers. As a result, FF algorithm often suffers from lower accuracy and requires more epochs to converge compared to backpropagation. These challenges are exacerbated in deeper networks, rendering the FF algorithm less effective for training large-scale DNNs. Additionally, FF algorithm struggles to accommodate widely-used structures such as residual blocks, which allow inputs to bypass certain layers and contribute directly to the block's output [17]. To optimize residual blocks effectively, all layers within the block must be trained interactively. However, due to the layer-by-layer nature of FF algorithm, layers within the same block are trained independently, without awareness of subsequent layers, leading to significant accuracy loss.

Algorithm 1: FF-INT8 with "Look-Ahead"

Input : Training set $(\mathbf{X}, \mathbf{Y})$, Number of epochs n
Output: Trained network M with k layers

1 Initialize neural network M
2 Initialize hyperparameter $\lambda = 0$ in loss function
3 Generate positive and negative sets from $(\mathbf{X}, \mathbf{Y})$
4 **for** $num\_epoch = 1, 2, \ldots, n$ **do**
5 Execute forward pass using INT8 precision
6 Compute goodness function $G = ||\mathbf{y}||^2$ for each layer
7 **for** *current layer* $= 1, 2, \ldots, k$ **do**
8 Compute loss for current layer using G
9 Update weights using gradients in INT8 precision
10 **end**
11 Increase hyperparameters λ in loss function
12 **end**

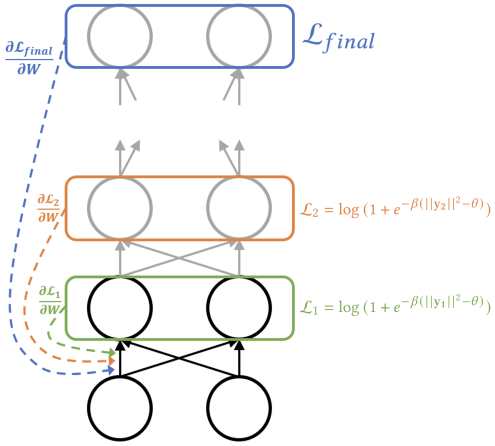

Fig. 5. Gradient computation of FF algorithm with "look-ahead", where loss functions of later layers are considered.

To solve these issues, our solution is to bridge connections between different layers, especially with final outputs, while retaining the efficiency of the FF algorithm. We name our method "*Look-Ahead*" scheme. In the standard FF algorithm, the loss function considers only the goodness function of the current layer during training. To enable feedback from subsequent layers, we redefine the loss function to incorporate the goodness functions of these layers. The revised loss function is defined as:

$$\mathcal{L}_{new} = \mathcal{L}_1 + \lambda \times (\mathcal{L}_2 + \mathcal{L}_3 + \cdots + \mathcal{L}_{final}) \tag{3}$$

Here, $\mathcal{L}_i$ represents the loss of the i^{th} layer, derived from the goodness function, and λ is a coefficient that balances the contributions of the current layer and the subsequent layers. Figure 5 illustrates the derivative computation with this modified loss function, as follow:

$$\frac{\partial \mathcal{L}_{new}}{\partial W} = \frac{\partial \mathcal{L}_1}{\partial W} + \lambda \times \frac{\partial (\mathcal{L}_2 + \mathcal{L}_3 + \cdots + \mathcal{L}_{final})}{\partial W} \tag{4}$$

In Section IV-B, we described a methodology where each layer is trained for n epochs independently, without considering information from later layers. For a neural network with k layers, this approach requires a total of $k \times n$ derivative computations. With the new loss function introduced in Equation 3, which involves derivatives from subsequent layers, we must perform complete forward passes for each derivative calculation. For the sake of efficiency, we propose Algorithm 1 to utilize *one* forward pass to update all layers. Specifically, for each epoch, we execute complete forward pass (line 5), and compute goodness function G for each layer (line 6). We can then construct loss functions of all layers as Equation 3 using goodness function of each layer (line 8), and compute the gradient (line 9). This approach facilitates interaction between layers, enabling them to optimize simultaneously and improve accuracy. Importantly, the goodness function for each layer depends solely on its neuron values, and the loss function is based solely on the goodness function. As a result, constructing the backward derivative chain is unnecessary, maintaining the computational efficiency of the loss and derivative calculations as vanilla FF-INT8 algorithm. Overall, the total number of derivative computations remains $k \times n$, the same as before. As later demonstrated in Section V-B, the "look-ahead" scheme significantly reduces the number of epochs needed for convergence, further improving efficiency. Regarding memory usage, since this scheme requires a full forward pass to compute the goodness function of later layers, all network weights must be retained in memory, resulting in a modest increase in memory footprint. However, compared to backpropagation, this approach still saves memory by avoiding the storage of intermediate activations and the full gradient chain, as it does not rely on a complete backward pass.

One more hyperparameter in "FF-INT8 Algorithm with look-ahead" is λ, which balances between current layer and other layers. For first few epochs, since each layer is less optimized, our priority is to train each layer so that it can basically tell between positive samples and negative samples. In other word, we want to discourage the interaction between layers at the beginning of training, since later layers are not optimized and information of later layers does not help the optimization of current layer. After a few epochs, each layer is better optimized. We then gradually increase λ to encourage interaction between layers to improve convergence accuracy.

V. Experimental Results

In this section, we first introduce our experimental setup. The first experiment is to compare FF-INT8 algorithm with and without "look-ahead", to show the benefit of it. We then theoretically analyze and compare the number of each operation, demonstrating that our methodology reduces computational cost. Finally, we comprehensively use our methodology to train different DNNs, and compare against other methodologies to demonstrate state-of-the-art performance.

A. Experimental Setup

1) Datasets and models: As shown in Table II, we evaluate four different DNNs as benchmarks. Since the experiments are conducted on an edge device with limited computation resources, we mainly use CIFAR10 as training dataset. The batch size is set to 32.
2) Hardware setup: The NVIDIA Jetson Orin Nano board, equipped with an INT8 engine, is used to measure training time, energy and memory usage. The device specifications, detailed in Table III, demonstrate that it is representative of edge devices used for DNN training.

TABLE II
DNN ARCHITECTURES AND DATASETS

DNN	Dataset	Num. of Params (M)
Multi-layer perceptron (MLP) (2 hidden layers)	MNIST	1.79
MobileNet-V2 [18]	CIFAR10	2.24
EfficientNet-B0 [19]		3.39
ResNet-18 [17]		11.19

TABLE III
TECHNICAL SPECIFICATIONS OF NVIDIA JETSON ORIN NANO

GPU	512-core NVIDIA Ampere architecture GPU
CPU	6-core Arm®Cortex®-A78AE v8.2 64-bit CPU
Memory	4GB 64-bit LPDDR5 34 GB/s
Power	7W-10W
AI Performance	20 TOPS

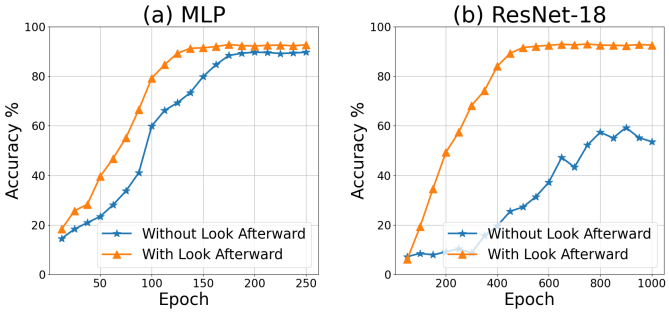

Fig. 6. Test accuracy across different epochs for MLP and ResNet-18 trained using FF-INT8, with and without "look-ahead" scheme respectively.

3) Hyperparameters: In Equation 1 and 2, we set threshold $\theta = 2.0$. In Equation 3, λ is initialized to 0, and increased by 0.001 each epoch.

B. Training with "Look-Ahead"

In the first set of experiment, we show that "look-ahead" scheme improves accuracy for FF-INT8 as described in Section IV-C. We first train MLP with 2 hidden layers using both FF-INT8 algorithms and demonstrate results in Figure 6(a). Without "look-ahead" scheme, the accuracy converges to nearly 90% with 180 epochs. However, with "look-ahead" scheme, MLP reaches slightly higher convergence accuracy with only 130 epochs. Thus, we argue that by interaction with later layers, earlier layers compute gradients and update weights in a "more optimized way", which leads to higher convergence accuracy and faster training.

Figure 6(b) demonstrates the results in ResNet-18, which consists of residue blocks. Different from MLP, if ResNet-18 is trained without "look-ahead" scheme, the accuracy converges to only 60%, while the accuracy curve is very unstable. There are two reasons for the bad performance:

1) The scale of ResNet-18 is much larger compared to MLP, which means it is much difficult for a greedy approach like FF algorithm to find the optimized solution.
2) Residue block allows input to be skipped over few layers and accumulated to block's output. However, without "look-ahead" scheme, when previous layers in a block are trained, the vanilla FF algorithm does not consider the accumulation operation in block's output layer. And when block's output layer is trained, the previous layers have already trained and cannot be modified. Thus, the optimization process is limited, which leads to low convergence accuracy.

TABLE IV
COMPARISON OF COMPUTATIONAL COST BETWEEN INT8 FORWARD-FORWARD ALGORITHM AND INT8/FP32 BACKPROPAGATION

Computation		Operation	Counts (OPs)
FF-INT8	Quantization Phase	32-bit CMP	32.4K
		32-bit FADD	165.9K
	MAC Phase	8-bit MUL	23.8M
		8-bit ADD	23.8M
BP-FP32	MAC Phase	32-bit FADD	898.2M
		32-bit FMUL	898.2M
GDAI8 [15] (BP-INT8)	Quantization Phase	32-bit CMP	7.2K
		32-bit FADD	18.4K
	MAC Phase	8-bit MUL	898.2M
		8-bit ADD	898.2M

As Figure 6(b) shows, by considering subsequent layers, "look-ahead" scheme solves both problems and significantly improves convergence accuracy. Given that many modern DNN architectures incorporate residual blocks, such method significantly enhances the FF-INT8 algorithm.

C. Analysis of Computational Cost

In Section V-B, our experiment suggests that FF-INT8 training requires a large number of epochs to converge. In this section, we analyze the theoretical computational cost to demonstrate that although FF-INT8 training has much more epochs compared to backpropagation, it is still efficient. We count the number of required operations to train 4-layer MLP using MNIST dataset with three settings, *i.e.* the proposed FF-INT8 training method with "look-ahead", and backpropagation with 32-bit floating-point (BP-FP32) as baseline. We also include Gradient Distribution-Aware INT8 training algorithm (GDAI8) [15] for comparison, which is an INT8 training algorithm based on backpropagation. Table IV summarize the amount of computation required for training a mini-batch of 10 samples for three approaches. In FF-INT8, we have a quantization phase and a multiply–accumulation (MAC) phase, where computation of quantization phase is negligible compared to MAC. Comparing MAC phase, FF-INT8 training requires 23.8M 8-bit MAC operations, whereas backpropagation approach (BP-FP32 or GDAI8) requires 898.2M MAC operations (FP32 or INT8). This is because the FF algorithm does not have large matrix multiplication to back-propagate derivatives from the last layer to the first. Thus, per mini-batch, FF-INT8 only requires 2.6% of MAC operations in backpropagation approach. INT8 arithmetic is also 4x faster than FP32 in hardware, which makes FF-INT8 hundres of times more efficient compared to BP-FP32 per epoch. Additionally, FF-INT8 only computes forward pass and does not compute backward pass of gradients. In many devices, forward pass is more efficient due to hardware optimization for model inference. In conclusion, although FF-INT8 training needs a large number of epochs, theoretically it is still more efficient compared to both backpropagation approaches.

TABLE V

SUMMARY OF MODEL ACCURACY, TRAINING TIME, ENERGY CONSUMPTION AND MEMORY FOOTPRINT BETWEEN DIFFERENT APPROACHES. TRAINING ALGORITHMS ARE BASED ON EITHER BP (BACKPROPAGATION) OR FF (THE FORWARD-FORWARD ALGORITHM). THE SUFFIX DENOTES THE PRECISION, WHERE FP32 MEANS 32-BIT FLOATING-POINT, AND INT8 MEANS QUANTIZING TO 8-BIT INTEGER. UI8 [13] REFERS TO UNIFIED INT8 TRAINING ALGORITHM, AND GDAI8 [15] REFERS TO GRADIENT DISTRIBUTION-AWARE INT8 TRAINING ALGORITHM.

Model	Training Algorithm	Accuracy (%)	Time (s)	Energy (J)	Memory (MB)
MLP	BP-FP32	94.5	482.3	2315.0	247.6
	BP-INT8	52.4	326.1	1206.6	213.9
	BP-UI8 [13]	92.3	335.2	1277.1	197.0
	BP-GDAI8 [15]	93.8	344.9	1345.4	182.6
	FF-INT8	94.3 (-0.2) (+0.5)	312.7 (-35.2%) (-9.3%)	1097.0 (-52.6%) (-18.4%)	140.7(-43.2%) (-22.9%)
MobileNet-v2	BP-FP32	91.5	2370.8	11593.2	649.8
	BP-INT8	5.9	1851.6	7836.0	571.6
	BP-UI8 [13]	87.2	1960.0	7618.5	592.6
	BP-GDAI8 [15]	90.9	1790.7	6528.1	578.9
	FF-INT8	91.1 (-0.4) (+0.2)	1703.9 (-28.1%) (-4.8%)	6174.3 (-46.7%) (-5.4%)	437.0 (-32.7%) (-24.5%)
EfficientNet-B0	BP-FP32	89.4	2692.8	13356.2	861.0
	BP-INT8	11.8	2095.0	8563.9	703.9
	BP-UI8 [13]	85.3	2230.8	8656.2	735.5
	BP-GDAI8 [15]	88.9	2177.1	8589.9	692.0
	FF-INT8	88.6 (-0.8) (-0.2)	2129.9 (-20.9%) (-2.2%)	8093.8 (-39.4%) (-5.8%)	505.2 (-41.3%) (-27.0%)
ResNet-18	BP-FP32	93.5	3853.0	18764.1	1096.4
	BP-INT8	7.2	2676.1	10436.8	885.8
	BP-UI8 [13]	89.7	2873.8	11466.5	920.7
	BP-GDAI8 [15]	92.9	2751.6	10291.0	894.1
	FF-INT8	93.1 (-0.4) (+0.2)	2697.9 (-30.0%) (-2.0%)	9926.5 (-47.1%) (-3.5%)	682.3 (-37.7%) (-23.7%)
Avg. difference between FF-INT8 and BP-FP32 (baseline)		**Reduce 0.4%**	**Save 28.6%**	**Save 46.4%**	**Save 38.7%**
Avg. difference between FF-INT8 and BP-GDAI8 (state-of-the-art)		**Improve 0.2%**	**Save 4.6%**	**Save 8.3%**	**Save 27.0%**

D. Accuracy, Time, Energy, Memory Footprint

For the last set of experiment, we compare model accuracy, training time, energy consumption between multiple training algorithms, and demonstrate results in Table V. As *baseline*, we train DNNs using backpropagation (BP) with 32-bit floating-point operations, denoted as BP-FP32. If directly quantizing gradients to INT8 using BP (BP-INT8), although we observe significant time, energy and memory savings, accuracy decreases dramatically due to the accumulation of quantization error, as analyzed in Section IV-A. With deeper architecture, direct quantization makes training even worse.

As a comparison, we train each DNN with two existing INT8 training algorithms, *i.e.* Unified INT8 Training Algorithm (BP-UI8) [13] and Gradient Distribution-Aware INT8 Training Algorithm (BP-GDAI8) [15]. Both algorithms are based on BP, and quantize gradients based on analysis of gradient distribution. Since they use BP with additional operation of gradient monitoring and analysis, the computational cost of both is slightly higher than direct quantization (BP-INT8). The training accuracy is close to traditional BP with FP32 operands (baseline). BP-GDAI8 has the *state-of-the-art* performance in terms of trade-off between model accuracy and efficiency, which is chosen as the comparison model.

Finally, we implement our FF-INT8 with "look-ahead" as described in Section IV-C. As analysis in Section IV-B, gradient quantization works better on FF algorithm due to its layer-by-layer greedy manner. Thus, the training accuracy is much higher compared to direct quantization in BP (BP-INT8). By comparing to BP-GDAI8, we show that training accuracy of FF-INT8 is 0.2% higher against the state-of-the-art, and is very close to the traditional backpropation (BP-FP32), with a 0.4% reduction. For training time and energy consumption, since Jetson Orin Nano board has an INT8 engine, we observe a significant improvement compared to training with FP32 operations. Our FF-INT8 algorithm is also more efficient compared to BP-GDAI8, with 4.6% saving in training time and 8.3% saving in energy consumption.

As mentioned in Section III, one of the major benefits of FF algorithm is smaller memory footprint. Normally, BP algorithms rely on automatic differentiation, which requires to store the large computational graph for gradient backpropagation in memory. But since FF algorithm does not have backward pass, it does not need to store this computational graph. Such improvement in memory footprint is further enhanced by INT8 operations. Thus, compared to BP-GDAI8, FF-INT8 saves memory footprint by 27.0%.

VI. CONCLUSION

In this paper, we present FF-INT8, a novel low-precision training methodology that leverages the Forward-Forward algorithm for INT8 quantized training. Unlike backpropagation-based approaches, FF-INT8 stabilizes gradient quantization by employing a layer-wise greedy training strategy, effectively mitigating accuracy degradation. To further enhance convergence and accuracy, we propose the "look-ahead" scheme, which enables earlier layers to incorporate feedback from subsequent layers, addressing a key limitation of the standard FF algorithm. By comparing against the state-of-the-art approach, we demonstrate that FF-INT8 significantly reduces training time, energy cost and memory footprint while maintaining high accuracy. FF-INT8 offers a promising alternative for energy-efficient neural network training.

REFERENCES

[1] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, "Compute trends across three eras of machine learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[2] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th international conference on parallel processing*, 2018, pp. 1–10.

[3] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[4] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, 2023.

[5] J. Ma and S. Reda, "Wenet: Configurable neural network with dynamic weight-enabling for efficient inference," in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2023, pp. 1–6.

[6] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[7] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," *arXiv preprint arXiv:2212.13345*, 2022.

[8] Y. He, L. Liu, J. Liu, W. Wu, H. Zhou, and B. Zhuang, "Ptqd: Accurate post-training quantization for diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[10] X. Chu, L. Li, and B. Zhang, "Make repvgg greater again: A quantization-aware approach," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 10, 2024, pp. 11 624–11 632.

[11] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[12] S. Fox, S. Rasoulinezhad, J. Faraone, P. Leong *et al.*, "A block minifloat representation for training deep neural networks," in *International Conference on Learning Representations*, 2020.

[13] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[14] K. Zhao, S. Huang, P. Pan, Y. Li, Y. Zhang, Z. Gu, and Y. Xu, "Distribution adaptive int8 quantization for training cnns," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 4, 2021, pp. 3483–3491.

[15] S. Wang and Y. Kang, "Gradient distribution-aware int8 training for neural networks," *Neurocomputing*, vol. 541, p. 126269, 2023.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[19] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.