Sparsification of Deep Neural Networks via Ternary Quantization

2024 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 22–25, 2024, LONDON, UK

SPARSIFICATION OF DEEP NEURAL NETWORKS VIA TERNARY QUANTIZATION

Luca Dordoni[⋆]*, Andrea Migliorati*[⋆]*, Giulia Fracastoro*[⋆]*, Sophie Fosson*[†]*, Tiziano Bianchi*[⋆]*, Enrico Magli*[⋆]

[⋆] Department of Electronics and Telecommunications, Politecnico di Torino, Turin, Italy

[†] Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy

ABSTRACT

In recent years, the demand for compact deep neural networks (DNNs) has increased consistently, driven by the necessity to deploy them in environments with limited resources such as mobile or embedded devices. Our work aims to tackle this challenge by proposing a combination of two techniques: sparsification and ternarization of network parameters. We extend the plain binarization by introducing a sparsification interval centered around 0. The network parameters falling in this interval are set to 0 and effectively removed from the network topology. Despite the increased complexity required by the ternarization scheme compared to a binary quantizer, we obtain remarkable sparsity rates that yield parameter distributions with significantly compressible sources with entropy lower than 1 bits/symbol.

***Index Terms*—** Quantized Neural Networks, Ternary Neural Networks, Network Sparsification

1. INTRODUCTION

Since their introduction, the demand for more memory-efficient neural networks has increased considerably [1–3]. For this reason, researchers have focused on developing techniques to reduce memory consumption and the size of the newer architectures. Among these methods is quantization [4], a technique used to reduce the precision of numerical values of the parameters within the network. By converting these values from high precision (e.g., 32-bit floating point) to lower precision (e.g., 8-bit integers), quantization enables significant reductions in model size, memory footprint, and computational complexity. This optimization is particularly valuable for deploying neural network models on devices with limited resources, such as mobile phones, embedded systems, and IoT devices, where memory is limited.

In this paper, we introduce a flexible ternary quantization technique and employ it to induce sparsity in the architecture of deep neural networks. Specifically, in our ternary model, parameters are quantized to $\{-1, 0, +1\}$. We quantize to 0 the parameters with magnitude smaller than a given threshold $\Delta > 0$ and we effectively remove them for the network topology. This approach aims to sparsify and quantize network architectures right from the initial stages of the learning process, employing quantization-aware training. To boost sparsification reaching high sparsity rates, we introduce a growth regime for Δ based on a specific function of choice. Our ternarization scheme differs from a static binary quantization framework that quantizes to $\{-1, +1\}$, see e.g. [1, 2], as it allows for parameters that have been previously removed from the topology to be introduced back into the network's structure if this would ensure a smaller loss during training. In other words, the parameters' update drives the choice of which parameter should be set to zero (i.e. which node has to be pruned) to maximize accuracy and sparsity.

Our contributions can be summarized as follows: *(i)* we tackle the challenge of reducing network resource consumption by proposing a ternary approach that combines the sparsification and quantization of DNNs; *(ii)* we promote network sparsification by introducing growth regimes for the Δ threshold hyper-parameter, increasing the number of model parameters set to 0 allowing for remarkably high sparsity rates; *(iii)* our ternary DNN architectures achieve improved classification accuracy compared to their binary counterparts, while also achieving higher compression rates.

2. BACKGROUND

In this section, we list relevant works in the field of quantized neural networks and ternarization.

2.1. Quantization

The most common quantization approach is performed as in the following:

$$Q(r) = \mathrm{Int}(r/S) - Z, \tag{1}$$

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU with a partnership on "Telecommunications of the Future" (PE00000001 "RESTART" program). This study was also carried out within the Future Artificial Intelligence Research (FAIR) and received funding from the European Union Next-GenerationEU (PNRR. Piano Nazionale di Ripresa e Resilienza, Missione 4, Componente 2, Investimento 1.3 - D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions, neither the EU nor the European Commission can be considered responsible for them. The contribution of Sophie Fosson is part of the project NODES which has received funding from the MUR-M4C2 1.5 of PNRR with grant agreement n. ECS00000036.

979-8-3503-2411-2/23/$31.00 ©2024 IEEE

where r is the parameter that needs to be quantized, S is the scaling factor, Z is an integer representing the origin point, and $\mathrm{Int}(\cdot)$ is the rounding function. The scaling factor is a parameter that defines the step between quantized levels, and it is computed as $S = \frac{\beta-\alpha}{2^b-1}$, where $[\alpha, \beta]$ represents the clipping range, and b is the quantization bit-width. The choice of the clipping range choice (or *calibration*) leads to symmetric quantization when $|\alpha| = |\beta|$, and to asymmetric quantization if $|\alpha| \neq |\beta|$.

2.2. Straight-Through Estimator

The backpropagation algorithm employed to train deep models and specifically to compute gradients encounters challenges with non-differentiable functions. To address this problem, [5] proposed a solution that goes by the name of *straight through estimator* (STE). STE consists of replacing the gradients of non-differentiable functions with the gradient of the identity function during the backward pass, allowing the flow of information through non-differentiable elements. This strategy enables the incorporation into the backpropagation process of operations whose derivatives are not well-defined, such as ternarization. In this work, we use a version of the STE that accounts for the saturation effect [2], as outlined in Eq. 2:

$$g_r = g_q \times \mathbb{1}_{|r|\leq 1}, \tag{2}$$

where g_r is the gradient of the loss with respect to r, g_q is an estimator of the gradient $\frac{\partial L}{\partial q}$, q and r are related through a non-smooth $q = \mathrm{Ternarize}(r) \times \mathbb{1}_{|r|\leq 1}$ performs clipping to treat saturation. In the backward pass, the non-differentiable operation is bypassed by replacing it effectively with an identity function.

2.3. Binarization and Ternarization

The pioneering BinaryConnect proposed in [1] brought network compression to its limit by introducing a novel architecture that used 1-bit parameters, giving rise to binarization techniques. Since then, numerous studies have explored low-bit width quantization, including 1-bit binary and 2-bit ternary parameters. In [6], the authors introduced Ternary Weights Networks (TWNs) exploiting the ternarization of DNNs. In TWNs, every parameter is quantized to $+1$, 0, or -1, therefore occupying 2 bits on average. Weights are quantized via a thresholding function based on a positive threshold parameter computed by minimizing the Euclidean distance between the quantized parameter and the full-precision (FP) one.

In [7], a Trained Ternary Quantization (TTQ) was proposed, based on the definition of weights $\{-W_n^{(l)}, 0, W_p^{(l)}\}$ instead of $\{-1, 0, +1\}$, with the superscript l standing for the layer number, i.e. these quantization values are layer-dependent. Δ_l is computed by employing a global parameter t, according to Eq. 3.

$$\Delta^{(l)} = t \times \max\left(|W^{(l)}|\right). \tag{3}$$

The work [8] proposes a ternary approach called EC2T that aims to sparsify and quantize the architecture based on two stages: compound model scaling, where a super-network is built starting from a pre-trained model whose dimensions are scaled to obtain an over-parametrized architecture, and quantization, where ternary (centroid) values are assigned relying on the novel cost function:

$$\mathbf{C}_c^{(l)} = d\left(\mathbf{W}^{(l)}, w_c^{(l)}\right) - \lambda^{(l)} \log_2\left(P_c^{(l)}\right), \tag{4}$$

where $\mathbf{C}_c^{(l)}$ represents the cost associated with assigning quantized values to the full-precision weights $\mathbf{W}^{(l)}$ given the centroids $w_c^{(l)}$ for the layer l. The cost function (4) is particularly effective because it introduces an entropy term to promote the sparsity of the architecture. In [9], the authors proposed a new framework, called FATNN, where the ternary representation is fully leveraged via a series of bitwise operations aimed at achieving optimized ternary inner product. Specifically, FATNN successfully reduces the computational complexity of TNNs by $2\times$. Li et al. [10] reimagined the role of thresholding operations in their proposed ternary quantization method called TRQ. In TRQ, ternary weights are generated as a combination of binarized stem and residual components, achieved by recursively quantizing full-precision parameters. More recent works have focused on ternary neural networks. Concerning hardware, a Ternary-CIM (T-CIM) processor with 16T1C ternary bitcel is proposed in [11], solving previous CIM processor issues such as low energy efficiency, throughput and poor linearity of analog-computing. T-CIM tackles these problems by using decision thresholding and by removing analog-to-digital conversion. Moreover, ternary models gained particular attention in [12] which shows that ternary quantization can be effectively applied to transformers and LLMs to achieve remarkable performance while being significantly more resource-efficient compared to their full-precision versions.

3. PROPOSED METHOD

This section introduces the proposed method and the rationale behind the proposed ternarization scheme. Firstly, we describe the standard ternarization employed in our work. Secondly, we discuss the optimization of the network parameters. Finally, we introduce growth regimes for Δ to enhance sparsification, along with a qualitative analysis of the upper bound to the growth of Δ. Sparse architectures entail simpler computational calculations as most operations in the training and inference phase are multiplications that can be readily set to 0 when parameters are zeroed out, effectively not contributing to the network's output and consequently to the loss computation. Moreover, using ternarized parameters allows the model

to be compatible with dedicated deep-learning hardware for even more efficient computations [6].

3.1. Ternarization

We employ a quantization scheme based on the ternary values of $\{-1, 0, +1\}$. The parameter quantization is performed with the thresholding operation as in Eq. 5:

$$\theta_q = \begin{cases} +1 & \text{if } \theta > \Delta \\ 0 & \text{if } |\theta| \leq \Delta \\ -1 & \text{if } \theta < -\Delta, \end{cases} \tag{5}$$

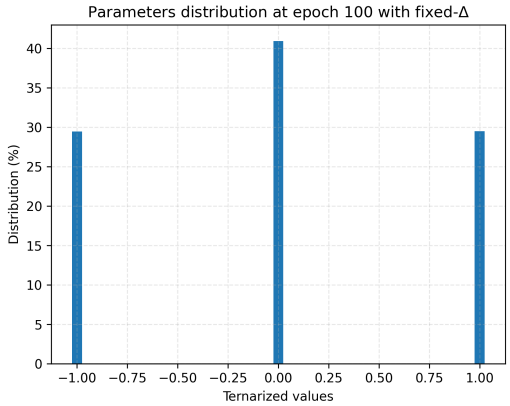
where θ is the full-precision parameter and θ_q is its quantized version, similar to how ternarization is performed in TWN. Our method harnesses quantization-aware training that requires a copy of the full-precision (FP) parameters to compute the gradients for the quantized parameters, as explained in Sec. 2. Following the common practice for quantized frameworks [1, 2], we clip the FP parameters to $[-1, +1]$ before ternarizing to prevent them from drifting away from the $[-1, +1]$ interval.

3.2. Δ Growth Regimes

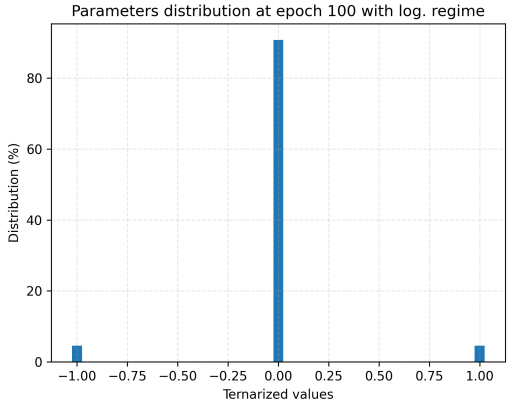
Our preliminary experimental findings led us to consider increasing Δ as the training progresses to achieve greater sparsity rates than the ones obtained with fixed Δ values. We introduce Δ growth regimes as in Eq. 6, representing the growth of Δ as a function of the training epoch:

$$\Delta_{new} = \Delta_0 + \Delta_0 \times M \times f\left(\text{Epoch}\right), \tag{6}$$

where Δ_{new} is the current-iteration threshold value, Δ_0 is the initial Δ value, and M represents a constant multiplier used to adjust the shape of the growth. f is a function of the training epoch ($Epoch$), we evaluate various functions: linear ($f(x) = x$), square ($f(x) = x^2$), exponential ($f(x) = e^x$) and logarithmic ($f(x) = \log(x)$). The reason behind the introduction of threshold regimes is that, in the *standard* fixed-Δ configuration, weights undergo random initialization and their subsequent optimization updates quickly lead them to spread beyond the Δ threshold value. This implies that, during the initial training iterations, a significant portion of the parameters is already quantized to either -1 or $+1$, while their full-precision counterparts fluctuate around these values, far outside the $[-\Delta, \Delta]$ interval. Hence, introducing a Δ threshold growth regime (6) allows the interval to cope with the rapid evolution of the parameters' values. This approach ensures that the ternarization threshold adapts to the nature of the parameter update, enabling efficient model sparsification throughout the entirety of the training phase. Fig. 1 represents the distributions of quantized parameters for fixed-Δ and increasing Δ frameworks, utilizing a threshold regime results in parameter distributions that are more concentrated around 0, leading to greater sparsity rates.



(a) Fixed-Δ



(b) Logarithmic growth

Fig. 1. A comparison between the distributions of fixed and logarithmic Δ ternarization regimes: (a) Parameter distribution for a fixed $\Delta = 0.1$ at epoch 100. (b) Parameter distribution for logarithmic growth ($\Delta_0 = 0.1$ and $M = 1.9$) at epoch 100.

Finally, our ternarization scheme also considers the imposition on Δ of a Δ_f maximum value that prevents the Δ threshold from increasing beyond a critical value. Indeed, Δ could reach an absolute value equal to 1, leading to overlapping intervals in Eq.5. Ultimately, the employed regime equation is:

$$\Delta_{new} = \min\left(\Delta_0 + \Delta_0 \times M \times f\left(\text{Epoch}\right), \Delta_f\right). \tag{7}$$

4. EXPERIMENTAL RESULTS

In the subsequent section, we first examine the experimental setup and metrics utilized for result comparison. Afterwards, experimental evidence is showcased, with focus on the effect of the initial threshold values, and the analysis of the optimal growth regime.

4.1. Setting

The experiments were performed on image classification tasks, employing a slightly modified ResNet-20 architecture [13]. Specifically, we increased the network inflation parameter to 5 instead of 1, leading to an increased number of parameters (4338030 over 175406), we refer to this architecture as ResNet-20*. We train and test the models for a total of 500 epochs on the CIFAR-10 dataset [14], which comprises 60000 32×32 colour images divided into 10 classes, and split into 50000 training images and 10000 test images. The model processes images in batches of fixed size of 256, with a learning rate (λ) that undergoes the scheduled decay: $[5\text{e}{-3}, 1\text{e}{-3}, 5\text{e}{-4}, 1\text{e}{-4}, 1\text{e}{-5}]$ respectively set at epoch $[0, 101, 142, 184, 220]$. The network parameters are initialized by sampling from a normal distribution $\theta \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$, where n is the number of parameters in the convolutional kernel. All models are implemented in PyTorch [15] and run on NVIDIA GeForce GTX Titan X GPUs.

4.2. Metrics

We evaluate our proposed method using four key metrics: *Top-1 validation accuracy:* which measures the percentage of correctly predicted labels; *Sparsity:* indicating the percentage of zeroed-out parameters in the neural network; *Entropy:* derived from the distribution of ternarized parameters and expressed in bits/symbol, measuring the average number of bits needed to represent the information content of a random variable; and *Training speed:* denoting the epoch at which peak accuracy is achieved.

4.3. Δ_0 Effect on Performance

First, we investigate the effects of Δ_0 for both fixed-Δ and Δ growth regime scenarios. As shown in Fig. 2, Δ_0 values of 10^{-1}, 10^{-2}, and 10^{-3} enable the model to achieve accuracy on the test set of over 90%, while values greater than $1\text{e}{-1}$ excessively sparsify the network during the initial training phase, thus disrupting the learning process. As discussed in Section 3.2, keeping Δ fixed during training leads to models with lower sparsity rates. Indeed, the best results are obtained with $\Delta_0 = 0.1$ whose sparsity rates hardly exceed 44%, as reported in Table 1). In the following, we analyse how increasing Δ with different growth regimes is beneficial for the accuracy and sparsity of the model.

Table 1. Accuracy and sparsity performance for $\Delta_0 = 0.1$.

	Outcomes					Mean	SD
Accuracy	91.01	91.45	91.47	91.56	91.45	91.39	0.22
Sparsity	44.65	45.01	43.43	44.24	43.21	44.11	0.77

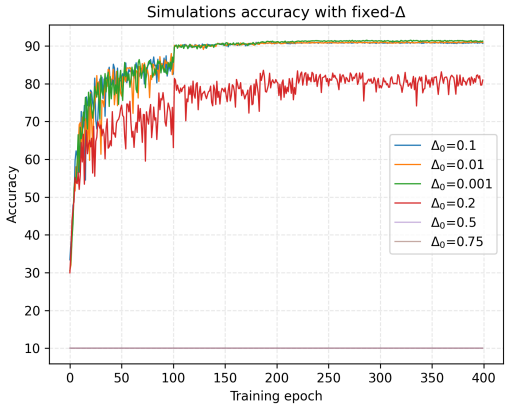


Fig. 2. Accuracy for different Δ_0 values in the fixed-Δ scenario. Initial Δ values above 0.1 yield sub-optimal accuracy.

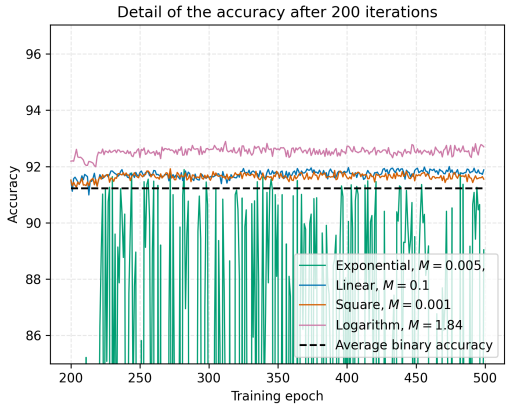
4.4. Evaluating Different Δ Growth Regimes

In Section 3.2, we introduced the concept of employing growth regimes for Δ to maximize the sparsity of the model. Generally speaking, a higher Δ value entails greater rates of zeroed-out parameters. However, if Δ is too large, it can lead to excessively high sparsity rates, resulting in highly oscillating validation accuracy, and, in more extreme cases, a significant accuracy drop. Hence, the choice of the function f is of key importance.

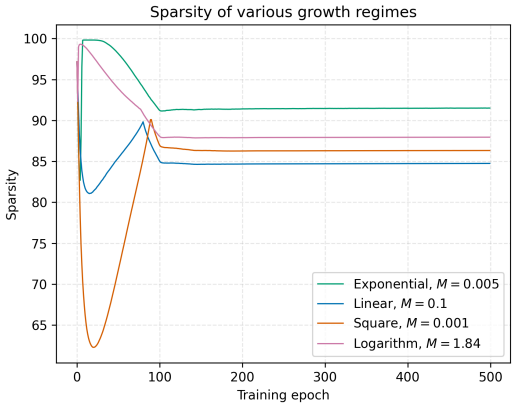
Here, we report the accuracy and sparsity of the models obtained with linear, quadratic, exponential, and logarithmic Δ growth regimes. Results are shown in Fig. 3. Our ternary architecture obtains great sparsity rates, averaging around 90%. We also achieve remarkable accuracy on the test set compared to the binary architecture counterpart (i.e. when parameters are straightforwardly binarized). One can notice that exponential regimes tend to induce sparsification early in the training, primarily due to the rapid increase in the Δ threshold. This inevitably leads to a large portion of the parameters being quantized to 0, and consequently to models whose learning abilities are limited by such distribution of the ternary parameters.

With accuracy exceeding 92% and sparsity exceeding 85%, the logarithmic growth simulation produces the best results among the considered growth regimes. We further substantiate our experimental evaluation by testing multiple Δ_f, M, and f combinations for a fixed $\Delta_0 = 0.1$. We report the results by pairing test accuracy, sparsity, entropy, and training speed in three plots.

Fig. 4(a) plots accuracy against sparsity rates, such that the best-performing configurations sit in the top-right corner. Once again, logarithmic growth regimes appear to be the most suitable choice as they yield the highest accuracy on the test set while losing only a few percentage points in terms of sparsity rate compared to others. Specifically, they reach accuracy

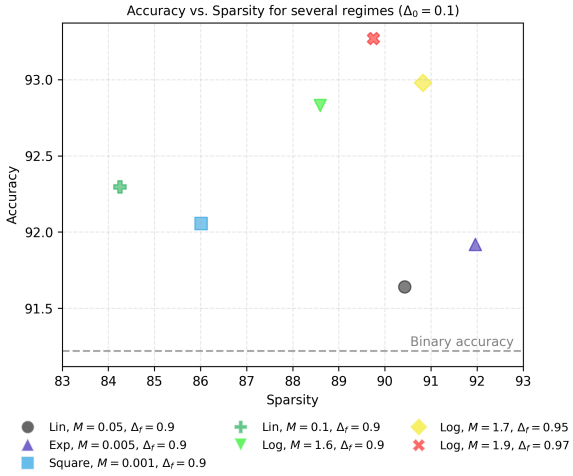


(a) Accuracy oscillations after training convergence.

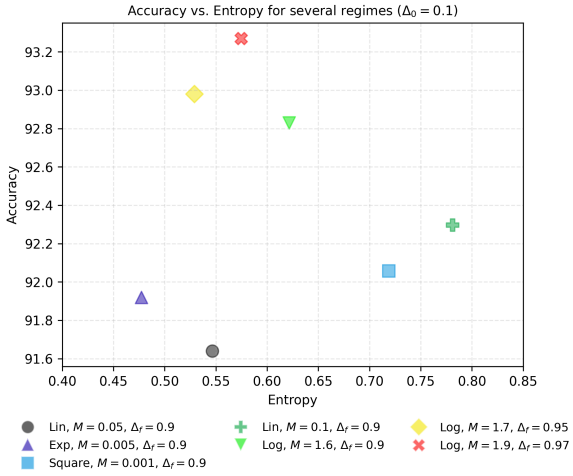


(b) Sparsity rate as a function of the training epoch.

Fig. 3. The figures depict the accuracy and the sparsity for several threshold regimes with $\Delta_f = 0.9$. Except for the exponential growth regime, which gives rise to excessive oscillations, all experiments exhibit remarkable sparsity and accuracy results.



(a) Accuracy vs. Sparsity.



(b) Accuracy vs. Entropy.

Fig. 4. Ternarized models performance for different f, M, and Δ_f.

levels of approximately 93.5% and have sparsity percentages close to 90%. It is worth noticing that all ternarized models we obtain outperform their binary counterparts (dotted line in Fig. 4(a)) by a significant amount (up to 2% for the logarithmic growth regime). The sparsity rates are correlated to the value of the growth upper bound Δ_f: the higher Δ_f, the higher they are. However, as previously observed, greater Δ_f values may lead to instability during training.

Fig. 4(b) reports accuracy against entropy for the obtained models. Interestingly enough, one can observe that, as high sparsity rates are caused by a high concentration of parameters around 0, our ternary models exhibit lower entropy than a binarized one whose entropy instead averages around 1 bits/symbol. Specifically, the entropy values for the logarithmic configurations are approximately 0.6 bits/symbol. Our findings suggest that the improvement in terms of classification accuracy over binary models also comes with a great compressibility of the ternarized model. Furthermore, we analyze the relationship between accuracy and training speed and report results in Fig. 5. We run three simulations for each setup to consider training variability. Our findings show no significant differences in training speed among the different growth regimes we experimented with. The tested configurations reach their highest accuracy values with comparable training speeds, proving consistency in how early the best accuracy is obtained across the different regimes.

Finally, Table 2 reports accuracy performance compared with competing approaches and full-precision baselines. The first section in Tab. 2 refers to results obtained with a ResNet-20 architecture, while the second shows performance obtained from a ResNet-20* model (i.e. inflation parameter set to 5). Our approach produces the best results for top-1

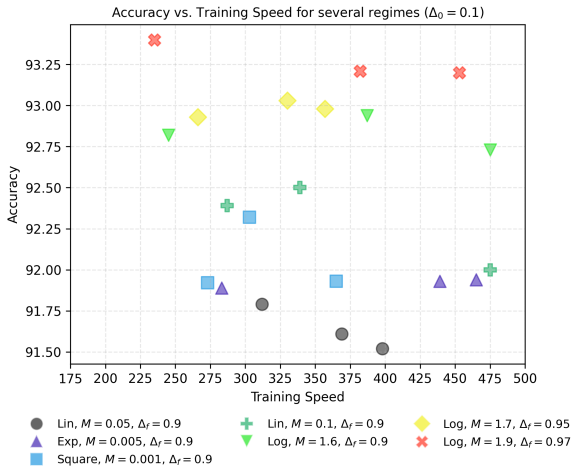


Fig. 5. Accuracy analysis for different ternary configurations as a function of the training speed. Each experiment is performed three times.

Table 2. Comparison of Δ growth regimes with FP baselines, binary counterparts, and selected ternary approaches. The table consists of two sections: the first compares results obtained with ResNet-20 models, while the second reports results obtained with ResNet-20*.

	Acc. (%)	**Sparsity** (%)	**Entropy** (bits/sym)	**Acc. Diff.** (%)
ResNet-20 (FP)	91.67	0.00	-	-
TTQ	91.13	30-50	-	-0.54
EC2T-1 ($\lambda = 0$)	91.16	45.17	-	-0.51
EC2T-2 ($\lambda > 0$)	90.76	73.26	-	-0.91
ResNet-20 (FP)*	93.61	0.00	-	-
ResNet-20* (Binary)	91.22	0.00	1.00	-2.39
Ours (lin. regime)	92.30	84.25	0.78	-1.31
Ours (log. regime)	**93.27**	**89.75**	**0.57**	**-0.34**

classification accuracy and sparsity for quantized models, exhibiting the smallest deviation from baseline accuracy, leading to models with significant performance and great compressibility.

5. CONCLUSIONS

In this paper, we present a flexible framework for ternary quantization that combines both sparsification and quantization. Our method allows parameters previously removed from the topology to be weighted back into the network's structure, driven by the parameters' update process based on STE. The proposed approach also introduces Δ threshold growth regimes to improve the sparsity of ternarized models during training while maintaining high accuracy on the image classification tasks. We show that this method yields neural network frameworks with sparsification rates over 90% and improvements up to 2% in top-1 validation accuracy compared with binary models. Our findings demonstrate that the notable sparsity rates lead to parameter distributions with lower entropy levels than binary models, yielding more compressible architectures with higher generalization capability.

6. REFERENCES

[1] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or -1," *arXiv preprint arXiv:1602.02830*, 2016.

[3] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[4] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[6] B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.

[7] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *International Conference on Learning Representations*, 2017.

[8] A. Marban, D. Becking, S. Wiedemann, and W. Samek, "Learning sparse ternary neural networks with entropy-constrained trained ternarization (ec2t)," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 3105–3113.

[9] P. Chen, B. Zhuang, and C. Shen, "Fatnn: Fast and accurate ternary neural networks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5199–5208.

[10] Y. Li, W. Ding, C. Liu, B. Zhang, and G. Guo, "Trq: Ternary neural networks with residual quantization," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, pp. 8538–8546, 2021.

[11] H. Jeong et al., "A ternary neural network computing-in-memory processor with 16t1c bitcell architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 5, pp. 1739–1743, 2023.

[12] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The era of 1-bit llms: All large language models are in 1.58 bits," *arXiv preprint arXiv:2402.17764*, 2024.

[13] 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, pp. 770–778.

[14] A. Krizhevsky, G. Hinton, et al., "Learning multiple layers of features from tiny images," 2009.

[15] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, pp. 8024–8035. 2019.