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[1]
1. Syracuse University 2. Florida International University *Equal Contribution
1. {tzhan120,sye106,kzhang17,jtang02,makan,ywang393}@syr.edu
2. wwen@fiu.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, while the other can be solved analytically. The proposed ADMM weight pruning method incurs no additional suboptimality besides that resulting from the nonconvex nature of the original optimization problem. Furthermore, our approach achieves a fast convergence rate.
The weight pruning results are very promising and consistently outperform prior work. On the LeNet-5 model for the MNIST data set, we achieve 40.2× weight reduction without accuracy loss. On the AlexNet model for the ImageNet data set, we achieve 20× 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 prior work (achieving a total of 13.4× weight reduction in convolutional layers). A significant acceleration for DNN training is observed as well, in that we can finish the whole training process on AlexNet around 80 hours. Our models 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], sparsity regularization [12,14], weight clustering [9,15], and low rank approximation [16,17], 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× weight pruning on the AlexNet model with virtually no accuracy degradation. This method has been extended and generalized in multiple directions, including energy efficiency-aware pruning [13], structure-preserved pruning using regularization methods [12], 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* or ADMM [18] 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, while the other can be solved analytically. Upon convergence of ADMM, we remove the weights which are (close to) zero and retrain the network.

In theory, the ADMM-based method *incurs no additional suboptimality besides that resulting from the nonconvex nature of the original optimization problem and the final retraining process;* this is in contrast to the prior heuristic pruning method [10], which incurs additional suboptimality due to its greedy nature. Furthermore, ADMM displays a fast convergence rate for a wide range of applications. In our experiments, only around 20 iterations of ADMM are needed to attain convergence; each ADMM iteration only requires approximately $\frac{1}{10}$ of the total iterations compared with the original DNN training, thereby making the proposed algorithm much faster than the prior heuristic pruning method. Finally, the proposed systematic weight pruning is a general framework that can be applied to structured weight pruning and general weight clustering as well.

Our extensive numerical experiments indicate that ADMM works very well in practice and is highly suitable for weight pruning. The weight pruning results consistently outperform prior work. On the LeNet-5 model for the MNIST data set, we achieve 40.2× weight reduction without accuracy loss, which is 3.3

times compared with [10]. On the AlexNet model for the ImageNet data set, we achieve 20× weight reduction without accuracy loss, which is more than twice compare 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 prior work (achieving a total of 13.4× weight reduction in convolutional layers). A significant acceleration for DNN training is observed as well, in that we can finish the whole training process on AlexNet around 80 hours. Our models are released at `https://github.com/KaiqiZhang/admm-pruning`.

2 Related Work on Weight Reduction/Model Compression

Mathematical investigations have demonstrated 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 [16,17] such as Singular Value Decomposition (SVD), which are typically difficult to achieve zero accuracy degradation with compressions, 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.

Reference [10] serves as a pioneering work for weight pruning. It uses the heuristic method of iteratively pruning the unimportant weights (weights with small magnitudes) and retraining the DNN. It can achieve a good parameter 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 [9,15]. Several works have been conducted for extensions. For instance, the *energy efficiency-aware pruning* method [13] 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 re-training 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 techniques.

3 Background of ADMM

ADMM was first introduced in the 1970s, and theoretical results in the following decades are collected in [18]. It is a powerful method for solving regularized convex optimization problems, especially for problems in applied statistics and machine learning. Moreover, recent works [19,20] 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.

ADMM is based on decomposing an optimization problem, which is difficult to solve directly, into two subproblems that can be solved separately and efficiently. For example, the optimization problem

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

lends itself well to the application of ADMM if $f(\cdot)$ is differentiable and $g(\cdot)$ is non-differentiable but has some structure that can be exploited. Common instances of g are the ℓ_1 norm and the indicator function of a constraint set. To prepare it for the application of ADMM, the above problem is first rewritten 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 $\mathbf{z}$ [18]. 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 q_1 is differentiable and convex, the complexity of solving this problem (e.g., via gradient descent) is the same as that of minimizing f. 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 sturcture, for instance if it is a regularizer in (1), then exploiting the properties of g allows 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 [20].

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 entrywise on its argument, and is typically chosen to be the ReLU function [22] 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

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

where 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 $\{\mathbf{W}_1, \ldots, \mathbf{W}_N\}$. The same notational convention applies to writing $\{\mathbf{b}_i\}$ instead of $\{\mathbf{b}_1, \ldots, \mathbf{b}_N\}$. The training of a DNN is a process of minimizing the loss by updating weights and biases. If we use the gradient descent method then the update at every step is

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

computed for $i = 1, \ldots, N$, 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

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

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 [21] 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

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

where $\mathbf{S}_i = \{\mathbf{W} \mid \text{card}(\mathbf{W}) \le l_i\}, i = 1,\dots,N$. It is clear that $\mathbf{S}_1,\dots,\mathbf{S}_N$ 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

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

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 the above problem 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 [18], 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

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

The augmented Lagrangian [18] of the above optimization problem is given by

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

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

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 $\{\rho_1, \dots, \rho_N\}$ are penalty parameters, $\mathrm{tr}(\cdot)$ 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

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

The ADMM algorithm proceeds by repeating, for $k = 0, 1, \dots$, the following steps [18,23]:

$$\{\mathbf{W}_i^{k+1}, \mathbf{b}_i^{k+1}\} := \underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\arg\min} \quad L_\rho\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\},\{\mathbf{Z}_i^k\},\{\mathbf{U}_i^k\}\big) \tag{2}$$

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

$$\mathbf{U}_i^{k+1} := \mathbf{U}_i^k + \mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1}, \tag{4}$$

until both of the following conditions are satisfied

$$\|\mathbf{W}_i^{k+1} - \mathbf{Z}_i^{k+1}\|_F^2 \ \le \epsilon_i, \quad \|\mathbf{Z}_i^{k+1} - \mathbf{Z}_i^{k}\|_F^2 \ \le \epsilon_i. \tag{5}$$

In order to solve the overall pruning problem, we need to solve subproblems (2) and (3). More specifically, problem (2) can be formulated as

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} \frac{\rho_i}{2}\|\mathbf{W}_i - \mathbf{Z}_i^k + \mathbf{U}_i^k\|_F^2, \tag{6}$$

where the first term is the loss function of the DNN, and the second term can be considered as a special L_2 regularization. Since the regularizer is a differentiable quadratic norm, the computational complexity of minimizing the above loss function (for example, via stochastic gradient descent) is the same as the complexity of solving (training) the original DNN, namely the complexity of solving $\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}}\ f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big)$. 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 (2), just as we can not prove optimality of the solution to the original DNN training problem.

On the other hand, problem (3) 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 [18]:

$$\mathbf{Z}_i^{k+1} = \mathbf{\Pi}_{\mathbf{S}_i}(\mathbf{W}_i^{k+1} + \mathbf{U}_i^k), \tag{7}$$

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 (3) 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 [18]. Finally, we update the dual variable $\mathbf{U}_i$ according to (4). 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]. First, our proposed method achieves a higher convergence rate compared with the iterative pruning and retraining method in [10]. In theory, ADMM achieves fast convergence when its subproblems are convex [18]. Although the convexity requirement is not satisfied in the weight pruning problem of DNN, we can still achieve convergence in around 20 ADMM iterations. Additionally, subproblem (2) 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 (2) 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, problem (3) and (4) are straightforward to carry out, thus their computational time can be ignored. As a synergy of the above effects, the total time of the proposed framework will be similar to (or at least in the same order of) training original DNN twice.

Second, the proposed ADMM-based weight pruning method achieves a higher compression ratio under the same accuracy compared with prior work. We can say that solving problem (6) incurs "no additional suboptimality" compared with solving this problem when the quadratic term is absent from the objective. In other words, whenever optimality is not achieved in problem (6) *it is due to the inherent lack of convexity in the DNN loss function* and not due to solving this problem by ADMM. And we recall that the optimal solution of problem (7) is found analytically. On the other hand, prior work in iterative weight pruning incurs "additional suboptimality" due to the greedy nature of the iterative pruning and retraining heuristic. We observe the higher compression ratio more clearly in the experimental results on the MNIST and ImageNet data sets.

4.3 The Final Retraining Step

For very small values of ϵ_i in (5), 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.

The final retraining step will incur a certain degree of suboptimality to the overall procedure. However, the degree of suboptimality will be much lower compared with prior work [10] because the latter uses an iterative, greedy pruning and retraining framework.

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.

As mentioned before, the computation time for the ADMM procedure is similar to the training of the original DNN twice, and the single retraining step converges much faster than the original training. Consequently, the total computation time of our method is less than training the original DNN three times, which is much faster than the iterative pruning and training method in [10].

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 [24] while the AlexNet models are trained in Caffe [25]. We carried out our experiments on NVIDIA Tesla P100 GPUs. The weight pruning results consistently outperform the prior work. On the LeNet-5 model, we achieve 40.2× weight reduction without accuracy loss, which is 3.3 times compared with [10]. On the AlexNet model, we achieve 20× weight reduction without accuracy loss, which is more than twice compare 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 [10].

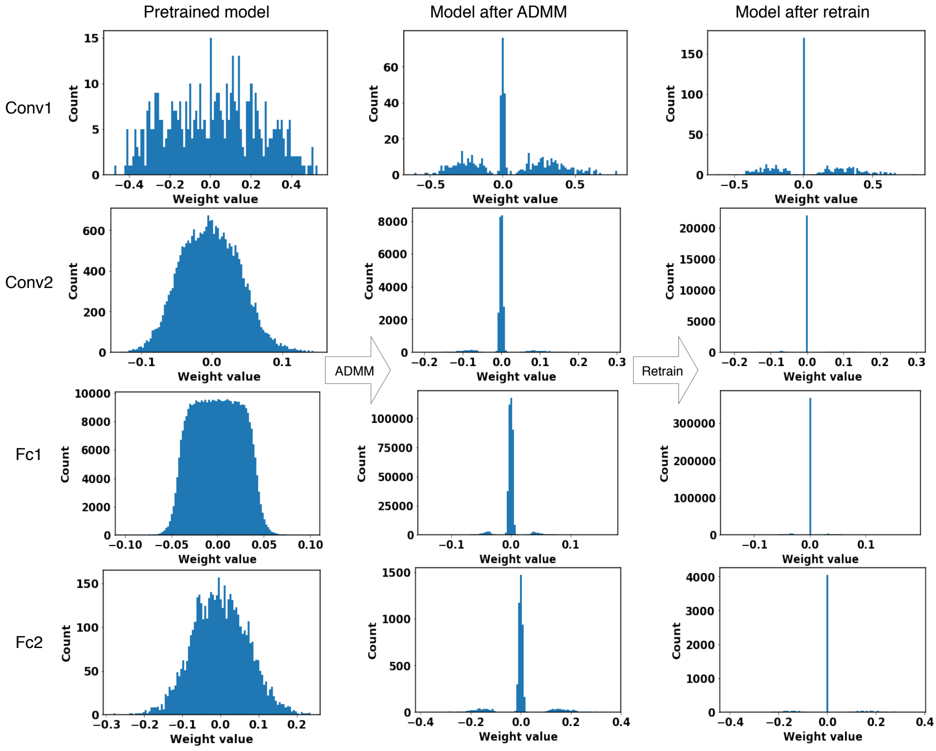

Fig. 1. Weight distribution of every (convolutional or fully connected) layer on LeNet-5. The subfigures in the left column are the weight distributions of the pretrained DNN model (serving as our starting point); the subfigures of the middle column are the weight distributions after the ADMM procedure; 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.

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	2.25K	9%	12%
fc1	400K	8K	2%	8%
fc2	5K	0.35K	7%	19%
Total	430.5K	10.7K	2.49%	8%

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 40.2× on the LeNet-5 model, which is 3.3 times compared with [10].

5.2 Testing results on AlexNet model using ImageNet benchmark

We implemented our systematic weight pruning method using the BAIR/BVLC AlexNet model[2] on the ImageNet ILSVRC-2012 benchmark. The implementation 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 detailed network structure shown in deploy.prototxt text on the website indicated in footnote 2.

Our first set of experiments only targets model size reductions for AlexNet, and the results are shown in Table 3. We do not allow for any accuracy loss compared with the original AlexNet model. It can be observed that our pruning

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

method can reduce the number of weights by 18× on AlexNet, which is twice compare with the prior iterative pruning heuristic. We achieve a top-5 accuracy of 80.3% 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	132.70K	20%	37%
conv5	442.4K	88.48K	20%	37%
fc1	37.7M	1.39M	3.7%	9%
fc2	16.8M	1.11M	6.6%	9%
fc3	4.1M	0.41M	10%	25%
Total	60.9M	3.39M	5.57%	11%

Our second set of experiments targets 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 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]. For the pruning ratio shown in Table 4, we achieve a top-5 accuracy 80.4% on the validation set of ImageNet ILSVRC-2012. This indicates our pruning method can reduce much more computation, almost by a factor of five, compared with prior work. Layerwise 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.

Our further experiments reduce the number of weights by 20× and achieve a top 5 accuracy 80% on the validation set of ImageNet ILSVRC-2012.

Several extensions [12,13] 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

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	21.50K	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%

an evolutionary algorithm for weight pruning, which incorporates randomness in both pruning and growing of weights following certain probability rules. It can achieve 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.

Finally, we would like to point out that the total training time of the ADMM framework can be very fast. We can finish the weight pruning on AlexNet around 80 hours, which is much faster than the training time reported in [10], i.e., 75+173 hours (this data may be outdated due to the significant improvement in GPU computation in recent years). This also demonstrates the advantage of the proposed framework.

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

Network	Top-5 Error (relative values)	Parameters	Weight Reduction Ratio
Baseline AlexNet [1]	19.8%	60.9M	1.0×
Naive Cut	+3.4%	13.8M	4.4×
SVD [17]	+0.8%	11.9M	5.1×
Layer-wise pruning [26]	+0.2%	6.7M	9.1×
Network pruning [10]	−0.1%	6.7M	9.1×
Our result (computation reduction)	−0.2%	6.07M	10.3×
Our result (focusing on weight reduction)	−0.1%	3.39M	18×
Our result (focusing on weight reduction)	+0.2%	3.05M	20×

6 Discussion on parameters and initialization of ADMM, and generalizations

For nonconvex problems in general, there is no guarantee that ADMM will converge to the 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 $\{\rho_1, \ldots, \rho_N\}$ [18]. 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 (6). 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 (6), 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 $\rho_1 = \cdots = \rho_N = 10^{-2}$ is an appropriate choice for LeNet-5, $\rho_1 = \cdots = \rho_N = 1.5 \times 10^{-2}$ is appropriate for LeNet-300-100, and $\rho_1 = \cdots = \rho_N = 1.5 \times 10^{-3}$ works well for AlexNet. With the above settings ADMM converges in approximately 20 iterations, and the resulting DNNs perform well in terms of test/validation accuracy.

Finally, we would like to point out that the proposed ADMM-based framework is general, in that it can take a broad category of nonconvex and combinatorial constraints. Therefore, it can be utilized in many extensions of the weight pruning problem, such as incorporating structure and regularity in the weight pruning results, combining weight pruning and weight clustering, simultaneous weight reduction and activation reduction, etc.

7 Conclusions and Future Work

In this paper, we have presented a systematic DNN weight pruning framework using ADMM. In this general framework, we initialize the ADMM procedure by using a pretrained model. 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. After ADMM convergence, we only need a single retraining step, which achieves a much higher convergence rate than the previous work. We reduced the number of weights by 22.9× on LeNet-300-100 and 40.2× on LeNet-5 without accuracy loss. For AlexNet, we reduced the number of weights by 20× without accuracy loss. If we focus on computation reduction, we can reduce the number of weights in convolutional layers by 13.4× on AlexNet, which is five times compared with the previous 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-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. 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
13. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. arXiv preprint arXiv:1611.05128 (2016)
14. 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

15. 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
16. 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
17. 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
18. 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
19. 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
20. 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)
21. Kiaee, F., Gagné, C., Abbasi, M.: Alternating direction method of multipliers for sparse convolutional neural networks. arXiv preprint arXiv:1611.01590 (2016)
22. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: Proc. icml. Volume 30. (2013) 3
23. 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
24. Abadi, M., Agarwal, A., Barham, P., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
25. 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
26. 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