2022 5th International Conference on Artificial Intelligence for Industries (AI4I) | 978-1-6654-5961-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/AI4I54798.2022.00015

Channel Pruning in Quantization-aware Training: an Adaptive Projection-gradient Descent-shrinkage-splitting Method

Zhijian Li
Department of Mathematics
University of California, Irvine
Irvine, USA
zhijil2@uci.edu

Jack Xin
Department of Mathematics
University of California, Irvine)
Irvine, USA
jack.xin@uci.edu

***Abstract*—We propose an adaptive projection-gradient descent-shrinkage- splitting method (APGDSSM) to integrate penalty based channel pruning into quantization-aware training (QAT). APGDSSM concurrently searches weights in both the quantized subspace and the sparse subspace. APGDSSM uses shrinkage operator and a splitting technique to create sparse weights, as well as the Group Lasso penalty to push the weight sparsity into channel sparsity. In addition, we propose a novel complementary transformed l_1 penalty to stabilize the training for extreme compression.**

***Index Terms*—convolutional neural network, quantization, channel pruning, LASSO,**

I. Introduction

Convolutional neural networks (CNNs) have been widely used for computer vision tasks such as image classification and segmentation. To increase efficiency and reduce memory costs in mobile and IoT applications, network compression is necessary. Quantization and channel pruning are two commonly adopted methods. QAT searches the optimal weight in the quantized subspace. For a CNN with L convolutional layers, let $\mathbf{w} = \{w_1, \cdots, w_L\}$ be weight tensors structured in (height, width, channel) per layer. The subspace of m-bit ($m \geq 2$) quantization $\mathcal{Q} \in R^n$ is

$$\mathcal{Q} = \mathbb{R} \times \{0, \pm 1, \pm 2, \cdots, \pm 2^{m-1}\}^n$$

Given an objective function $\mathcal{L}$, the quantization problem is $\text{argmin}_{\mathbf{u} \in \mathcal{Q}} \mathcal{L}(\mathbf{u})$ for which [4] proposed a widely used QAT algorithm based on an auxiliary float weight w to perform QAT. With learning rate γ, it can be formulated as

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma \nabla \mathcal{L}(\mathbf{u}),\ \mathbf{u} \leftarrow \text{Proj}_Q(\mathbf{w}) \tag{1}$$

where the $\text{Proj}_Q(\cdot)$ is the projection that maps the float weight into the quantized subspace. For a theoretical convergence analysis of (1) and a relaxed formulation with improved performance , see [16]. Channel pruning is a structured compression well-studied by itself ([3], [5], [10], [13], [14] and references therein). Integrating QAT into adversarial training and studying the sparsity of quantized models are performed in [9]

This work is partly supported by NSF grants DMS-1854434, DMS-1952644, DMS-1924548.

The main contribution of our work here is to propose an integrated objective to *do channel pruning and weight quantization in one shot*. This is achieved by minimizing a new objective function with group sparse penalty over Q through an adaptive splitting, projection, gradient descent and proximal operations (APGDSSM algorithm). The adaptive step is to avoid weights in a layer all becoming very small, or fix potential model collapse when trained by the integrated steps of the algorithm. Besides adapting training schedule, we also found a new penalty, the so called complementary transformed-ℓ_1 (CTℓ_1), to steer weights away from the trivial state in each layer. Using CTℓ_1) gives more room to trade-off accuracy for efficiency than adapting training schedule. Experimental results on CIFAR-10, CIFAR-100, and Imagenet support our proposed methodology and framework.

II. Related Work

For a loss function l, the Lasso regularized problem is

$$\mathcal{L}(\mathbf{w}) = l(\mathbf{w}) + \lambda ||\mathbf{w}||_1. \tag{2}$$

It is well-known that Lasso regularization does parameter selection for the model, and several approaches exist for solving problem (2). In [1], an iterative algorithm of proximal operator (FISTA) solves (2), where the proximal operator for a penalty function g is defined as $\text{Prox}_g(\mathbf{w}) = \text{argmin}_{\mathbf{u}}\, g(\mathbf{u}) + \frac{1}{2}||\mathbf{u} - \mathbf{x}||^2$. The algorithm is:

$$\mathbf{w}^{t+1} = \text{Prox}_\lambda\big(\mathbf{w}^t - \gamma \nabla f(\mathbf{w}^t)\big)$$

where

$$\text{Prox}_\lambda(x) = \text{sgn}(x) \cdot \max(|x| - \lambda, 0).$$

An alternative method to solve (2) is the Alternating Direction Method of Multipliers (ADMM), through an augmented Lagrangian (Boyd et al. [2]):

$$\mathcal{L}(\mathbf{w}, \mathbf{u}, \mathbf{z}) = f(\mathbf{w}) + \lambda ||\mathbf{u}||_1 + \langle \mathbf{z}, \mathbf{w} - \mathbf{u} \rangle + \frac{\beta}{2}||\mathbf{w} - \mathbf{u}||^2 \tag{3}$$

978-1-6654-5961-7/22/$31.00 ©2022 IEEE
DOI 10.1109/AI4I54798.2022.00015

ADMM is adapted to neural network training in [12], [15]. The convergence theorems of ISTA and ADMM require both the loss function and penalty function to be convex, which does not apply to deep neural networks. The relaxed splitting variable method (RSVM, [6]) sparsifies non-convex neural networks by minimizing a simplified augmented Lagrangian:

$$l_{\lambda/\beta}(\mathbf{w}, \mathbf{u}) = f(\mathbf{w}) + \lambda||\mathbf{u}||_1 + \frac{\beta}{2}||\mathbf{u} - \mathbf{w}||^2.$$

RVSM updates weights as

$$\mathbf{w} \leftarrow \mathbf{w} - \gamma\nabla f(\mathbf{w}) - \gamma\beta(\mathbf{w} - \mathbf{u}),\ \mathbf{u} \leftarrow \text{Prox}_{\lambda/\beta}(\mathbf{w}) \quad (4)$$

which extends to non-differential penalties (e.g. ℓ_0) with the corresponding proximal operator. The RVSM does not require convex or differentiable penalty function for convergence [6], and it applies to adversarially trained networks [5]. Though models trained by RVSM usually have unstructured sparsity with limited channel sparsity, RVSM extends readily to a group-wise variable splitting method (RGSM, [14]) based on Group Lasso (GL) penalty:

$$||\mathbf{w}||_{GL} = \sum_{l=1}^{L}\sum_{i\in I_l}||w_{l,i}||_2$$

to increase channel sparsity, where I_l is the collection of channels in the l-th layer. GL penalty with its proximal operator in closed form is applied channel-wise in network training to realize sparse channels [5], [14]. In [11], RGSM and QAT are combined in a multi-stage process to achieve both channel pruning and binary weights.

III. Methodology and APGSSM Algorithm

To train quantized neural networks with sparse channels, we proposed an algorithm to concurrently search the optimal weights in the quantized subspace and the sparse subspace, as shown in Algorithm 1. The objective is

$$\min_{u\in\mathcal{Q}} \mathcal{L}(\mathbf{u}) := l(\mathbf{u}) + \lambda_2||\mathbf{u}||_{GL} + \lambda_1||\mathbf{u}||_1 \quad (5)$$

The procedure of training is shown in Algorithm 1. We note that the Lasso regularization term in equation (5) is imposed implicitly; the l_1 penalty does not contribute to the gradient. Instead, we use the shrinkage operator to minimize it. For parameters, we use symbols against the epoch number t, e.g. λ_1^t, to indicate that there is an adaptive scheme for the values.

This algorithm concurrently searches both the quantized subspace and the subspace of sparse weight (with small l_1 norm). We can either use only shrinkage operator (APGDSM) or use it together with the splitting (APGDSSM). The splitting term updates the gradient descent of $\frac{\beta}{2}||\mathbf{w}^t - \mathbf{u}^t||^2$, which makes the float weight $\mathbf{w}^t$ close to the quantized weight $\mathbf{u}^t$. Since $\mathbf{u}^t$ is much more sparse than $\mathbf{w}^t$, the splitting step renders $\mathbf{w}^t$ with more small elements, which strengthens the performance of the following shrinkage operator. However, pushing $\mathbf{w}^t$ close to $\mathbf{u}^t$ can jeopardize the performance, as it is not the descending direction guided by gradient.

Algorithm 1 APGDSM and APGDSSM

Input: Float weights w^0. Hyperparameters $\lambda_1, \lambda_2, \beta$.
Output : Quantized weights u.

for $t = 1, \cdots, 200$ **do**:
  $\mathbf{u}^t = Proj_Q(\mathbf{w}^t)$
  $f(\mathbf{u}^t) = l(\mathbf{u}^t) + \lambda_2^t||\mathbf{u}^t||_{GL}$
  $\mathbf{w}^t = \mathbf{w}^{t-1} - \alpha\nabla f(\mathbf{u}^t)$
  if Splitting **then**: ▷ Split if APGDSSM
    $\mathbf{w}^t = \mathbf{w}^t - \gamma^t\beta^t(\mathbf{w}^t - \mathbf{u}^t)$
  end if
  $\mathbf{w}^t = Prox_{\lambda_1^t}(\mathbf{w}_g^t)$
end for
$\mathbf{u} = Proj_Q(\mathbf{w}^{200})$

TABLE I
Adaptive scheme for the parameters in Algorithm 1. At epochs listed in the left-side column, we multiply the parameters by the fatcor in the right-side column

Epoch	Factor for λ_1&λ_2	Factor for β
35	0.5	0.5
70	0.2	0.2
110	0.5	0.1
150	0.5	0.1

IV. Implementation and Experiments

We use the standard adaptive scheme for the learning rate γ^t. The initial learning rate is 0.1, and we multiply the learning rate by a factor of 0.1 at epochs 80, 120, and 160. During the training, we need to change the scale of the regularization parameters to fit the current learning rate. For both λ_1, λ_2, and β, we empirically design a scheme to adapt the values of parameters. The reason we have a different adaptive scheme from the learning rate is that the training has a high probability to collapse if the parameters are re-scaled too late. As in Algorithm 1, all GL regularization, shrinkage operator, and splitting terms drive the weights to be sparse, which can lead the neural network to reach 100% channel sparsity at some point. When it happens, the training collapses as the cross-entropy loss becomes infinity.

V. Results

We validate Algorithm 1 in CIFAR10 and CIFAR100 with ResNet ([8]). The results are shown in Table II. As the table shows, the GL penalty and the shrinkage operator can significantly improve the weight sparsity and the channel sparsity with minor reduction on accuracy. The splitting step before the shrinkage operator can greatly improve the sparsity. Of course, the model performance would be somewhat affected.

Meanwhile, we numerically verify the convergence of the sparsity in Figure 2. Although the weight sparsity will decease every time the values of parameters updated, the channel sparsity has a nice convergence along training. The channel-wise GL penalty is the key to push the weight sparsity created by shrinkage and potential splitting into channel sparsity. In Figure 3, we show the comparison of a float ResNet56 and a 4-

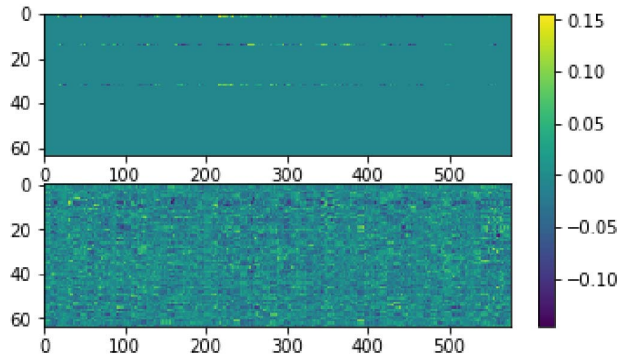

Fig. 1. Visualization the 45th layer of a float resnet56 (bottom) and a 4-bit resnet56 pruned by APGDSSM (top). The layer originally has shape [64,64,3,3] and is permuted and reshaped to shape 64×576 for visualization. Each row of the plots represents a channel.

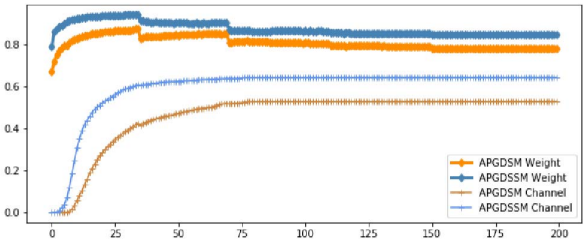

Fig. 2. Weight sparsity and channel sparsity against epochs. The weights sparsity deceases at milestones of the adaptive schemes, while the channel sparsity has smooth convergence

TABLE II
4-BIT QUANTIZED MODELS WITH PRUNING METHODS ON CIFAR-10 DATASET. THE INITIAL VALUES OF PARAMETERS ARE $\lambda_1^1 = 0.04, \lambda_2^1 = 5.e-6, \beta^1 = 1.e-3$. FOR CIFAR-10 AND $\lambda_1^1 = 0.02, \lambda_2^1 = 5e-6, \beta^1 = 1e-3$ FOR CIFAR-100

Model	Pruning	Ch. sp	Wt .sp	Accuracy
CIFAR-10				
Resnet20	None	9.53%	42.73%	91.75%
Resnet20	APGDSM	14.67%	72.68%	91.53%
Resnet20	APGDSSM	24.56%	85.04%	90.64%
Resnet56	None	25.16%	61.83%	93.24%
Resnet56	APGDSM	52.76%	78.11%	92.58%
Resnet56	APGDSSM	64.28%	84.59%	91.69%
CIFAR-100				
Resnet110	None	24.63%	53.20%	71.74%
Resnet110	APGDSM	33.61%	69.44%	71.68%
Resnet110	APGDSSM	36.62%	85.04%	71.59%

bit ResNet56 pruned via APGDSSM. We see that the channels are largely pruned in most layers.

VI. EFFICIENCY AND ACCURACY TRADE-OFF

As we mentioned in the previous sections, the training might collapse if the adaptive scheme and parameter values are selected carelessly. In order to further compress the models, we propose a complementary transformed-l_1 (CTL$_1$) penalty to prevent the training from collapse. This penalty is inspired by the transformed l_1 (TL$_1$) regularization in robust compressed sensing [17]. We define

$$||x||_{CTL_1,a} := 1 - \rho_a(x) = 1 - \frac{|x|}{a+|x|}$$

We remark that $||\cdot||_{CTL,a}$ is not a norm but only a regularization. We abuse the norm notation here for convenience. Note that

$$\lim_{a\to 0^+} ||x||_{CTL_1,a} = 1 - ||x||_0 = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}$$

For small choice of a, the value of $||x||_{CTL_1,a}$ is negligible when $|x|$ is large. The behavior of the CTL$_1$ penalty is illustrated in figure 4. To prevent the neural network from having a zero layer, we apply it to each layer of our model

$$||\mathbf{w}||_{CTL_1,a} := \sum_{l=1}^{L} 1 - \frac{||w_l||_1}{a+||w_l||_1}$$

By imposing this CTL$_1$ penalty, we force each layer to have some nonzero weights, so the training will not collapse. The augmented objective is

$$\min_{\mathbf{u}\in\mathcal{Q}} \mathcal{L}(\mathbf{u}) := f(\mathbf{u}) + \lambda_2||\mathbf{u}||_{GL} + \lambda_3||\mathbf{u}||_{CTL_1} + \lambda_1||\mathbf{u}||_1 \quad (6)$$

As a result, we can have more 'aggressive' choices for the values of parameters and the adaptive scheme to further pruning the neural networks.

Algorithm 2 APDSSM with CTl_1 penalty

Input: Float weights w^0. Hyperparameters $\lambda_1, \lambda_2, \beta$.
Output : Quantized weights u.

for $t = 1, \cdots, 200$ **do**:
$\mathbf{u}^t = Proj_Q(\mathbf{w}^t)$
$f(\mathbf{u}^t) = l(\mathbf{u}^t) + \gamma^t\lambda_2||\mathbf{u}^t||_{GL} + \lambda_3||\mathbf{u}^t||_{CTL_1,\gamma^t a}$
$\mathbf{w}^t = \mathbf{w}^{t-1} - \gamma^t\nabla f(\mathbf{u}^t)$
$\mathbf{w}^t = \mathbf{w}^t - \gamma^t\beta(\mathbf{w}^t - \mathbf{u}^t)$
$\mathbf{w}^t = Prox_{\gamma^t\lambda_1}(\mathbf{w}_g^{t-1})$
end for
$\mathbf{u} = Proj_Q(\mathbf{w}^{200})$

TABLE III
THE STRONGER PRUNING SCHEME STABILIZED BY CTl_1 PENALTY $\lambda_1 = 0.2, \beta = 0.01$; R.=RESNET.

Model	λ_2 initial	Ch. sp	Wt. sp	Accuracy
CIFAR-10				
R.56	$1.5 \cdot 10^{-3}$	73.67%	95.80%	90.27%
R.56	$5 \cdot 10^{-3}$	82.90%	96.70%	88.71%
CIFAR-100				
R.110	$5 \cdot 10^{-4}$	55.12%	80.07%	70.75%
R.110	$1 \cdot 10^{-3}$	58.06%	80.75%	70.16%

In Algorithm2, we let the parameters λ_1, and λ_2, λ_3 and β have the same adaptive scheme by multiply it by the learning rate. This scheme makes the parameters decrease slower. Hence, as shown in Table III, the channel sparsity increases significantly. The CTl_1 penalty allows us to further trader-off the performance to efficiency based on our needs.

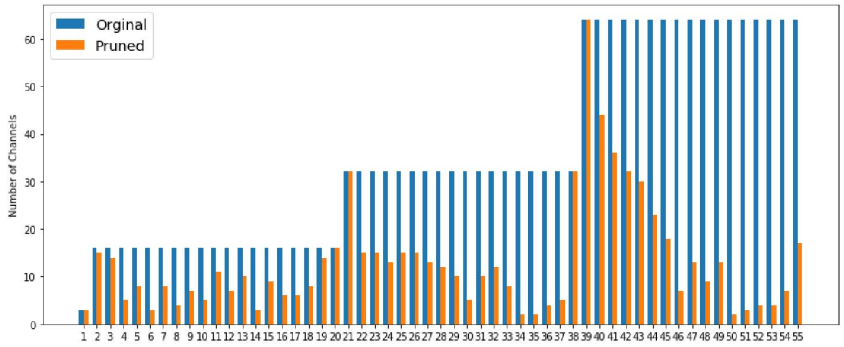

Fig. 3. The blue bars are the numbers of channels in layers of float ResNet56. The orange bars are the numbers of channels in layers of pruned 4-bit model by APGDSSM. The 55 of the 56 layers in ResNet56 are convolutional.

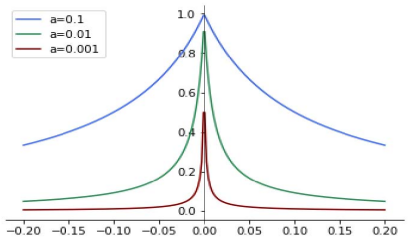

Fig. 4. CTl_1 penalty $1 - \frac{|x|}{a+|x|}$ for different values of a.

VII. Conclusion

In this paper, we proposed APGDSSM to integrate the penalty based channel pruning and QAT. We remark that relaxations of QAT ([7], [16]) will lead to sub-optimal outcomes, because such methods search the sparse subspace first and then find local optimal quantized weights around the searched sparse weights. The two subspaces need to be searched concurrently from the beginning. We verify that APGDSSM can deliver sparse quantized neural network with minor trader-off for performance. Further, we designed an auxiliary complementary transformed l_1 penalty to prevent training from collapsing, so we can trade more performance for efficiency if needed.

References

[1] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[2] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[3] Kevin Bui, Fredrick Park, Shuai Zhang, Yingyong Qi, and Jack Xin. Nonconvex regularization for network slimming: Compressing cnns even more. In *International Symposium on Visual Computing*, pages 39–53. Springer, 2020.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[5] Thu Dinh, Bao Wang, Andrea Bertozzi, Stanley Osher, and Jack Xin. Sparsity meets robustness: channel pruning for the Feynman-Kac formalism principled robust deep neural nets. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 362–381. Springer, 2020.

[6] Thu Dinh and Jack Xin. Convergence of a relaxed variable splitting method for learning sparse neural networks via ℓ_1, ℓ_0, and transformed-ℓ_1 penalties. In *Proceedings of SAI Intelligent Systems Conference*, pages 360–374. Springer, 2020.

[7] Tim Dockhorn, Yaoliang Yu, Eyyüb Sari, Mahdi Zolnouri, and Vahid Partovi Nia. Demystifying and generalizing binaryconnect. *Advances in Neural Information Processing Systems*, 34, 2021.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Zhijian Li, Bao Wang, and Jack Xin. An integrated approach to produce robust deep neural network models with high efficiency. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 451–465. Springer, 2021.

[10] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[11] Jiancheng Lyu and Spencer Sheen. A channel-pruned and weight-binarized convolutional neural network for keyword spotting. In *Le Thi H., Le H., Pham Dinh T., Nguyen N. (eds), Advanced Computational Methods for Knowledge Engineering. ICCSAMA 2019. Advances in Intelligent Systems and Computing*, volume 1121. Springer, Cham, 2020.

[12] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pages 2722–2731. PMLR, 2016.

[13] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[14] Biao Yang, Jiancheng Lyu, Shuai Zhang, Yingyong Qi, and Jack Xin. Channel pruning for deep neural networks via a relaxed groupwise splitting method. In *IEEE International Conference on Artificial Intelligence for Industries*, pages 97–98, 2019.

[15] Shaokai Ye, Kaidi Xu, Sijia Liu, Hao Cheng, Jan-Henrik Lambrechts, Huan Zhang, Aojun Zhou, Kaisheng Ma, Yanzhi Wang, and Xue Lin. Adversarial robustness vs. model compression, or both? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 111–120, 2019.

[16] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. BinaryRelax: A Relaxation Approach for Training Deep Neural Networks with Quantized Weights. *SIAM Journal on Imaging Sciences*, 11(4):2205–2223, 2018.

[17] Shuai Zhang and Jack Xin. Minimization of transformed ℓ_1 penalty: Closed form representation and iterative thresholding algorithms. *Comm. Math Sci.*, 15(2):511–537, 2017.