Supplementary Material: FATNN: Fast and Accurate Ternary Neural Networks

Peng Chen, Bohan Zhuang, Chunhua Shen
Data Science & AI, Monash University, Australia

S1. Acceleration Details

As a fundamental operation in convolutional neural networks, the vector inner product is one of the core components in acceleration. In this section, we first elaborate more about the "non-overflow" property and the same parallelism degree as well as their importance to the fast implementation. Then we present the detailed implementation of the proposed fast ternary inner product and introduce its application in the convolutional and fully-connected layers.

Non-overflow property: If the range of input quantized vectors and the multiplication result in the inner product keeps the same, we term it "non-overflow" property. The multiplication result here, indicates the intermediate result of multiplication between the two input quantized vectors. Binary quantized values $\{1, -1\}$ and ternary quantized values $\{1, 0, -1\}$ are examples. We attribute the fast implementation of our FATNN to the "non-overflow" property because it enables the same parallelism degree for the multiplication (i.e., xnor) and accumulation (i.e., $\mathrm{popcount}$) operations. Specifically, for the BNNs case, 8 full-precision values can be packed into one byte. When xnor is employed for the multiplication, the parallelism degree is 8. Moreover, $\mathrm{popcount}$ also accumulates 8 data at the same time (same parallelism degree with the xnor operation). For the TNNs case, only 4 full-precision values can be packed into one byte. Thus, the parallelism degree of our $\mathrm{TM}(\cdot)$ in Eq. (3) is 4. Interestingly, the $\mathrm{popcount}$ operation can also accumulate the 4 data simultaneously. However, if standard 2-bit quantization values $\{0, 1, 2, 3\}$ are leveraged, the multiplication by combination of bit-wise operators (such as xnor) can own the parallelism degree of 4 if packed in byte. However, 4 bits are required to encode the multiplication result. Thus, the parallelism degree for the accumulation procedure is 2 at most (less than 4). Consequently, the computational efficiency will be halved. Based on the analysis, it can be learnt that the "non-overflow" property is an important attribute to enable the fast implementation of ternary and binary networks.

As explained in the Section 3, we design a fast implementation for TNNs by exploiting the "non-overflow" property. Algorithm 1 summarizes the inference flow of the proposed fast ternary inner product. In Algorithm 1, lines $2 \sim 5$ quantize the full-precision input vector into the codec of the quantized values. As 2 bits are required to encode one ternary value, 4 full-precision values can be packed into one byte. It is worth noting that it is also possible to pack the full-precision data into other data structures. For example, 8 full-precision data can be packed into short type or 16 full-precision data can be packed into 32-bit int type. More specifically, during the packing, each full-precision data is compared with the corresponding quantization thresholds characterized by $\{\alpha_1, \alpha_2\}$ and assigned to the corresponding codec value. After that, based on Eqs. (5), (6), (7), the accumulation is performed as lines $8 \sim 10$ in Algorithm 1. Finally the logic level inner product result is obtained according to Eq. (3) in line 11.

Algorithm 1: Fast Ternary Inner Product

Input: (1): Full-precision weight vector $\boldsymbol{w} \in \mathbb{R}^N$ and activation vector $\boldsymbol{a} \in \mathbb{R}^N$. (2): Pre-allocated temporary buffer $\hat{\boldsymbol{w}}$ and $\hat{\boldsymbol{a}}$ with unsigned char type in the length of $N/4$. (3): Quantization parameters α_1^w, α_2^w and α_1^a, α_2^a which are used to parameterize the step sizes for weights and activations, respectively.

Output: The ternary inner product result z.

1 Step (1): data packing;
2 **for** $i \leftarrow 0$ **to** $\frac{N-1}{4}$ **do**
3 Pack 4 values of $\boldsymbol{a}[4i : 4i+3]$ into one unsigned char and store it in $\hat{\boldsymbol{a}}[i]$;
4 Pack 4 values of $\boldsymbol{w}[4i : 4i+3]$ into one unsigned char and store it in $\hat{\boldsymbol{w}}[i]$;
5 **end**
6 Step (2): ternary inner product;
7 acc = 0;
8 **for** $i \leftarrow 0$ **to** $\frac{N-1}{4}$ **do**
9 acc += popcount(TM($\hat{\boldsymbol{w}}[i]$, $\hat{\boldsymbol{a}}[i]$));
10 **end**
11 $z = acc - N$

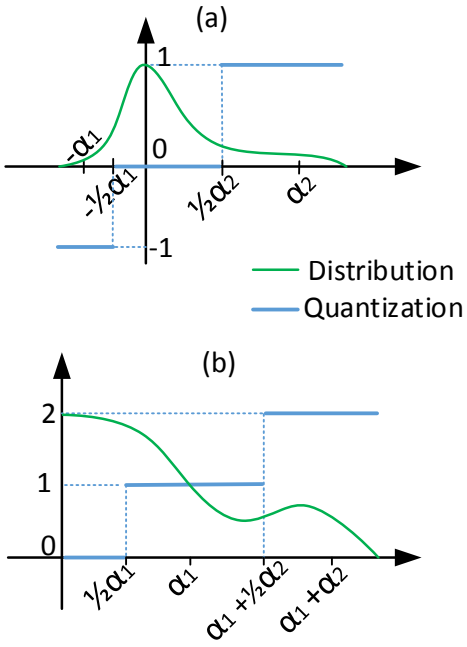
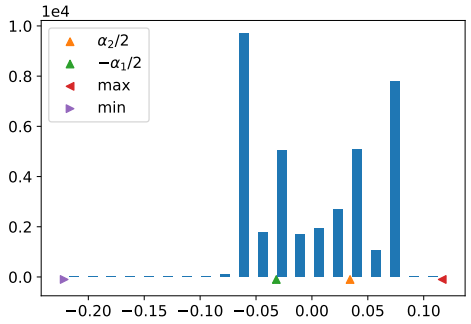


Figure S1: The proposed non-uniform ternary quantization for (a) a tensor in the real domain; (b) a tensor which only contains non-negative values. The vertical axis represents the quantized domain and the horizontal axis denotes the real domain. The green curve indicates the distribution of the full-precision tensor and the blue line shows the quantized values by discretizing the full precision data according to the learned thresholds. We aim to learn the optimal step size of each quantization level.

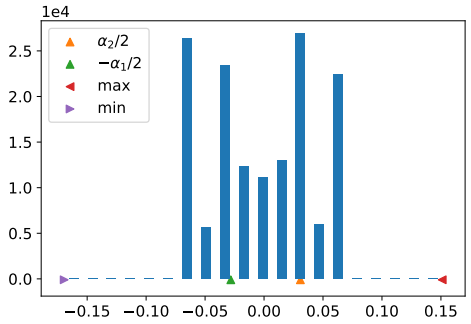
Following the implementation of the fast ternary inner product, the convolutional layer can be realized by first expanding the input activation into a matrix (im2col) and then conducting the matrix multiplication (gemm). To enhance the efficiency, the data packing in Algorithm 1 is integrated into the im2col operation. After that, the matrix multiplication is realized based on the inner product step in Algorithm 1. Tricks, such as winograd [9], are commonly employed in the gemm operation, however we do not integrate these tricks for simplicity. The fully-connected layer is similar to the implementation of the convolutional layer, which can be regarded as a special case of the latter with kernel size being equal to the feature map resolution. Other operations, such as the ReLU non-linearity and skip connection layers, can be fused in the im2col procedure. Besides, we fuse the batch normalization layers into the corresponding convolutional or fully-connected layers.

S2. Visualization of Non-uniform Step Sizes

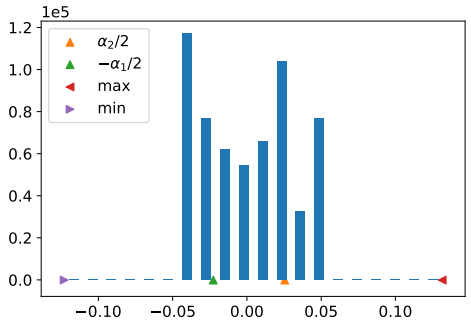
In this paper, we discretize the full-precision tensor into the ternary quantized values with trainable non-uniform step sizes. For each quantized layer, we learn two parameters $\alpha = \{\alpha_1, \alpha_2\}$ for weights and activations separately.



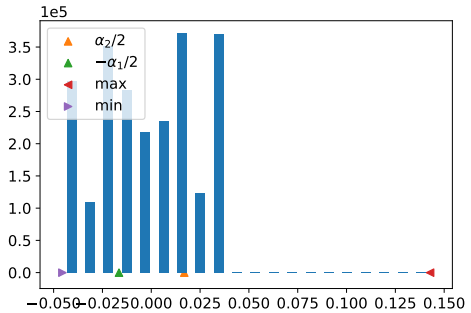
(a) layer1_1_conv2



(b) layer2_1_conv2

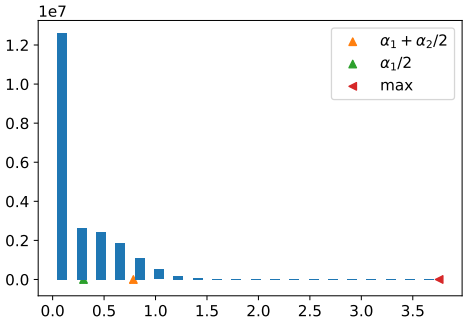


(c) layer3_1_conv2

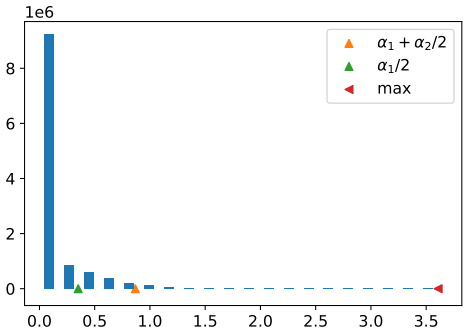


(d) layer4_1_conv2

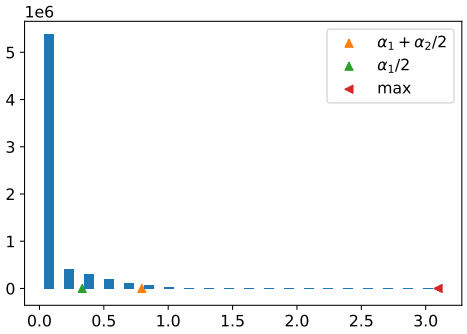
Figure S2: Weight quantization for ResNet-18.



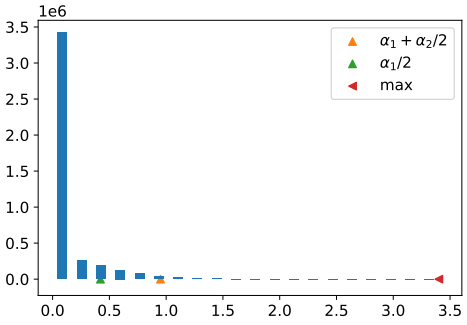
(a) layer1_1_conv2



(b) layer2_1_conv2



(c) layer3_1_conv2



(d) layer4_1_conv2

Figure S3: Activation quantization for ResNet-18.

As illustrated in Figure S1, the quantization thresholds are directly related to the learnt parameters α. When the two scale factors are identical ($\alpha_1 = \alpha_2$), the proposed quantization algorithm reduces to the uniform step size quantization [2, 11, 3]. Thus, it is interesting to investigate the properties of the learnt α. We plot the distribution of the full-precision weights and activations as well as the corresponding quantization thresholds on ResNet-18 in Figure S2 and S3, respectively. We list the statistics of four layers in ResNet-18. Besides, the max value, min value and the two quantization thresholds of the tensors are marked in each sub-figure. On the one hand, from Figure S2, we observe that the distribution of the weight in the model varies a lot in different layers. In order to reduce the information loss during quantization, we propose to learn the quantization thresholds automatically to better fit the data distribution. On the other hand, Figure S3 demonstrates that, the full-precision activations consist of dense relative small values and sparse relative large values. The number of elements of each interval is unbalanced. It can be seen from Figure S3 that the quantization step sizes learnt based on the stochastic gradient descent are non-uniform ones and differ at different layers.

S3. More Execution Time Benchmarks

We present more acceleration benchmark results in this section.

Firstly, for convenience of comparison, we list the the exact execution time for the layer-wise benchmark described in experiments section in Table S1 and Table S2, respectively. We use Q821/Q835/2080Ti to indicate the Qualcomm 821, 835 and Nvidia 2080Ti separately. Besides, "bin" represents binary and "ter" means ternary in Table S1 and Table S2. We measure the layer-wise execution time for convolution layers (kernel size = 3×3, padding = 1, stride = 1, the same number of input and output channels and batch size = 1) with six different shape configurations. For the first four cases, we fix the channel number to be 64 and increase the resolution from 28 to 224. For the last two cases, we fix the resolution to be 56 and double the channel number from 64 to 256. When focusing on the speedup of the proposed ternary network against the conventional 2-bit implementation (for example, the Figure 2 or the 'ter/ter' and '2/2' rows in Table S1 and Table S2), the theoretical speedup ratio for all cases is 2. However, comparing the first 4 cases (doubled width/height) or the second and the last two cases (doubled channel), speedup decreases then becomes stable. The reason is that the speedup is highly impacted by the hardware utilization, which is further related to the task size. When the task size is small, the hardware unit is only partially used. As the feature map size increases, hardware utilization is improved and the speedup becomes more stable.

Secondly, we conduct more test for the overall network

Table S1: Exact execution time (ms) on embedded-side platforms. We run 5 times and report the mean results.

Device	A/W	case1	case2	case3	case4	case5	case6
Q821	bin/bin	0.6	1.2	2.6	8.3	2.3	6
	ter/ter	0.9	1.7	4.4	14.8	4.2	13.5
	2/2	1.9	3.5	8	24.9	7.2	21.2
Q835	bin/bin	0.7	0.9	1.9	6.1	1.8	4.8
	ter/ter	0.9	1.5	3.7	13.2	3.2	12.7
	2/2	2.1	3	6.5	20.5	5.6	15.8

Table S2: Exact execution time (μs) on server-side platforms. We run 5 times and report the mean results.

Device	A/W	case1	case2	case3	case4	case5	case6
2080Ti	bin/bin	11	12.5	19.5	56.5	22	55.5
	ter/ter	11	14.5	26	90	34	87
	2/2	38	44	73.5	274	91	217.5

with commercial Qualcomm SNPE SDK. The half floating-point ResNet-18 by the Qualcomm SNPE SDK costs 44.7 ms on Qualcomm 821 (batch size = 1 on ImageNet) while the ternary counterpart is 25.2 ms. The practical speedup is impacted by high-precision MISC operations (such as im2col) and modern hardware architectures (such as separate pipelines for floating-point and fixed-point computation).

S4. Evaluation on CIFAR-10

S4.1. Training hyper-parameters on CIFAR-10

We follow the hyper-parameter setting in previous works [10, 8] to train the networks on the classification task. Specifically, for ResNet-20 on CIFAR-10, we train up to 200 epochs. The initial learning rate starts from 0.1 and is divided by 10 at epoch of 82 and 123, respectively. We use a weight decay of 1e-4 and a batch size of 128. For VGG-Small on CIFAR-10, the learning rate begins with 0.02 and is divided by 10 at epoch of 80 and 160, separately. The weight decay is set to be 5e-4, batch size to be 128 and total epochs to be 200. For NIN on CIFAR-10, we train 90 epochs with the initial learning rate to be 1e-2. The learning rate is divided by 10 at epoch of 30 and 60. Weight decay is set to be 1e-5.

S4.2. Comparison on CIFAR-10

We further demonstrate the effectiveness of the proposed method on CIFAR-10 [4] dataset. For CIFAR-10, the input images are firstly padded with 4 pixels and then cropped into 32×32 samplings. Random horizontal flip is employed for data augmentation. NIN [6], VGG-Small [7] and ResNet-20 are employed for the evaluation. Comparisons between our FATNN and other quantization algorithms are listed in Table S3.

Table S3: Top-1 accuracy (%) comparisons between our FATNN and other algorithms, including RTN [5], HWGQ [1], LQ-Net [10] and TSQ [8] on CIFAR-10 dataset.

Method	A/W	ResNet-20	NIN	VGG-Small
	32/32	92.1	89.8	93.8
FATNN	ter/ter	**90.2**	**89.9**	**93.7**
RTN	ter/ter	-	89.6	-
HWGQ	2/1	-	-	92.5
LQ-Net	2/1	88.4	-	93.4
TSQ	2/ter	-	-	93.5
LQ-Net	2/2	90.2	-	93.5

From Table S3, we observe that for ResNet-20 and VGG-Small, the proposed method is able to achieve comparable accuracy for ternary networks compared with higher bit counterparts ("2/2" or "2/ter") quantized by LQ-Net and TSQ. Moreover, it surpasses RTN, HWGQ and LQ-Net on all the corresponding "2/1" or "ter/ter" networks. For NIN, our FATNN even beats the full-precision counterpart. It implies that network quantization has certain kind of regularization effect to suppress the over-fitting problem. Overall, the proposed FATNN demonstrates superior performance on the CIFAR-10 dataset.

References

[1] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 5918–5926, 2017. 4

[2] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *Proc. Int. Conf. Learn. Repren.*, 2020. 3

[3] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and

Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4350–4359, 2019. 3

[4] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 4

[5] Yuhang Li, Xin Dong, Sai Qian Zhang, Haoli Bai, Yuanpeng Chen, and Wei Wang. Rtn: Reparameterized ternary network. In *AAAI*, 2020. 4

[6] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 4

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. Int. Conf. Learn. Repren.*, 2015. 4

[8] Peisong Wang, Qinghao Hu, Yifan Zhang, Chunjie Zhang, Yang Liu, and Jian Cheng. Two-step quantization for low-bit neural networks. In *Proc. IEEE Conf. Comp. Vis. Patt. Recogn.*, pages 4376–4384, 2018. 4

[9] Athanasios Xygkis, Lazaros Papadopoulos, David Moloney, Dimitrios Soudris, and Sofiane Yous. Efficient winograd-based convolution kernel implementation on edge devices. In *Proc. 55th Annual Design Automation Conf.*, DAC '18, pages 136:1–136:6, New York, NY, USA, 2018. 2

[10] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proc. Eur. Conf. Comp. Vis.*, 2018. 4

[11] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 3