A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers

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

Abstract. Weight pruning methods for deep neural networks (DNNs) have been investigated recently, but prior work in this area is mainly heuristic, iterative pruning, thereby lacking guarantees on the weight reduction ratio and convergence time. To mitigate these limitations, we present a systematic weight pruning framework of DNNs using the alternating direction method of multipliers (ADMM). We first formulate the weight pruning problem of DNNs as a nonconvex optimization problem with combinatorial constraints specifying the sparsity requirements, and then adopt the ADMM framework for systematic weight pruning. By using ADMM, the original nonconvex optimization problem is decomposed into two subproblems that are solved iteratively. One of these subproblems can be solved using stochastic gradient descent, the other can be solved analytically. Besides, our method achieves a fast convergence rate. The weight pruning results are very promising and consistently outperform the prior work. On the LeNet-5 model for the MNIST data set, we achieve 71.2× weight reduction without accuracy loss. On the AlexNet model for the ImageNet data set, we achieve 21× weight reduction without accuracy loss. When we focus on the convolutional layer pruning for computation reductions, we can reduce the total computation by five times compared with the prior work (achieving a total of 13.4× weight reduction in convolutional layers). Our models and codes are released at `https://github.com/KaiqiZhang/admm-pruning`.

Keywords: systematic weight pruning, deep neural networks (DNNs), alternating direction method of multipliers (ADMM)

1 Introduction

Large-scale deep neural networks or DNNs have made breakthroughs in many fields, such as image recognition [1,2,3], speech recognition [4,5], game playing [6], and driver-less cars [7]. Despite the huge success, their large model size and computational requirements will add significant burden to state-of-the-art computing systems [1,8,9], especially for embedded and IoT systems. As a result, a number of prior works are dedicated to *model compression* in order to simultaneously reduce the computation and model storage requirements of DNNs, with minor effect on the overall accuracy. These model compression techniques include *weight pruning* [9,10,11,12,13,14,15,16], sparsity regularization [17,18,19], weight clustering [9,20,21], and low rank approximation [22,23], etc.

A simple but effective weight pruning method has been proposed in [10], which prunes the relatively less important weights and performs retraining for maintaining accuracy in an iterative manner. It can achieve 9$\times$ weight reduction ratio on the AlexNet model with virtually no accuracy degradation. This method has been extended and generalized in multiple directions, including energy efficiency-aware pruning [12], structure-preserved pruning using regularization methods [17], and employing more powerful (and time-consuming) heuristics such as evolutionary algorithms [11]. While existing pruning methods achieve good model compression ratios, they are heuristic (and therefore cannot achieve optimal compression ratio), lack theoretical guarantees on compression performance, and require time-consuming iterative retraining processes.

To mitigate these shortcomings, we present a systematic framework of weight pruning and model compression, by (i) formulating the weight pruning problem as a constrained nonconvex optimization problem with combinatorial constraints, which employs the cardinality function to induce sparsity of the weights, and (ii) adopting the *alternating direction method of multipliers* (ADMM) [24] for systematically solving this optimization problem. By using ADMM, the original nonconvex optimization problem is decomposed into two subproblems that are solved iteratively. In the weight pruning problem, one of these subproblems can be solved using stochastic gradient descent, and the other can be solved analytically. Upon convergence of ADMM, we remove the weights which are (close to) zero and retrain the network.

Our extensive numerical experiments indicate that ADMM works very well in weight pruning. The weight pruning results consistently outperform the prior work. On the LeNet-5 model for the MNIST data set, we achieve 71.2$\times$ weight reduction without accuracy loss, which is 5.9 times compared with [10]. On the AlexNet model for the ImageNet data set, we achieve 21$\times$ weight reduction without accuracy loss, which is 2.3 times compared with [10]. Moreover, when we focus on the convolutional layer pruning for computation reductions, we can reduce the total computation by five times compared with the prior work (achieving a total of 13.4$\times$ weight reduction in convolutional layers). Our models and codes are released at `https://github.com/KaiqiZhang/admm-pruning`.

2 Related Work on Weight Reduction/Model Compression

Mathematical investigations have demonstrated a significant margin for weight reduction in DNNs due to the redundancy across filters and channels, and a number of prior works leverage this property to reduce weight storage. The techniques can be classified into two categories: *1) Low rank approximation* methods

[22,23] such as Singular Value Decomposition (SVD), which are typically difficult to achieve zero accuracy degradation with compression, especially for very large DNNs; *2) Weight pruning* methods which aim to remove the redundant or less important weights, thereby achieving model compression with negligible accuracy loss.

A prior work [10] serves as a pioneering work for weight pruning. It uses a heuristic method of iteratively pruning the unimportant weights (weights with small magnitudes) and retraining the DNN. It can achieve a good weight reduction ratio, e.g., 9× for AlexNet, with virtually zero accuracy degradation, and can be combined with other model compression techniques such as weight clustering [9,20]. It has been extended in several works. For instance, the *energy efficiency-aware pruning* method [12] has been proposed to facilitate energy-efficient hardware implementations, allowing for certain accuracy degradation. The *structured sparsity learning* technique has been proposed to partially overcome the limitation in [10] of irregular network structure after pruning. However, neither technique can outperform the original method [10] in terms of compression ratio under the same accuracy. There is recent work [11] that employs an evolutionary algorithm for weight pruning, which incorporates randomness in both pruning and growing of weights, following certain probabilistic rules. Despite the higher compression ratio it achieves, it suffers from a prohibitively long retraining phase. For example, it needs to start with an already-compressed model for further pruning on the ImageNet data set, instead of the original AlexNet or VGG models.

In summary, the prior weight pruning methods are highly heuristic and suffer from a long retraining phase. On the other hand, our proposed method is a systematic framework, achieves higher compression ratio, exhibits faster convergence rate, and is also general for structured pruning and weight clustering.

3 Background of ADMM

ADMM was first introduced in the 1970s, and theoretical results in the following decades have been collected in [24]. It is a powerful method for solving regularized convex optimization problems, especially for problems in applied statistics and machine learning. Moreover, recent works [25,26] demonstrate that ADMM is also a good tool for solving nonconvex problems, potentially with combinatorial constraints, since it can converge to a solution that may not be globally optimal but is sufficiently good for many applications.

For some problems which are difficult to solve directly, we can use variable splitting first, and then employ ADMM to decompose the problem into two subproblems that can be solved separately and efficiently. For example, the optimization problem

$$\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}) + g(\mathbf{x}), \tag{1}$$

assumes that $f(\cdot)$ is differentiable and $g(\cdot)$ is non-differentiable but has exploitable structure properties. Common instances of g are the ℓ_1 norm and the indicator function of a constraint set. To make it suitable for the application of ADMM, we use variable splitting to rewrite the problem as

$$\begin{aligned} \underset{\mathbf{x},\ \mathbf{z}}{\text{minimize}} \quad & f(\mathbf{x}) + g(\mathbf{z}), \\ \text{subject to} \quad & \mathbf{x} = \mathbf{z}. \end{aligned}$$

Next, via the introduction of the augmented Lagrangian, the above optimization problem can be decomposed into two subproblems in $\mathbf{x}$ and $\mathbf{z}$ [24]. The first subproblem is $\underset{\mathbf{x}}{\text{minimize}}\ f(\mathbf{x}) + q_1(\mathbf{x})$, where $q_1(\cdot)$ is a quadratic function of its argument. Since f and q_1 are differentiable, the first subproblem can be solved by gradient descent. The second subproblem is $\underset{\mathbf{z}}{\text{minimize}}\ g(\mathbf{z})+q_2(\mathbf{z})$, where $q_2(\cdot)$ is a quadratic function of its argument. In problems where g has some special structure, for instance if it is a regularizer in (1), exploiting the properties of g may allow this problem to be solved analytically. More details regarding the application of ADMM to the weight pruning problem will be demonstrated in Section 4.2.

4 Problem Formulation and Proposed Framework

4.1 Problem Formulation of Weight Pruning

Consider an N-layer DNN, where the collection of weights in the i-th (convolutional or fully-connected) layer is denoted by $\mathbf{W}_i$ and the collection of biases in the i-th layer is denoted by $\mathbf{b}_i$. In a convolutional layer the weights are organized in a four-dimensional tensor and in a fully-connected layer they are organized in a two-dimensional matrix [26].

Assume that the input to the (fully-connected) DNN is $\mathbf{x}$. Every column of $\mathbf{x}$ corresponds to a training image, and the number t of columns determines the number of training images in the input batch. The input $\mathbf{x}$ will enter the first layer and the output of the first layer is calculated by

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1),$$

where $\mathbf{h}_1$ and $\mathbf{b}_1$ have t columns, and $\mathbf{b}_1$ is a matrix with identical columns. The non-linear activation function $\sigma(\cdot)$ acts entry-wise on its argument, and is typically chosen to be the ReLU function [27] in state-of-the-art DNNs. Since the output of one layer is the input of the next, the output of the i-th layer for $i = 2, \ldots, N-1$ is given by

$$\mathbf{h}_i = \sigma(\mathbf{W}_i\mathbf{h}_{i-1} + \mathbf{b}_i).$$

The output of the DNN corresponding to a batch of images is

$$\mathbf{s} = \mathbf{W}_N\mathbf{h}_{N-1} + \mathbf{b}_N.$$

In this case $\mathbf{s}$ is a $k \times t$ matrix, where k is the number of classes in the classification, and t is the number of training images in the batch. The element

$\mathbf{s}_{ij}$ in matrix $\mathbf{s}$ is the score of the j-th training image corresponding to the i-th class. The total loss of the DNN is calculated as

$$f\big(\{\mathbf{W}_1,\dots,\mathbf{W}_N\},\{\mathbf{b}_1,\dots,\mathbf{b}_N\}\big) = -\frac{1}{t}\sum_{j=1}^{t}\log\frac{e^{\mathbf{s}_{y_j j}}}{\sum_{i=1}^{k} e^{\mathbf{s}_{ij}}} + \lambda\sum_{i=1}^{N}\|\mathbf{W}_i\|_F^2,$$

where $\|\cdot\|_F^2$ denotes the Frobenius norm, the first term is cross-entropy loss, y_j is the correct class of the j-th image, and the second term is L_2 weight regularization.

Hereafter, for simplicity of notation we write $\{\mathbf{W}_i\}_{i=1}^N$, or simply $\{\mathbf{W}_i\}$, instead of $\{\mathbf{W}_1,\dots,\mathbf{W}_N\}$. The same notational convention applies to writing $\{\mathbf{b}_i\}$ instead of $\{\mathbf{b}_1,\dots,\mathbf{b}_N\}$. The training of a DNN is a process of minimizing the loss by updating weights and biases. If we use the gradient descent method, the update at every step is

$$\begin{aligned}\mathbf{W}_i &= \mathbf{W}_i - \alpha\frac{\partial f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big)}{\partial \mathbf{W}_i},\\ \mathbf{b}_i &= \mathbf{b}_i - \alpha\frac{\partial f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big)}{\partial \mathbf{b}_i},\end{aligned}$$

for $i=1,\dots,N$, where α is the learning rate.

Our objective is to prune the weights of the DNN, and therefore we minimize the loss function subject to constraints on the cardinality of weights in each layer. More specifically, our training process solves

$$\begin{aligned}&\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} && f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big),\\ &\text{subject to} && \text{card}(\mathbf{W}_i)\le l_i,\ i=1,\dots,N,\end{aligned}$$

where card$(\cdot)$ returns the number of nonzero elements of its matrix argument and l_i is the desired number of weights in the i-th layer of the DNN[1]. A prior work [28] uses ADMM for DNN training with regularization in the objective function, which can result in sparsity as well. On the other hand, our method directly targets at sparsity with incorporating hard constraints on the weights, thereby resulting in a higher degree of sparsity.

4.2 Systematic Weight Pruning Framework using ADMM

We can rewrite the above weight pruning optimization problem as

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

[1] Our framework is also compatible with the constraint of l total number of weights for the whole DNN.

where $\mathbf{S}_i = \{\mathbf{W}_i \mid \text{card}(\mathbf{W}_i) \le l_i\}, i = 1, \ldots, N$. It is clear that $\mathbf{S}_1, \ldots, \mathbf{S}_N$ are nonconvex sets, and it is in general difficult to solve optimization problems with nonconvex constraints. The problem can be equivalently rewritten in a form without constraint, which is

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} g_i(\mathbf{W}_i), \tag{2}$$

where $g_i(\cdot)$ is the indicator function of $\mathbf{S}_i$, i.e.,

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

The first term of problem (2) is the loss function of a DNN, while the second term is non-differentiable. This problem cannot be solved analytically or by stochastic gradient descent. A recent paper [24], however, demonstrates that such problems lend themselves well to the application of ADMM, via a special decomposition into simpler subproblems. We begin by equivalently rewriting the above problem in ADMM form as

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

The augmented Lagrangian [24] of the above optimization problem is given by

$$\begin{aligned} L_\rho\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\},\{\mathbf{Z}_i\},\{\mathbf{\Lambda}_i\}\big) &= f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i) \\ &\quad + \sum_{i=1}^{N} \text{tr}[\mathbf{\Lambda}_i^T(\mathbf{W}_i - \mathbf{Z}_i)] + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i\|_F^2, \end{aligned}$$

where $\mathbf{\Lambda}_i$ has the same dimension as $\mathbf{W}_i$ and is the Lagrange multiplier (also known as the dual variable) corresponding to the constraint $\mathbf{W}_i = \mathbf{Z}_i$, the positive scalars $\{\rho_1, \ldots, \rho_N\}$ are penalty parameters, $\text{tr}(\cdot)$ denotes the trace, and $\|\cdot\|_F^2$ denotes the Frobenius norm. Defining the scaled dual variable $\mathbf{U}_i = (1/\rho_i)\mathbf{\Lambda}_i$, the augmented Lagrangian can be equivalently expressed as

$$\begin{aligned} L_\rho\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\},\{\mathbf{Z}_i\},\{\mathbf{\Lambda}_i\}\big) &= f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i) \\ &\quad + \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{W}_i - \mathbf{Z}_i + \mathbf{U}_i\|_F^2 - \sum_{i=1}^{N} \frac{\rho_i}{2} \|\mathbf{U}_i\|_F^2. \end{aligned}$$

The ADMM algorithm proceeds by repeating, for $k = 0, 1, \ldots,$ the following steps [24,29]:

$$\{\mathbf{W}_i^{k+1}, \mathbf{b}_i^{k+1}\} := \underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\arg\min} \quad L_\rho\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}, \{\mathbf{Z}_i^k\}, \{\mathbf{U}_i^k\}\big) \tag{3}$$

$$\{\mathbf{Z}_i^{k+1}\} := \underset{\{\mathbf{Z}_i\}}{\arg\min} \quad L_\rho\big(\{\mathbf{W}_i^{k+1}\}, \{\mathbf{b}_i^{k+1}\}, \{\mathbf{Z}_i\}, \{\mathbf{U}_i^k\}\big) \tag{4}$$

$$\mathbf{U}_i^{k+1} := \mathbf{U}_i^k + \mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1}, \tag{5}$$

until both of the following conditions are satisfied

$$\|\mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1}\|_F^2 \ \le \epsilon_i, \quad \|\mathbf{Z}_i^{k+1} - \mathbf{Z}_i^k\|_F^2 \ \le \epsilon_i. \tag{6}$$

In order to solve the overall pruning problem, we need to solve subproblems (3) and (4). More specifically, problem (3) can be formulated as

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}\big) + \sum_{i=1}^N \frac{\rho_i}{2}\|\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k\|_F^2, \tag{7}$$

where the first term is the loss function of the DNN, and the second term can be considered as a special L_2 regularizer. Since the regularizer is a differentiable quadratic norm, and the loss function of the DNN is differentiable, problem (7) can be solved by stochastic gradient descent. More specifically, the gradients of the augmented Lagrangian with respect to $\mathbf{W}_i$ and $\mathbf{b}_i$ are given by

$$\frac{\partial L_\rho\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}, \{\mathbf{Z}_i^k\}, \{\mathbf{U}_i^k\}\big)}{\partial \mathbf{W}_i} = \frac{\partial f\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}\big)}{\partial \mathbf{W}_i} + \rho_i(\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k),$$

$$\frac{\partial L_\rho\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}, \{\mathbf{Z}_i^k\}, \{\mathbf{U}_i^k\}\big)}{\partial \mathbf{b}_i} = \frac{\partial f\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}\big)}{\partial \mathbf{b}_i}.$$

Note that we cannot prove optimality of the solution to subproblem (3), just as we can not prove optimality of the solution to the original DNN training problem due to the nonconvexity of the loss function of DNN.

On the other hand, problem (4) can be formulated as

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

Since $g_i(\cdot)$ is the indicator function of the set $\mathbf{S}_i$, the globally optimal solution of this problem can be explicitly derived as [24]:

$$\mathbf{Z}_i^{k+1} = \mathbf{\Pi}_{\mathbf{S}_i}(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k), \tag{8}$$

where $\mathbf{\Pi}_{\mathbf{S}_i}(\cdot)$ denotes the Euclidean projection onto the set $\mathbf{S}_i$. Note that $\mathbf{S}_i$ is a nonconvex set, and computing the projection onto a nonconvex set is a difficult problem in general. However, the special structure of $\mathbf{S}_i = \{\mathbf{W} \mid \text{card}(\mathbf{W}) \le l_i\}$ allows us to express this Euclidean projection analytically. Namely, the solution of (4) is to keep the l_i elements of $\mathbf{W}_i^{k+1} + \mathbf{U}_i^k$ with the largest magnitudes and set the rest to zero [24]. Finally, we update the dual variable $\mathbf{U}_i$ according to (5). This concludes one iteration of the ADMM algorithm.

We observe that the proposed systematic framework exhibits multiple major advantages in comparison with the heuristic weight pruning method in [10]. Our proposed method achieves a higher compression ratio with a higher convergence rate compared with the iterative pruning and retraining method in [10]. For example, we achieve 15$\times$ compression ratio on AlexNet with only 10 iterations of ADMM. Additionally, subproblem (3) can be solved in a fraction of the number of iterations needed for training the original network when we use warm start initialization, i.e., when we initialize subproblem (3) with $\{\mathbf{W}_i^k, \mathbf{b}_i^k\}$ in order to find $\{\mathbf{W}_i^{k+1}, \mathbf{b}_i^{k+1}\}$. For example, when training on the AlexNet model using the ImageNet data set, convergence is achieved in approximately $\frac{1}{10}$ of the total iterations required for the original DNN training. Also, problems (4) and (5) are straightforward to carry out, thus their computational time can be ignored. As a synergy of the above effects, the total computational time of 10 iterations of ADMM will be similar to (or at least in the same order of) the training time of the original DNN. Furthermore, we achieve 21$\times$ compression ratio on AlexNet without accuracy loss when we use 40 iterations of ADMM.

4.3 The Final Retraining Step

For very small values of ϵ_i in (6), ADMM needs a large number of iterations to converge. However, in many applications, such as the weight pruning problem considered here, a slight increase in the value of ϵ_i can result in a significant speedup in convergence. On the other hand, when ADMM stops early, the weights to be pruned may not be identically zero, in the sense that there will be small nonzero elements contained in $\mathbf{W}_i$. To deal with this issue, we keep the l_i elements with the largest magnitude in $\mathbf{W}_i$, set the rest to zero and no longer involve these elements in training (i.e., we prune these weights). Then, we *retrain the DNN*. Note that we only need a single retraining step and the convergence is much faster than training the original DNN, since the starting point of the retraining is already close to the point which can achieve the original test/validation accuracy.

4.4 Overall Illustration of Our Proposed Framework

We take the weight distribution of every (convolutional or fully connected) layer on LeNet-5 as an example to illustrate our systematic weight pruning method. The weight distributions at different stages are shown in Figure 1. The subfigures in the left column show the weight distributions of the pretrained model, which serves as our starting point. The subfigures in the middle column show that after the convergence of ADMM for moderate values of ϵ_i, we observe a clear separation between weights whose values are close to zero and the remaining weights. To prune the weights rigorously, we set the values of the close-to-zero weights exactly to zero and retrain the DNN without updating these values.

The subfigures in the right column show the weight distributions after our final retraining step. We observe that most of the weights are zero in every layer. This concludes our weight pruning procedure.

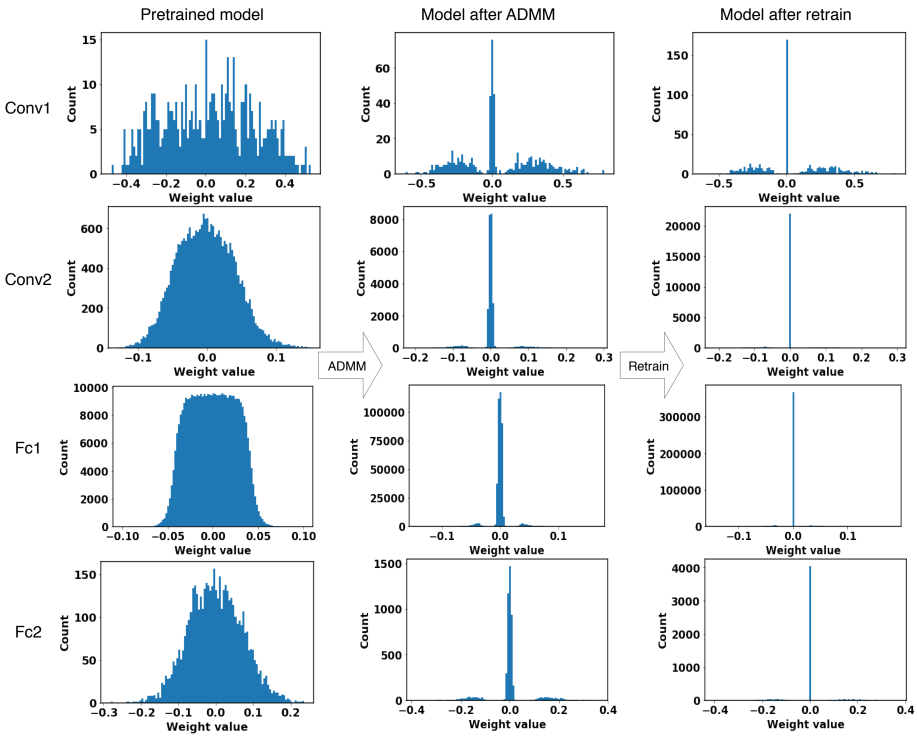


Fig. 1. Weight distribution of every (convolutional or fully connected) layer on LeNet-5. The subfigures in the left column are the weight distributions of the pretrained DNN model (serving as our starting point); the subfigures of the middle column are the weight distributions after the ADMM procedure; the subfigures of the right column are the weight distributions after our final retraining step. Note that the subfigures in the last column include a small number of nonzero weights that are not clearly visible due to the large number of zero weights.

5 Experimental Results

We have tested the proposed systematic weight pruning framework on the MNIST benchmark using the LeNet-300-100 and LeNet-5 models [2] and the ImageNet ILSVRC-2012 benchmark on the AlexNet model [1], in order to perform an apple-to-apple comparison with the prior heuristic pruning work [10]. The LeNet models are implemented and trained in TensorFlow [30] and the AlexNet models

Table 1. Weight pruning results on LeNet-300-100 network

Layer	Weights	Weights after prune	Weights after prune %	Result of [10] %
fc1	235.2K	9.41K	4%	8%
fc2	30K	2.1K	7%	9%
fc3	1K	0.12K	12%	26%
Total	266.2K	11.6K	4.37%	8%

Table 2. Weight pruning results on LeNet-5 network

Layer	Weights	Weights after prune	Weights after prune %	Result of [10] %
conv1	0.5K	0.1K	20%	66%
conv2	25K	2K	8%	12%
fc1	400K	3.6K	0.9%	8%
fc2	5K	0.35K	7%	19%
Total	430.5K	6.05K	1.4%	8%

are trained in Caffe [31]. We carry out our experiments on NVIDIA Tesla P100 GPUs. The weight pruning results consistently outperform the prior work. On the LeNet-5 model, we achieve 71.2× weight reduction without accuracy loss, which is 5.9 times compared with [10]. On the AlexNet model, we achieve 21× weight reduction without accuracy loss, which is 2.3 times compared with [10]. Moreover, when we focus on the convolutional layer pruning for computation reductions, we can reduce the total computation by five times compared with the prior work [10].

5.1 Testing results on LeNet models on MNIST data set

Table 1 shows our per-layer pruning results on the LeNet-300-100 model. LeNet-300-100 is a fully connected network with 300 and 100 neurons on the two hidden layers, respectively, and achieves 98.4% test accuracy on the MNIST benchmark. Table 2 shows our per-layer pruning results on the LeNet-5 model. LeNet-5 contains two convolutional layers, two pooling layers and two fully connected layers, and can achieve 99.2% test accuracy on the MNIST benchmark.

Our pruning framework does not incur accuracy loss and can achieve a much higher compression ratio on these networks compared with the prior iterative pruning heuristic [10], which reduces the number of parameters by 12× on both LeNet-300-100 and LeNet-5. On the LeNet-300-100 model, our pruning method reduces the number of weights by 22.9×, which is 90% higher than [10]. Also, our pruning method reduces the number of weights by 71.2× on the LeNet-5 model, which is 5.9 times compared with [10].

5.2 Testing results on AlexNet model using ImageNet benchmark

We implement our systematic weight pruning method using the BAIR/BVLC AlexNet model[2] on the ImageNet ILSVRC-2012 benchmark. The implemen-

[2] `https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet`

tation is on the Caffe tool because it is faster than TensorFlow. The original BAIR/BVLC AlexNet model can achieve a top-5 accuracy 80.2% on the validation set. AlexNet contains 5 convolutional (and pooling) layers and 3 fully connected layers with a total of 60.9M parameters, with the detailed network structure shown in deploy.prototxt text on the website indicated in footnote 2.

Our first set of experiments only target model size reductions for AlexNet, and the results are shown in Table 3. It can be observed that our pruning method can reduce the number of weights by 21× on AlexNet, which is more than twice compared with the prior iterative pruning heuristic. We achieve a top-5 accuracy of 80.2% on the validation set of ImageNet ILSVRC-2012. Layer-wise comparison results are also shown in Table 3, while comparisons with some other model compression methods are shown in Table 5. These results clearly demonstrate the advantage of the proposed systematic weight pruning framework using ADMM.

Table 3. Weight pruning results on AlexNet network (purely focusing on weight reductions) without accuracy loss

Layer	Weights	Weights after prune	Weights after prune%	Result of [10]
conv1	34.8K	28.19K	81%	84%
conv2	307.2K	61.44K	20%	38%
conv3	884.7K	168.09K	19%	35%
conv4	663.5K	132.7K	20%	37%
conv5	442.4K	88.48K	20%	37%
fc1	37.7M	1.06M	2.8%	9%
fc2	16.8M	0.99M	5.9%	9%
fc3	4.1M	0.38M	9.3%	25%
Total	60.9M	2.9M	4.76%	11%

Our second set of experiments target computation reduction besides weight reduction. Because the major computation in state-of-the-art DNNs is in the convolutional layers, we mainly target weight pruning in these layers. Although on AlexNet, the number of weights in convolutional layers is less than that in fully connected layers, the computation on AlexNet is dominated by its 5 convolutional layers. In our experiments, we conduct experiments which keep the same portion of weights as [10] in fully connected layers but prune more weights in convolutional layers. For AlexNet, Table 4 shows that we can reduce the number of weights by 13.4× in convolutional layers, which is five times compared with 2.7× in [10]. This indicates our pruning method can reduce much more computation compared with the prior work [10]. Layer-wise comparison results are also shown in Table 4. Still, it is difficult to prune weights in the first convolutional layer because they are needed to directly extract features from the raw inputs. Our major gain is because (i) we can achieve significant weight reduction in conv2 through conv5 layers, and (ii) the first convolutional layer is relatively small and less computational intensive.

Table 4. Weight pruning results on AlexNet network (focusing on computation reductions) without accuracy loss

Layer	Weights	Weights after prune	Weights after prune%	Result of [10]
conv1	34.8K	21.92K	63%	84%
conv2	307.2K	21.5K	7%	38%
conv3	884.7K	53.08K	6%	35%
conv4	663.5K	46.45K	7%	37%
conv5	442.4K	30.97K	7%	37%
fc1	37.7M	3.39M	9%	9%
fc2	16.8M	1.51M	9%	9%
fc3	4.1M	1.03M	25%	25%
Total of conv1-5	2332.6K	173.92k	7.46%	37.1%

Several extensions [12,17] of the original weight pruning work have improved in various directions such as energy efficiency for hardware implementation and regularity, but they cannot strictly outperform the original work [10] in terms of compression ratio under the same accuracy. The very recent work [11] employs an evolutionary algorithm for weight pruning, which incorporates randomness in both pruning and growing of weights following certain probability rules. It can achieve a comparable model size with our work. However, it suffers from a prohibitively long retraining phase. For example, it needs to start with an already-compressed model with 8.4M parameters for further pruning on ImageNet, instead of the original AlexNet model. By using an already-compressed model, it can reduce the number of neurons per layer as well, while such reduction is not considered in our proposed framework.

Table 5. Weight reduction ratio comparisons using different model compression techniques on the AlexNet model

Network	Top-5 Error	Parameters	Weight Reduction Ratio
Baseline AlexNet [1]	19.8%	60.9M	1.0×
SVD [23]	20.6%	11.9M	5.1×
Layer-wise pruning [32]	20.0%	6.7M	9.1×
Network pruning [10]	19.7%	6.7M	9.1×
Our result (10 iterations of ADMM)	19.8%	4.06M	15×
NeST [11]	19.7%	3.9M	15.7×
Dynamic surgery [14]	20.0%	3.45M	17.7×
Our result (25 iterations of ADMM)	19.8%	3.24M	18.8×
Our result (40 iterations of ADMM)	19.8%	2.9M	21×

6 Discussion

6.1 Parameters and initialization of ADMM

For nonconvex problems in general, there is no guarantee that ADMM will converge to an optimal point. ADMM can converge to different points for different choices of initial values $\{\mathbf{Z}_1^0, \ldots, \mathbf{Z}_N^0\}$ and $\{\mathbf{U}_1^0, \ldots, \mathbf{U}_N^0\}$ and penalty parameters $\{\rho_1, \ldots, \rho_N\}$ [24]. To resolve this limitation, we set the pretrained model $\{\mathbf{W}_i^p, \mathbf{b}_i^p\}$, a good solution of $\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}}\ f\big(\{\mathbf{W}_i\}, \{\mathbf{b}_i\}\big)$, to be the starting point when we use stochastic gradient descent to solve problem (7). We initialize $\mathbf{Z}_i^0$ by keeping the l_i elements of $\mathbf{W}_i^p$ with the largest magnitude and set the rest to be zero. We set $\mathbf{U}_1^0 = \cdots = \mathbf{U}_N^0 = 0$. For problem (7), if the penalty parameters $\{\rho_1, \ldots, \rho_N\}$ are too small, the solution will be close to the minimum of $f(\cdot)$ but fail to regularize the weights, and the ADMM procedure may converge slowly or not converge at all. If the penalty parameters are too large, the solution may regularize the weights well but fail to minimize $f(\cdot)$, and therefore the accuracy of the DNN will be degradated. In actual experiments, we find that $\rho_1 = \cdots = \rho_N = 10^{-4}$ is an appropriate choice for LeNet-5 and LeNet-300-100, and that $\rho_1 = \cdots = \rho_N = 1.5 \times 10^{-3}$ works well for AlexNet.

6.2 Parameters of the desired number of weights in each layer

We initialize l_i based on existing results in the literature, then we implement our weight pruning method on the DNN and test its accuracy. If there is no accuracy loss on the DNN, we decrease l_i in every layer proportionally. We use binary search to find the smallest l_i that will not result in accuracy loss.

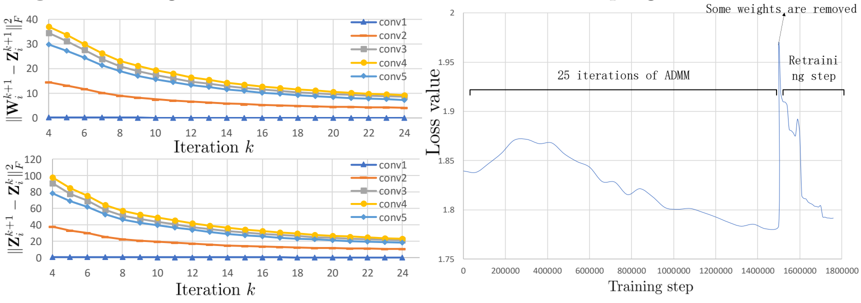
6.3 Convergence behavior of ADMM and loss value progression on AlexNet

Convergence behavior of ADMM (5 CONV layers in AlexNet) is shown in Figure 2 (left sub-figure). The loss value progression of AlexNet is shown in Figure 2 (right sub-figure). We start from an existing DNN model without pruning. After the convergence of ADMM, we remove the weights which are (close to) zero, which results in an increase in the loss value. We then retrain the DNN and the loss decreases to the same level as it was before pruning.

6.4 Discussion on our proposed framework

The cardinality function is nonconvex and nondifferentiable, which complicates the use of standard gradient algorithms. ADMM circumvents the issue of differentiability systematically, and does so without introducing additional numerical complexity. Furthermore, although ADMM achieves global optimality for convex problems, it has been shown in the optimization literature that it performs extremely well for large classes of nonconvex problems. In fact, ADMM-based

Fig. 2. Convergence behavior of ADMM and loss value progression of AlexNet



pruning can be perceived as a smart regularization technique in which the regularization target will be dynamically updated in each iteration. The limitation is that we need to tune the parameters l_i. However, *some* parameter tuning is generally inevitable; even a soft regularization parameter requires fine-tuning in order to achieve the desired solution structure. On the positive side, the freedom in setting l_i allows the user to obtain the exact desired level of sparsity.

7 Conclusions and Future Work

In this paper, we presented a systematic DNN weight pruning framework using ADMM. We formulate the weight pruning problem of DNNs as a nonconvex optimization problem with combinatorial constraints specifying the sparsity requirements. By using ADMM, the nonconvex optimization problem is decomposed into two subproblems that are solved iteratively, one using stochastic gradient descent and the other analytically. We reduced the number of weights by 22.9$\times$ on LeNet-300-100 and 71.2$\times$ on LeNet-5 without accuracy loss. For AlexNet, we reduced the number of weights by 21$\times$ without accuracy loss. When we focued on computation reduction, we reduced the number of weights in convolutional layers by 13.4$\times$ on AlexNet, which is five times compared with the prior work.

In future work, we will extend the proposed weight pruning method to incorporate structure and regularity in the weight pruning procedure, and develop a unified framework of weight pruning, activation reduction, and weight clustering.

Acknowledgments Financial support from the National Science Foundation under awards CNS-1840813, CNS-1704662 and ECCS-1609916 is gratefully acknowledged.

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. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324
3. 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
4. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine **29**(6) (2012) 82–97
5. Dahl, G.E., Yu, D., Deng, L., Acero, A.: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on audio, speech, and language processing **20**(1) (2012) 30–42
6. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
7. Makantasis, K., Karantzalos, K., Doulamis, A., Doulamis, N.: Deep supervised learning for hyperspectral data classification through convolutional neural networks. In: Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International, IEEE (2015) 4959–4962
8. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
9. 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)
10. 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
11. 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)
12. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. arXiv preprint arXiv:1611.05128 (2016)
13. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: International Conference on Machine Learning. (2017) 2498–2507
14. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient dnns. In: Advances In Neural Information Processing Systems. (2016) 1379–1387
15. Tung, F., Muralidharan, S., Mori, G.: Fine-pruning: Joint fine-tuning and compression of a convolutional network with bayesian optimization. arXiv preprint arXiv:1707.09102 (2017)
16. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. In: 2017 IEEE International Conference on Computer Vision (ICCV), IEEE (2017) 5068–5076
17. 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
18. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 806–814
19. Zhou, H., Alvarez, J.M., Porikli, F.: Less is more: Towards compact cnns. In: European Conference on Computer Vision, Springer (2016) 662–677

20. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: International Conference on Machine Learning. (2015) 2285–2294
21. Park, E., Ahn, J., Yoo, S.: Weighted-entropy-based quantization for deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2017)
22. 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
23. 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
24. 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
25. Takapoui, R., Moehle, N., Boyd, S., Bemporad, A.: A simple effective heuristic for embedded mixed-integer quadratic programming. International Journal of Control (2017) 1–11
26. Leng, C., Li, H., Zhu, S., Jin, R.: Extremely low bit neural network: Squeeze the last bit out with admm. arXiv preprint arXiv:1707.09870 (2017)
27. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml. Volume 30. (2013) 3
28. Kiaee, F., Gagné, C., Abbasi, M.: Alternating direction method of multipliers for sparse convolutional neural networks. arXiv preprint arXiv:1611.01590 (2016)
29. Liu, S., Fardad, M., Masazade, E., Varshney, P.K.: On optimal periodic sensor scheduling for field estimation in wireless sensor networks. In: Global Conference on Signal and Information Processing (GlobalSIP), IEEE (2013) 137–140
30. Abadi, M., Agarwal, A., Barham, P., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
31. 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
32. 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