ADAM-ADMM: A Unified, Systematic Framework of Structured Weight Pruning for DNNs

Tianyun Zhang[1*], **Kaiqi Zhang**[1*], **Shaokai Ye**[1*], **Jiayu Li**[1], **Jian Tang**[1],
Wujie Wen[2], **Xue Lin**[3], **Makan Fardad**[1] **& Yanzhi Wang**[3]
1. Syracuse University 2. Florida International University
3. Northeastern University *Equal Contribution
1. {tzhan120,kzhang17,sye106,jli221,jtang02,makan}@syr.edu
2. wwen@fiu.edu 3. {xue.lin,yanz.wang}@northeastern.edu

Abstract

Weight pruning methods of deep neural networks (DNNs) have been demonstrated to achieve a good model pruning ratio without loss of accuracy, thereby alleviating the significant computation/storage requirements of large-scale DNNs. Structured weight pruning methods have been proposed to overcome the limitation of irregular network structure and demonstrated actual GPU acceleration. However, the pruning ratio (degree of sparsity) and GPU acceleration are limited (to less than 50%) when accuracy needs to be maintained.

In this work, we overcome pruning ratio and GPU acceleration limitations by proposing a unified, systematic framework of structured weight pruning for DNNs, named ADAM-ADMM (Adaptive Moment Estimation-Alternating Direction Method of Multipliers). It is a framework that can be used to induce different types of structured sparsity, such as filter-wise, channel-wise, and shape-wise sparsity, as well non-structured sparsity. The proposed framework incorporates stochastic gradient descent with ADMM, and can be understood as a dynamic regularization method in which the regularization target is analytically updated in each iteration. A significant improvement in weight pruning ratio is achieved without loss of accuracy, along with fast convergence rate. With a small sparsity degree of 33% on the convolutional layers, we achieve 1.64% accuracy enhancement for the AlexNet (CaffeNet) model. This is obtained by mitigation of overfitting. Without loss of accuracy on the AlexNet model, we achieve 2.6× and 3.65× average measured speedup on two GPUs, clearly outperforming the prior work. The average speedups reach 2.77× and 7.5× when allowing a moderate accuracy loss of 2%. In this case the model compression for convolutional layers is 13.2×, corresponding to 10.5× CPU speedup. Our models and codes are released at https://github.com/KaiqiZhang/ADAM-ADMM.

1 Introduction

Deep neural networks (DNNs) utilize multiple functional layers cascaded together to extract features at multiple levels of abstraction [1, 2], and are thus both computationally and storage intensive. As a result, many studies on DNN model compression are underway, including weight pruning [3, 4, 5, 6, 7], low-rank approximation [8, 9, 10], low displacement rank approximation (structured matrices) [11, 12, 13, 14], etc. Weight pruning can achieve a high model pruning ratio without loss of accuracy. A pioneering work [3, 4] adopts an iterative weight pruning heuristic and results in a sparse neural network structure. It can achieve 9× weight reduction with no accuracy loss on AlexNet [1]. This weight pruning method has been extended by the same group [15] and other researchers

Preprint. Work in progress.

[5, 7, 16, 17] to either use more sophisticated algorithms to achieve a higher weight pruning ratio, or to obtain a fine-grained trade-off between a higher pruning ratio and a lower accuracy degradation.

Despite the promising results, these general weight pruning methods often produce non-structured and irregular connectivity in DNNs. This leads to degradation in the degree of parallelism and actual performance in GPU and hardware platforms. Moreover, the weight pruning ratio is mainly achieved through compressing the fully-connected (FC) layers [3, 4, 16], which are less computationally intensive compared with convolutional (CONV) layers and are becoming less important in state-of-the-art DNNs such as ResNet [18]. To address these limitations, recent work [6, 19] have proposed to learn structured sparsity, including sparsity at the levels of filters, channels, filter shapes, layer depth, etc. These works focus on CONV layers and actual GPU speedup is reported as a result of structured sparsity [6]. However, these structured weight pruning methods are based on regularization techniques and are still quite heuristic [6, 19]. The weight pruning ratio and GPU acceleration are both quite limited. For example, the average weight pruning ratio on CONV layers of AlexNet is only 1.5× without any accuracy loss, corresponding to 33% sparsity.

In this work, we overcome this limitation by proposing *a unified, systematic framework of structured weight pruning for DNNs*, named ADAM-ADMM. It is a unified framework for different types of structured sparsity such as filter-wise, channel-wise, and shape-wise sparsity, as well as non-structured sparsity. It achieves a significant improvement in weight pruning ratio under the same accuracy, along with fast convergence rate in sparsity training. The ADAM-ADMM framework effectively incorporates stochastic gradient descent with the Alternating Direction Method of Multipliers (ADMM), a powerful technique in optimization theory that has been shown to perform well for non-convex optimization problems with combinatorial constraints [20, 21]. Through ADMM the structured pruning problem is decomposed into two subproblems, where one is effectively solved using stochastic gradient descent such as the Adam algorithm, while the other is solved analytically for different types of structures. In the context of deep learning, the ADAM-ADMM framework can be understood as a smart and dynamic regularization technique in which the regularization target is analytically updated in each iteration.

We conduct extensive experiments using ImageNet data set on two GPUs (NVIDIA 1080Ti and Jetson TX2), a CPU (Intel i7-6700K), and a Rasperry PI 3 embedded processor. With a small 33% degree of sparsity on CONV layers, we achieve 1.64% enhancement in accuracy for the AlexNet (CaffeNet) model. Without accurate loss on the AlexNet model, we achieve 2.6× and 3.65× average measured speedup on two GPUs, which clearly outperforms the GPU acceleration of 49% reported in SSL [6]. The speedups reach 2.77× and 7.5× when allowing a moderate accuracy loss of 2%. In this case the model compression for CONV layers is 13.2×, corresponding to 10.5× CPU speedup. Our ADAM-ADMM framework also achieves the highest pruning ratio in non-structured weight pruning. Without accuracy loss, we achieve 16.1× pruning in CONV layers of AlexNet (CaffeNet), which increases to 40.5× (conv2-conv5) when a moderate accuracy loss of 2% is allowed. Our models and codes are released at `https://github.com/KaiqiZhang/ADAM-ADMM`.

2 Related work

General, non-structured weight pruning. The pioneering work by Han *et al.* [3, 4] achieved 9× parameter reduction in AlexNet and 13× in VGG-16. However, most reduction is achieved in FC layers, and the 2.7× reduction achieved in CONV layers will not lead to an overall acceleration in GPU speed (indeed, it will cause a degradation in GPU speed if one exploits sparse matrix multiplication [6].) Extensions of the iterative weight pruning framework of Han *et al.* [3, 4], such as [16] (dynamic network surgery), [5] (NeST) and [15], use more delicate algorithms such as selective weight growing and pruning. But the weight pruning ratios on CONV layers are still limited, e.g., 3.1× in [16], 3.23× in [5], and 4.16× in [15] for AlexNet with no accuracy degradation. This level of non-structured weight pruning cannot guarantee GPU acceleration. In fact, our ADAM-ADMM framework can achieve 16.1× non-structured weight pruning in CONV layers of AlexNet without accuracy degradation, however, still only minor GPU acceleration is actually observed.

Structured weight pruning. To overcome the limitation in non-structured, irregular weight pruning, recent work on SSL [6] proposes to learn structured sparsity at the levels of filters, channels, filter shapes, layer depth, etc. This work is one of the first with actually measured GPU accelerations. This is because CONV layers after structured pruning will transform to a full matrix multiplication in

GPU (with reduced matrix size). However, the weight pruning ratio and GPU acceleration are both limited. The average weight pruning ratio on CONV layers of AlexNet is only 1.5× without accuracy loss. The reported GPU acceleration is 49%, which will further reduce with the advance in GPU technology (and higher degree of available parallelism). Besides, the recent work [19] achieves 2× channel pruning with 1% accuracy degradation on VGGNet.

Other types of DNN model compression techniques. There are many other types of DNN model compression techniques. Examples include low-rank approximation using single-value decomposition (SVD) [8, 9, 10], and low-displacement rank approximation using structured matrices such as circulant matrices [11, 14], Toeplitz matrices [12, 13], etc. These methods originally target FC layers only, but recent work [14, 10] has generalized to CONV layers as well. These techniques result in a regular network structure, but in general a lower pruning ratio and larger accuracy degradation compared with parameter pruning. We point out that these compression techniques are compatible with ADMM and will be the topic of future investigations orthogonal to this work.

3 The Unified ADAM-ADMM Framework for Structured Pruning

3.1 Problem formulation of structured pruning

Consider an N-layer DNN in which the first M layers are CONV layers and the rest are FC layers. The weights and biases of the i-th layer are respectively denoted by $\mathbf{W}_i$ and $\mathbf{b}_i$, and the loss function associated with the DNN is denoted by $f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big)$; see [22]. In this paper, $\{\mathbf{W}_i\}_{i=1}^N$ and $\{\mathbf{b}_i\}_{i=1}^N$ respectively characterize the collection of weights and biases from layer 1 to layer N.

The process of training a DNN involves using stochastic gradient descent, mostly the Adam algorithm [23], to minimize the loss function. In this paper, our objective is to implement structured pruning on the DNN. In the following discussion we focus on the CONV layers because they have the highest computation requirements, but the proposed framework is applicable to the FC layers as well. More specifically, we minimize the loss function subject to specific structured sparsity constraints on the weights in the CONV layers, i.e.,

$$\begin{aligned} \underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad & f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big), \\ \text{subject to} \quad & \mathbf{W}_i \in \mathbf{S}_i,\ i = 1, \ldots, M, \end{aligned} \tag{1}$$

where $\mathbf{S}_i$ is the set of $\mathbf{W}_i$ with a specific "structure". Our aim in this procedure is to strike a desirable balance between weight pruning ratio and parallel computation. The details of different types of structures will be discussed later in Section 4.

3.2 The proposed ADAM-ADMM framework

In problem (1) the constraint is non-convex and combinatorial. As a result, this problem cannot be solved directly by stochastic gradient descent methods such as the Adam algorithm. However, the property that $\mathbf{W}_i$ satisfies certain combinatorial "structures" allows us to integrate the ADMM framework with stochastic gradient descent to effectively solve this problem.

To apply the ADMM framework, we (i) define indicator functions to incorporate combinatorial constraints into the objective function, and (ii) define auxilliary variables that allow us to decompose the optimization problem into two subproblems that individually can be solved effectively. In what follows, we elaborate on these steps.

Corresponding to every set $\mathbf{S}_i$, $i = 1, \ldots, M$ we define the indicator function

$$g_i(\mathbf{W}_i) = \begin{cases} 0 & \text{if } \mathbf{W}_i \in \mathbf{S}_i, \\ +\infty & \text{otherwise.} \end{cases}$$

Furthermore, we incorporate auxilliary variables $\mathbf{Z}_i$, $i = 1, \ldots, M$ with the restriction that $\mathbf{Z}_i = \mathbf{W}_i$. The original problem (1) is then equivalent to

$$\begin{aligned} \underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad & f\big(\{\mathbf{W}_i\}_{i=1}^N, \{\mathbf{b}_i\}_{i=1}^N\big) + \sum_{i=1}^M g_i(\mathbf{Z}_i), \\ \text{subject to} \quad & \mathbf{W}_i = \mathbf{Z}_i,\ i = 1, \ldots, M. \end{aligned} \tag{2}$$

Through formation of the augmented Lagrangian [20], the ADMM framework decomposes problem (2) into two subproblems, and solves them iteratively until convergence[1]. The first subproblem is

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\}_{i=1}^N,\{\mathbf{b}_i\}_{i=1}^N\big)+\sum_{i=1}^{M}\frac{\rho_i}{2}\|\mathbf{W}_i-\mathbf{Z}_i^k+\mathbf{U}_i^k\|_F^2, \tag{3}$$

where $\mathbf{U}_i^k := \mathbf{U}_i^{k-1}+\mathbf{W}_i^k-\mathbf{Z}_i^k$. The first term in the objective function of (3) is the differentiable loss function of the DNN, and the second term is a quadratic regularization term of the $\mathbf{W}_i$'s, which is differentiable and convex. As a result (3) can be solved by stochastic gradient descent. Although we cannot guarantee the global optimality of the solution, it is due to the non-convexity of the DNN loss function rather than the quadratic term enrolled by our method.

On the other hand, the second subproblem is given by

$$\underset{\{\mathbf{Z}_i\}}{\text{minimize}} \quad \sum_{i=1}^{M} g_i(\mathbf{Z}_i)+\sum_{i=1}^{M}\frac{\rho_i}{2}\|\mathbf{W}_i^{k+1}-\mathbf{Z}_i+\mathbf{U}_i^k\|_F^2. \tag{4}$$

Note that $g_i(\cdot)$ is the indicator function of $\mathbf{S}_i$, thus this subproblem can be solved analytically and optimally [20]. For $i=1,\ldots,M$, the optimal solution is

$$\mathbf{Z}_i^{k+1}=\mathbf{\Pi}_{\mathbf{S}_i}(\mathbf{W}_i^{k+1}+\mathbf{U}_i^k), \tag{5}$$

where $\mathbf{\Pi}_{\mathbf{S}_i}(\cdot)$ is Euclidean projection of $\mathbf{W}_i^{k+1}+\mathbf{U}_i^k$ onto $\mathbf{S}_i$. The set $\mathbf{S}_i$ is different when we apply different types of structured sparsity. We will discuss how to implement the Euclidean projection to different types of structures in Section 4.

3.3 Explanation of ADAM-ADMM in the deep learning context

We name the proposed framework ADAM-ADMM because it is different from the conventional utilization of ADMM, i.e., to accelerate the convergence of an originally convex problem [20, 21].Rather, we propose to integrate the ADMM framework with stochastic gradient descent, and employ ADAM as a particular implementation of stochastic gradient descent. [2] Our experimental results demonstrate the significantly higher performance of the proposed framework in weight pruning and GPU acceleration.

Aside from mathematical proof in optimization theory, the advantage of the proposed ADAM-ADMM framework can be explained in the deep learning context. Problem (3) can be understood as a smart, dynamic L_2 regularization method, in which the regularization target $\mathbf{Z}_i^k-\mathbf{U}_i^k$ will change judiciously and analytically in each iteration. It will achieve higher performance in weight pruning or other objective functions compared with the traditional regularization method in which the regularization term is fixed.

4 Discussion and solutions of different types of structured sparsity

Problem (1) employs the constraint set $\mathbf{S}_i$ to represent the structure in the i-th CONV layer of the DNN. In this section, we introduce constraint sets corresponding to different types of structured sparsity, and subsequently, the solution to corresponding structured weight pruning problems in the ADAM-ADMM framework. Non-structured, irregular sparsity constraints (i.e., pruning) are also included in the framework. The suitability for GPU acceleration is discussed for different types of sparsity, and we finally introduce the proper combination of structured sparsities to facilitate GPU accelerations.

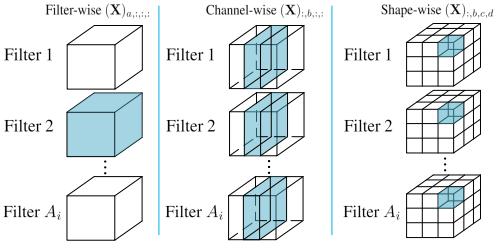

Figure 1: Illustration of filter-wise, channel-wise and shape-wise structured sparsities from left to right.

[1] The details of ADMM are presented in [20, 22]. We omit the details due to space limitation.

[2] Our proposed framework can also employ other types of stochastic gradient descent. For example, the training of CaffeNet [24] uses the momentum method to implement stochastic gradient descent, and we demonstrate in this paper that this mehtod can be used in conjunction with ADMM.

The collection of weights in the i-th CONV layer is a four-dimensional tensor, i.e., $\mathbf{W}_i \in R^{A_i \times B_i \times C_i \times D_i}$, where A_i, B_i, C_i, and D_i are respectively the number of filters, the number of channels in a filter, the height of the filter, and the width of the filter, in layer i. In what follows, if $\mathbf{X}$ denotes the weight tensor in a specific layer, let $(\mathbf{X})_{a,:,:,:}$ denote the a-th filter in $\mathbf{X}$, $(\mathbf{X})_{:,b,:,:}$ denote the b-th channel, and $(\mathbf{X})_{:,b,c,d}$ denote the collection of weights located at position $(:, b, c, d)$ in every filter of $\mathbf{X}$, as illustrated in Figure 1.

4.1 Filter-wise structured sparsity

When we train a DNN with sparsity at the filter level, the constraint on the weights in the i-th CONV layer is given by

$$\mathbf{W}_i \in \mathbf{S}_i := \{\mathbf{X} \mid \text{the number of nonzero filters in } \mathbf{X} \text{ is less than or equal to } \alpha_i\}.$$

Here, nonzero filter means that the filter contains some nonzero weight. To solve subproblem (5) with such constraints, we first calculate $O_a = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{a,:,:,:}\|_F^2$ for $a = 1, \ldots, A_i$, where $\|\cdot\|_F$ denotes the Frobenius norm. We then keep α_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{a,:,:,:}$ corresponding to the α_i largest values in $\{O_a\}_{a=1}^{A_i}$ and set the rest to zero.

4.2 Channel-wise structured sparsity

When we train a DNN with sparsity at the channel level, the constraint on the weights in the i-th CONV layer is given by

$$\mathbf{W}_i \in \mathbf{S}_i := \{\mathbf{X} \mid \text{the number of nonzero channels in } \mathbf{X} \text{ is less than or equal to } \beta_i\}.$$

Here, we call the b-th channel nonzero if $(\mathbf{X})_{:,b,:,:}$ contains some nonzero element. To solve subproblem (5) with such constraints, we first calculate $O_b = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,:,:}\|_F^2$ for $b = 1, \ldots, B_i$. We then keep β_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,:,:}$ corresponding to the β_i largest values in $\{O_b\}_{b=1}^{B_i}$ and set the rest to zero.

4.3 Shape-wise structured sparsity

When we train a DNN with sparsity at the filter shape level, the constraint on the weights in the i-th CONV layer is given by

$$\mathbf{W}_i \in \mathbf{S}_i := \left\{\mathbf{X} \mid \text{the number of nonzero vectors in } \{\mathbf{X}_{:,b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i} \text{ is less than or equal to } \theta_i\right\}.$$

To solve subproblem (5) with such constraints, we first calculate $O_{b,c,d} = \|(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,c,d}\|_F^2$ for $b = 1, \ldots, B_i$, $c = 1, \ldots, C_i$ and $d = 1, \ldots, D_i$. We then keep θ_i elements in $(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k)_{:,b,c,d}$ corresponding to the θ_i largest values in $\{O_{b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i}$ and set the rest to zero.

4.4 Non-Structured Weight sparsity

When we train a DNN with non-structured weight sparsity, the constraint on the weights in the i-th CONV layer is

$$\mathbf{W}_i \in \mathbf{S}_i := \{\mathbf{X} \mid \text{the number of nonzero elements in } \mathbf{X} \text{ is less than or equal to } \gamma_i\}.$$

To solve subproblem (5), we keep γ_i elements in $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ with largest magnitudes and set the rest to zero [22].

4.5 Summary and combination of structured sparsities to facilitate GPU acceleration

Convolutional computations in DNNs are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors to matrices [25], named *general matrix multiplication* or GEMM. It is stated in [10] that the combination of filter-wise and filter shape-wise sparsities can be employed to directly reduce the dimension of a weight matrix by removing its zero rows and columns, thereby enabling GPU acceleration. This is illustrated in Figure 2. In the results we will use *row pruning* and *column pruning* to represent the results of filter-wise and shape-wise sparsities.

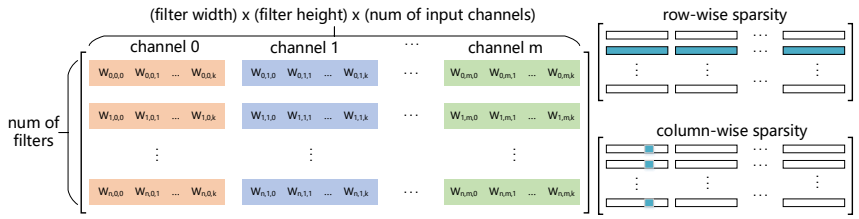

Figure 2: Illustration of 2D weight matrix for GEMM (left) and row-wise and column-wise sparsity (right)

Besides these two types of sparsities, channel sparsity allows for the pruning of entire channels, and is suitable for emerging (systolic array-based) hardware implementations of DNNs that are channel-based, including Google TPU [26]. Finally, it is worth mentioning that non-structured, general sparsity rarely leads to GPU acceleration due to the irregular structure after pruning, but is suitable for mobile and hardware implementations.

5 Experiment Results and Discussions

In this section, we evaluate the proposed ADAM-ADMM framework on the ImageNet ILSVRC-2012 data set. Evaluation results on the MNIST data set are shown in Supplementary Materials. Our experiments are based on the CaffeNet model (`https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet`) for a fair comparison with prior work SSL [6] on structured pruning. CaffeNet is a replication of AlexNet [1] with minor changes, yielding a Top-1 accuracy of 57.4%. We initialize ADMM by using the pre-trained CaffeNet model, and we set the penalty parameters $\rho_1 = \cdots = \rho_N = 3 \times 10^{-3}$ for filter-wise and shape-wise structured sparsity, and $\rho_1 = \cdots = \rho_N = 1.5 \times 10^{-3}$ for non-structured sparsity.

The ADAM-ADMM framework provides a fair comparison among different types of structured sparsity and also non-structured sparsity. In the experiments we aim to answer two questions. First, what will be the performance enhancement of the proposed ADAM-ADMM framework compared with the prior work on weight pruning? Second, what are the pros and cons for different types of structured sparsity and also non-structured sparsity? To answer the second question, we have conducted extensive speedup testings on two GPU nodes, the high-performance NVIDIA GeForce GTX 1080Ti and the low-power NVIDIA Jetson TX2 (targeting embedded applications), as well as the Intel I7-6700K Quad-Core CPU and the Rasperry PI 3 embedded processor.

The training of sparse DNN models is performed in Caffe using NVIDIA 1080Ti and Tesla P100 GPUs. The comparisons on GEMM computation efficiency and acceleration are conducted in Caffe [24]. The batch size is 1, which is typical for inference [3, 6, 10]. The baseline models and structured sparse models execute using cuBLAS on GPU and Intel Math Kernel Library (MKL) on CPU. The non-structured sparse models use cuSPARSE library on GPU.

5.1 Guidelines for Accuracy Enhancement: Refining and Balanced Pruning

We discuss two guidelines for accuracy retrieving and enhancement in the ADAM-ADMM framework. The first is *final refining/retraining*. For a higher sparsity training speed, we may not wait for the full convergence of ADMM. Rather, we remove the group of weights which are 0 or close to 0 after 9-12 ADMM iterations, and then we retrain the remaining non-zero weights to retrieve accuracy. We can achieve an effective tradeoff between training speed and accuracy in this way.

We name the second guideline *balanced pruning*, where we simultaneously prune the CONV and FC layers (rather than CONV layers only) in order to achieve a higher accuracy. We hypothesize that overfitting can be effectively avoided through the balanced pruning of both CONV and FC layers. In this way the overall accuracy can be improved by 0.3% to 0.4% for the ImageNet data set.

5.2 Comparison Results on Structured Sparsity

We compare our method with the two configurations of the SSL method [6] on AlexNet/CaffeNet. The first has no accuracy degradation (Top-1 error 42.53%) and average sparsity of 33% on conv2-conv5.

Table 1: Comparison of our method and SSL method on column sparsity **without accuracy loss** on CaffeNet model using ImageNet ILSVRC-2012 data set

Method	Top1 error	Statistcs	conv1	conv2	conv3	conv4	conv5	conv2-5[§]
SSL [6]	42.53%	column sparsity	0.0%	20.9%	39.7%	39.7%	24.6%	1.5×
our method	40.96%	column sparsity	0.0%	20.9%	39.7%	39.7%	24.6%	1.5×
our method	42.53%	column sparsity	0.0%	70.0%	77.0%	85.0%	81.0%	
		GPU1×	1.00	2.27	3.35	3.64	1.04	4.8×
		GPU2×	1.00	2.83	3.92	4.63	3.22	

[§] pruning ratio

We note that the 1st CONV layer of AlexNet/CaffeNet is very small with only 35K weights compared with 2.3M in conv2-conv5, and is often not the optimization focus [6, 10]. The second has around 2% accuracy degradation (Top-1 error 44.66%) with average sparsity of 81.7% on conv2-conv5.

Table 1 shows the comparison of our method with the first configuration of SSL. We provide two configurations from ADAM-ADMM, the first with the same sparsity in each layer as SSL, and the second with the same accuracy as SSL (no accuracy degradation compared with original model). With the same sparsity degree, the overall accuracy reaches 59.04%, which is 1.64% higher than the original Top-1 accuracy. This improvement is because overfitting can be mitigated through ADMM and outperforms the improvement in [27]. With the same accuracy, we can achieve a much higher degree of sparsity of 79% on conv2-conv5. This corresponds to 4.8× pruning, which is significantly higher compared with 1.5× pruning in conv2-conv5 in [6].

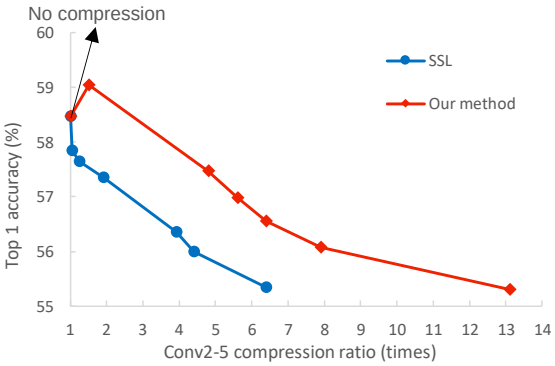

Figure 3: Top-1 accuracy vs. conv2-5 pruning ratio for structured pruning methods (our method and SSL)

We test the actual GPU accelerations using two GPUs: GPU1 is NVIDIA 1080Ti and GPU2 is NVIDIA TX2. The acceleration ratio is computed with respect to the corresponding layer of the original DNN executing on the same GPU and same setup. One can observe that the average acceleration of conv2-conv5 on 1080Ti is 2.6×, while the average acceleration on TX2 is 3.65×. These results clearly outperform the GPU acceleration of 49% reported in SSL [6] without accuracy loss, as well as the recent work [10]. The acceleration ratio on TX2 is higher than 1080Ti because the latter has a high parallelism degree, which will not be fully utilized when the matrix size GEMM of a CONV layer is significantly reduced.

Table 2: Comparison of our method and SSL method on column and row sparsity **with acceptable accuracy loss** on CaffeNet model using ImageNet ILSVRC-2012 dataset

Method	Top1 error	Statistcs	conv1	conv2	conv3	conv4	conv5	conv2-5[§]
SSL [6]	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%	6.4×
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
our method	43.35%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%	
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		CPU×	1.05	2.76	6.28	8.64	3.92	6.4×
		GPU1×	1.00	1.25	4.10	1.49	1.19	
		GPU2×	1.00	2.29	6.52	5.94	3.25	
our method	44.67%	column sparsity	0.0%	87.1%	90.0%	90.0%	88.1%	
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		CPU×	1.05	7.75	14.68	13.55	6.02	13.2×
		GPU1×	1.00	2.32	5.34	1.82	1.59	
		GPU2×	1.00	4.77	12.55	7.97	4.70	

[§] pruning ratio

Table 3: Comparison of our method and other methods on non-structured sparsity **without accuracy loss** on AlexNet/CaffeNet model using ImageNet ILSVRC-2012

Method	Top1 err.	Statistcs	conv1	conv2	conv3	conv4	conv5	conv1-5[§]	conv2-5[§]
weight prun. [3]	42.77%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%	2.7×	2.7×
dyn. surgery [16]	43.09%	sparsity	46.2%	59.4%	71.0%	67.7%	67.5%	3.1×	3.1×
NeST [5]	42.76%	sparsity	51.1%	65.2%	81.5%	61.9%	59.3%	3.2×	3.3×
method [15]	N/A	sparsity	17.0%	74.0%	77.0%	77.0%	77.0%	4.2×	4.3×
l_1 [6]	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%	5.0×	5.9×
our method	41.33%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%	5.0×	5.9×
our method	42.50%	sparsity	0.0%	93.0%	94.3%	93.6%	93.5%	13.1×	16.1×

[§] pruning ratio

Table 4: Comparison of our method and l_1 regularization method on non-structured sparsity **with acceptable accuracy loss** on CaffeNet model using ImageNet ILSVRC-2012 dataset

Method	Top1 error	Statistcs	conv1	conv2	conv3	conv4	conv5	conv1-5[§]	conv2-5[§]
l_1 [6]	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%	21.8×	24.0×
our method	43.36%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%	21.8×	24.0×
our method	44.62%	sparsity	0.0%	97.0%	98.0%	97.5%	97.0%	25.5×	40.5×

[§] pruning ratio

Table 2 shows the comparison with the second configuration of SSL. Again we provide two configurations from ADAM-ADMM. With the same sparsity in each layer as SSL, we can again achieve a higher accuracy. With the same accuracy (a moderate accuracy loss within 2% compared with original DNN), a higher degree of 92.4% average sparsity of conv2-conv5 is achieved, translating into 13.2× weight pruning. The actual GPU acceleration results are also high: 2.77× on 1080Ti and 7.5× on TX2. One can clearly see that the speedup on 1080Ti saturates because the high parallelism degree cannot be fully exploited. And this trend will be more significant in more powerful GPUs. The acceleration on CPU can be higher under this setup, reaching 10.5× on average on conv2-conv5.

We can clearly observe that the overall accuracy increases with a moderate pruning ratio in the ADAM-ADMM framework. This is because ADAM-ADMM can effectively mitigate overfitting. The overall accuracy vs. pruning ratio is demonstrated in Figure 3 in comparison with SSL results, clearly demonstrating the advantage of the ADAM-ADMM framework.

5.3 Non-Structured Sparsity and Comparison Results on Different Platforms

The ADAM-ADMM framework is applicable to non-structured weight pruning as well. In this section we demonstrate the non-structured weight pruning results (CONV layer pruning) achieved by the ADAM-ADMM framework, as well as comparisons with reference work on non-structured pruning [3, 5, 15, 16]. Without any accuracy loss compared with the original CaffeNet/AlexNet model, we achieve 16.1× weight pruning in conv2-conv5, which are 6.0×, 5.2×, 4.9×, 3.75×, and 2.73× compared with references [3], [16], [5], [15], and the l_1 regularization method in [6], respectively. In fact, our approach additionally achieves the highest accuracy among these methods. Within 2% accuracy degradation, we achieve 40.5× weight pruning in conv2-conv5.

However, even such high degree of sparsity cannot lead to a good GPU acceleration. With 40.5× weight pruning ratio in conv2-conv5, we can only achieve less than 2× speedup on NVIDIA TX2 and even speed reduction on 1080Ti (where the speed is calculated when both GPUs exploit sparse matrix multiplication). This coincides with the conclusion in [6] that the high parallelism degree in GPUs cannot be fully exploited in irregular sparsity patterns even if the state-of-the-art sparse matrix multiplication package is utilized. However, such significant pruning results are clearly beneficial for embedded processors, FPGA and ASIC designs due to the significantly shrunk model size and computation requirement. For example, over 20× speedup can be observed when executing on Rasperry PI 3 embedded processor thanks to the reduced computation and memory footprint.

6 Conclusion

In this paper we proposed a unified, systematic framework of structured weight pruning for DNNs. It is a unified framework for different types of structured sparsity such as filter-wise, channel-wise, shape-wise sparsity as well for non-structured sparsity. In our experiments, we achieve 2.6× and 3.65× measured speedup on two GPUs without accuracy loss. The speedups reach 2.77× and 7.5× on GPUs and 10.5× on CPU when allowing a moderate accuracy loss of 2%. Our pruning ratio and speedup clearly outperform prior work.

References

[1] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

[2] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[3] Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems (NIPS). (2015) 1135–1143

[4] Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. International Conference on Learning Representations (ICLR) (2016)

[5] Dai, X., Yin, H., Jha, N.K.: Nest: A neural network synthesis tool based on a grow-and-prune paradigm. arXiv preprint arXiv:1711.02017 (2017)

[6] Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems. (2016) 2074–2082

[7] Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. arXiv preprint arXiv:1611.05128 (2016)

[8] Denil, M., Shakibi, B., Dinh, L., De Freitas, N., et al.: Predicting parameters in deep learning. In: Advances in neural information processing systems. (2013) 2148–2156

[9] Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems. (2014) 1269–1277

[10] Wen, W., Xu, C., Wu, C., Wang, Y., Chen, Y., Li, H.: Coordinating filters for faster deep neural networks. CoRR, abs/1703.09746 (2017)

[11] Cheng, Y., Yu, F.X., Feris, R.S., Kumar, S., Choudhary, A., Chang, S.F.: An exploration of parameter redundancy in deep networks with circulant projections. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 2857–2865

[12] Sindhwani, V., Sainath, T., Kumar, S.: Structured transforms for small-footprint deep learning. In: Advances in Neural Information Processing Systems. (2015) 3088–3096

[13] Zhao, L., Liao, S., Wang, Y., Li, Z., Tang, J., Pan, V., Yuan, B.: Theoretical properties for neural networks with weight matrices of low displacement rank. arXiv preprint arXiv:1703.00144 (2017)

[14] Ding, C., Liao, S., Wang, Y., Li, Z., Liu, N., Zhuo, Y., Wang, C., Qian, X., Bai, Y., Yuan, G., et al.: C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In: Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, ACM (2017) 395–408

[15] Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., Dally, W.J.: Exploring the regularity of sparse structure in convolutional neural networks. arXiv preprint arXiv:1705.08922 (2017)

[16] Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient dnns. In: Advances In Neural Information Processing Systems. (2016) 1379–1387

[17] Dong, X., Chen, S., Pan, S.: Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: Advances in Neural Information Processing Systems. (2017) 4860–4874

[18] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2016) 770–778

[19] He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: International Conference on Computer Vision (ICCV). Volume 2. (2017) 6

[20] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning **3**(1) (2011) 1–122

[21] Hong, M., Luo, Z.Q.: On the linear convergence of the alternating direction method of multipliers. Mathematical Programming **162**(1-2) (2017) 165–199

[22] Zhang, T., Ye, S., Zhang, K., Tang, J., Wen, W., Fardad, M., Wang, Y.: A systematic dnn weight pruning framework using alternating direction method of multipliers. arXiv preprint arXiv:1804.03294 (2018)

[23] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[24] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia, ACM (2014) 675–678

[25] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759 (2014)

[26] Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al.: In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th Annual International Symposium on Computer Architecture, ACM (2017) 1–12

[27] Han, S., Pool, J., Narang, S., Mao, H., Tang, S., Elsen, E., Catanzaro, B., Tran, J., Dally, W.J.: Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. arXiv preprint arXiv:1607.04381 (2016)