ADMM-based Weight Pruning for Real-Time Deep Learning Acceleration on Mobile Devices

Hongjia Li
li.hongjia@husky.neu.edu
Northeastern University
Boston, USA

Ning Liu
Xiaolong Ma
Sheng Lin
Northeastern University
Boston, USA

Shaokai Ye
Tianyun Zhang
Syracuse University
Syracuse, USA

Xue Lin
xue.lin@northeastern.edu
Northeastern University
Boston, USA

Wenyao Xu
wenyaoxu@buffalo.edu
University at Buffalo
Buffalo, USA

Yanzhi Wang
yanz.wang@northeastern.edu
Northeastern University
Boston, USA

ABSTRACT

Deep learning solutions are being increasingly deployed in mobile applications, at least for the inference phase. Due to the large model size and computational requirements, model compression for deep neural networks (DNNs) becomes necessary, especially considering the real-time requirement in embedded systems. In this paper, we extend the prior work on systematic DNN weight pruning using ADMM (Alternating Direction Method of Multipliers). We integrate ADMM regularization with masked mapping/retraining, thereby guaranteeing solution feasibility and providing high solution quality. Besides superior performance on representative DNN benchmarks (e.g., AlexNet, ResNet), we focus on two new applications facial emotion detection and eye tracking, and develop a top-down framework of DNN training, model compression, and acceleration in mobile devices. Experimental results show that with negligible accuracy degradation, the proposed method can achieve significant storage/memory reduction and speedup in mobile devices.

CCS CONCEPTS

- Computing methodologies → Neural networks; Real-time simulation;

KEYWORDS

Mobile devices, neural networks, acceleration, real-time

1 INTRODUCTION

Recently, deep learning has been expanded into many new application fields, such as automatic drive system, 3D printing detection, and medical imaging and diagnosis [15, 19, 25]. As an example for the latter application, deep neural networks (DNNs) can be trained for the detection of facial expressions and performing eye tracking for the patients [6, 16]. By extracting complex and high-level features from large-scale data, DNNs can achieve a high accuracy and provide significant help and convenience for both doctors and patients.

DNNs are typically trained in an offline manner, and are often deployed in low-power, embedded, or mobile devices during inference. One of the major challenges is the large model size and computational requirement, which makes it difficult for real-time implementation in mobile devices. To overcome this challenge, many efforts have been devoted to DNN model compression from both industry and academia. One pioneering work [9] adopts an iterative heuristic for DNN weight pruning, achieving good pruning results: 9× weight reduction in AlexNet [17] and 12× in LeNet-5 [18]. Despite the promising results, the compression gain mainly focuses on the fully-connected (FC) layers, and the pruning ratio is limited on convolutional (CONV) layers (e.g., 2.7× for CONV layers in AlexNet). This limitation needs to be overcome as CONV layers become the most computationally intensive layers in current DNNs [11, 17].

Later weight pruning work extend to (i) use more sophisticated heuristic such as both weight prune and grow [4, 8], (ii) strike a desirable tradeoff between pruning ratio and accuracy, and (iii) incorporate regularity or structure in weight pruning framework [11, 17]. To partially overcome the heuristic nature, recently, a systematic DNN weight pruning framework has been proposed in [27], based on the powerful ADMM (Alternating Direction Methods of Multipliers) technique [1]. This work formulates the DNN weight pruning problem as a mathematical optimization problem, and observes the compatibility between the combinatorial constraints (associated with weight pruning) with ADMM. It achieves improved weight pruning results, 21× on AlexNet and 71.2× on LeNet-5, with no accuracy loss. However, this work adopts a direct application of ADMM, and lacks rigorous guarantee on feasibility (satisfying all
2 A SYSTEMATIC WEIGHT PRUNING FRAMEWORK USING ADMM

2.1 Systematic View of Weight Pruning

Similar to [27], we provide a systematic view of DNN weight pruning during training as an optimization problem. Consider a general N-layer DNN. Sets of weights and biases of the i-th (CONV or FC) layer are denoted by \( W_i \) and \( b_i \), respectively. Let us denote the loss function of the N-layer DNN as \( f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N}) \). Then the overall problem is defined as

\[
\begin{align*}
\text{minimize} & \quad f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N}), \\
\text{subject to} & \quad W_i \in S_i, \quad i = 1, \ldots, N.
\end{align*}
\]

(1)

The set \( S_i = \{ W_i | \text{card}(\text{supp}(W_i)) \leq \alpha_i \} \) reflects constraint for weight pruning, where ‘card’ refers to cardinality and ‘supp’ refers to support set. Elements in \( S_i \) are \( W_i \) solutions, satisfying that the number of non-zero elements in \( W_i \) is limited by \( \alpha_i \) for layer \( i \). Because of such combinatorial constraints, problem (1) cannot be solved using conventional stochastic gradient descent which assumes no hard constraints. This is key reason that prior work use heuristic methods to get rid of these constraints. To overcome this limitation, a key observation is that such form of combinatorial constraints is compatible with ADMM technique.

2.2 Connection to ADMM

ADMM [23, 24] is a powerful optimization tool, by decomposing an original problem into two subproblems that can be solved separately and iteratively. Consider optimization problem

\[
\min_x \quad f(x) + g(x). 
\]

(2)

In ADMM, the problem is first re-written as

\[
\min_{x,z} \quad f(x) + g(z), \quad \text{subject to} \quad x = z.
\]

(3)

Next, by using augmented Lagrangian [1], the above problem is decomposed into two subproblems on \( x \) and \( z \). The first is \( \min_x f(x) + q_1(x|z) \), where \( q_1(x|z) = \| x - z \|_2^2 \) is a quadratic function of \( x \) with fixed \( z \). Subproblem 2 is \( \min_z g(z) + q_2(z|x) \), where \( q_2(z|x) = \| z - x \|_2^2 \) is a quadratic function on \( z \) with fixed \( x \). The two subproblems will be solved iteratively until convergence is achieved [23, 24].

ADMM is conventionally utilized to accelerate convergence of convex optimization problems. The optimality and fast convergence have been proven for convex problems [1, 23]. As a special property, ADMM can effectively deal with a subset of combinatorial constraints and yields optimal (or at least high quality) solutions [14, 21].

Our observation is that associated constraints in DNN weight pruning belong to this subset. More specifically, we use indicator functions to incorporate combinatorial constraints into objective function. The indicator functions are \( g_i(W_i) = \begin{cases} 0 & \text{if } W_i \in S_i, \\ +\infty & \text{otherwise}. \end{cases} \)

Then problem (1) becomes:

\[
\begin{align*}
\text{minimize} & \quad f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N}) + \sum_{i=1}^{N} g_i(W_i) \\
& \quad \text{subject to} \quad W_i \in S_i, \quad i = 1, \ldots, N.
\end{align*}
\]

(4)

Despite the compatibility of the combinatorial constraints with ADMM, there is difficulty in using ADMM directly due to the non-convex nature of the objective function in (1). Therefore, special mechanisms are needed to guarantee the solution feasibility and solution quality.

2.3 Systematic DNN Weight Pruning

Instead of direct application of ADMM, we develop an integrated framework of ADMM regularization and masked mapping and retraining as showed in Algorithm 1. We guarantee solution feasibility (satisfying all constraints) and provide high quality (maintaining test accuracy).

The ADMM regularization starts from a DNN model without compression. By incorporating auxiliary variables \( Z_i \)'s, and dual variables \( U_i \)'s, we decompose (4) into two subproblems, and iteratively solves them until convergence. The first subproblem is

\[
\begin{align*}
\text{minimize} & \quad f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N}) + \sum_{i=1}^{N} \beta_i \left( \| W_i - Z_i \|_2^2 + \| U_i \|_2^2 \right).
\end{align*}
\]

(5)

The first term in (5) is the differentiable (non-convex) loss function of the DNN, while the other quadratic terms are differentiable and convex. As a result, this subproblem can be solved by stochastic
gradient descent similar to the one that would be used to train the original DNN.

The second subproblem is

$$\min_{\{Z_i\}} \sum_{i=1}^{N} g_i(Z_i) + \sum_{i=1}^{N} \frac{\rho_i}{2} \|W_i^{k+1} - Z_i + U_i^{k+1}\|^2. \quad (6)$$

The optimal, analytical solution is the Euclidean projection of $W_i^{k+1} + U_i^{k+1}$ onto the set $S_i$. Since $a_i$ is the desired number of weights after pruning in the $i$-th layer, we can prove that the Euclidean projection results in keeping $a_i$ elements in $W_i^{k+1} + U_i^{k+1}$ with the largest magnitudes and setting the remaining weights to zeros. After both subproblems solved, we update the dual variables $U_i$ as Eqn. (7) and complete one iteration in ADMM regularization.

$$U_i^{k+1} = U_i^k + W_i^{k+1} - Z_i^{k+1} \quad (7)$$

**Masked Mapping and Retraining:** We extend the formulation in [27] by introducing masked mapping and retraining step. After ADMM regularization, we obtain intermediate $W_i$ solutions. In this step, we first perform the said Euclidean projection (mapping) to guarantee that at most $a_i$ weights in each layer are non-zero. Next, we mask the zero weights and retrain the DNN with non-zero weights using training sets (while keeping the masked weights 0). In this way test accuracy can be partially restored.

Algorithm 1 shows the pseudo-codes of the proposed weight pruning training algorithm, in which ADMM ITERATION is normally set as $\frac{1}{10}$ of MAX_ITERATION_FOR_SGD.

```
1 Initialize training hyperparameters;
2 for CURRENT_ITERATION < MAX_ITERATION_FOR_SGD do
3     Solve(Eqn. (5));
4     if CURRENT_ITERATION % ADMM_ITERATION == 0 then
5         Solve(Eqn. (6));
6         Solve dual update according to Eqn. (7);
7     end
8 end
9 Masked mapping;
10 Retrain the pruned model;
```

**Feasibility and Solution Quality:** One can observe that constraints on weight pruning are satisfied through the mapping step and that the retraining process restores the accuracy loss of mapping. ADMM regularization acts as a smart, adaptive DNN regularization (see Eqn. (5)), where the regularization targets are dynamically updated in each iteration by solving subproblem 2 (optimally and analytically). This is one key reason that this method outperforms many prior works on DNN weight pruning based on fixed regularization [26], where regularization targets are not updated.

**Sample Results on Representative DNNs:** Sparse matrices are employed for representing the pruned weights due to the reduced space complexity (by storing only the non-zero entries and index to the next non-zero entry) and associated computation savings. We have performed testing on representative DNNs, LeNet-5 [18] for MNIST dataset, and AlexNet [17] and ResNet-50 [11] for ImageNet dataset. As shown in Table 1, we achieve 167x reduction in number of weights in LeNet-5, 31x in AlexNet, and 15x in ResNet-50, with (almost) no accuracy loss. These results consistently outperform prior arts especially on ResNet, which is difficult for pruning in prior work.

<table>
<thead>
<tr>
<th>Network</th>
<th>Method</th>
<th>Accuracy Loss</th>
<th>Weights Pruning Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet-5</td>
<td>Network Pruning [10]</td>
<td>0.0%</td>
<td>34.5K 12.5x</td>
</tr>
<tr>
<td></td>
<td>Direct ADMM [27]</td>
<td>0.0%</td>
<td>6.05K 71.2x</td>
</tr>
<tr>
<td></td>
<td>Our Proposed Method</td>
<td>0.2%</td>
<td>2.58K 167x</td>
</tr>
<tr>
<td>AlexNet</td>
<td>NEST [4]</td>
<td>0.0%</td>
<td>3.9M 15.7x</td>
</tr>
<tr>
<td></td>
<td>Direct ADMM [27]</td>
<td>0.0%</td>
<td>2.9M 21x</td>
</tr>
<tr>
<td></td>
<td>Our Proposed Method</td>
<td>0.1%</td>
<td>1.9M 31x</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Fine-grained Pruning [22]</td>
<td>0.1%</td>
<td>12.6M 2.6x</td>
</tr>
<tr>
<td></td>
<td>Our Proposed Method</td>
<td>0.4%</td>
<td>2.18M 15x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8%</td>
<td>1.82M 18x</td>
</tr>
</tbody>
</table>

**2.4 Incorporating Structures in Weight Pruning**

As discussed before, the DNN after weight pruning is an irregular, sparse neural network, and sparse matrices with indices are utilized for weight storage. One clear disadvantage is the limitation on parallelism degree and therefore degradation in hardware performance as also observed in [26]. Prior work (e.g., [26]) incorporates regularity or “structures” in DNN weight pruning in order to solve this problem, but lacks a systematic approach to achieve this goal.

We make an observation that structured pruning is compatible with the ADMM-based weight pruning framework, and therefore can be solved systematically. We use CONV layers (the most computationally intensive in current DNNs [11, 17]) as an illustrative example while FC layers can be treated similarly. There are three types of structured sparsities, filter-wise, channel-wise, and shape-wise sparsities as shown in Figure 1.
CONV operations in DNNs are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors to matrices [3], named general matrix multiplication or GEMM, as shown in Figure 2, in order to facilitate implementation from mobile devices to GPUs. Filter-wise pruning corresponds to reducing number of rows, while channel-wise and filter shape-wise prunings correspond to reducing the number of columns. As a result, a combination of the said three structured sparsities will reduce the dimension in GEMM while maintaining a full matrix, thereby facilitating acceleration in mobile/hardware platforms.

All the above three structured pruning scenarios can be incorporated into the ADMM regularization framework. For filter-wise sparsity as an example, the constraint set $S_i$ will indicate that the number of nonzero filters in $W_i$ is less than a predefined value. For channel-wise sparsity, constraint set $S_i$ will indicate that the number of nonzero channels in $W_i$ is less than a predefined value. In this way structured pruning will replace the second subproblem in (6), and the optimal Euclidean mapping will be derived accordingly. The first subproblem in (5) and the masking mapping/retraining step will maintain the same form (and solution method).

3 MOBILE IMPLEMENTATION/ACCELERATION OF PRUNED DNNs

In this section, we describe mobile implementation/acceleration of the inference phase of pruned DNNs. As shown in Figure 3, there are four high-level modules on Android-based platforms: model constructor, parameters loader, dataset loader and inference engine. In the first module, the network architecture will be constructed and through parameters import, pre-trained weights and bias will be loaded. In the third module, the test data are loaded through camera or files and we perform inference in the last module through C++ interface.

Since GEMM is utilized for CONV layer computation in mobile systems, and CONV layer is the most computationally intensive layer, we adopt sparse matrices in GEMM based on the weight pruning results. Specifically, the computational complexity is reduced from $O(N^3)$ to $O(K \cdot N)$, where $N$ is weight matrix dimension, and $K$ is the number of non-zero elements in the matrix. We implement sparse matrix computation in a bottom-up manner from C++ array template, instead of existing libraries (e.g., OpenCV [2], Eigen [7]). This is because of the limited scalability of the mobile version of such libraries to support large-scale DNNs.

The DNN inference acceleration algorithm is shown in Algorithm 2, focusing on GEMM acceleration for a specific layer. In each layer, the input matrix and weight matrix in GEMM are denoted by $X$ and $W$, respectively. The bias vector is $b$. To accelerate the computation, the structure of the weight matrix $W$ is transformed to the sparse matrix structure denoted as $W'$, in which Dictionary of Keys is adopted to efficiently construct and represent. Step 5 to 12 show the details of the general multiplication of the dense input matrix $X$ and the sparse weight matrix $W'$. The dimensionality of $W'$ is $K \times 3$.

### Algorithm 2: Pruned Model Inference Algorithm on Mobile Devices (focusing on GEMM for one layer)

1. **Input:** $m \times n$ matrix $X$
2. **Parameters:** $W$, $b$
3. **Output:** matrix $Y$

   /* convert dense matrix to sparse matrix */

   $W' \leftarrow W$;

   /* sparse matrix * dense matrix operation */

   for index $\leftarrow 0$ until $K$
   
   row $\leftarrow W'_{index,0}$;
   col $\leftarrow W'_{index,1}$;
   value $\leftarrow W'_{index,2}$;
   for $j \leftarrow 0$ until $n$
   
   $Y_{row,j} \leftarrow Y_{row,j} + value \cdot X_{col,j}$;
   end
   end

12. $Y \leftarrow Y + b$;
13. Return $Y$;

4 EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, we perform evaluation on two medical related applications, starting from DNN construction, systematic weight pruning using ADMM, and finally implementation of DNN inference on
multiple mobile devices to evaluate the applicability of the inference phase. Table 2 summarizes the specifications of test mobile platforms.

The performance evaluation includes two aspects: (a) storage reduction due to the reduced DNN model size, (b) inference acceleration (running time reduction). The largest portion of inference time is from GEMM operation, which will benefit from sparse matrices. We compare the run-time cost of GEMM as well as the overall run-time.

### 4.1 Case I: Facial Emotion Recognition

Facial emotion recognition is utilized in many fields such as medical, entertainment and security. For this application, FER-2013 face database [6] is used to train and test the DNN model. The dataset comprises a total of 35,887 pre-cropped, 48-by-48-pixel grayscale images of faces each labeled with one of the 7 emotion classes: anger, disgust, fear, happiness, sadness, surprise, and neutral. Due to the small portion of the disgust class, the dataset is merged into six classes including angry, fear, happy, sad, surprise, and neutral [13].

A typical DNN is constructed, comprising 9 CONV layers with one max-pooling after every three CONV layers. There are 32, 64, and 218 filters in these three CONV-layer groups, respectively. In addition, 2 FC layers are constructed followed by a softmax layer in the end. Table 3 shows the weight pruning result. Our pruning algorithm can achieve a high compression ratio by 4.6x without accuracy loss.

### 4.2 Case II: Eye Tracking

Eye tracking is one widely used application in many areas such as human-computer interaction, medical diagnoses, psychological studies and computer vision. To implement the weight pruned eye tracking model for embedded systems, we use GazeCapture, a large-scale mobile eye tracking dataset, containing data from over 1,450 people with almost 2.5M frames [16].

The eye tracking DNN model takes as input the detected and cropped portions of the original frame, including left eye, right eye, and face images (all of size 224 x 224). Additionally, the face grid is considered as another input, with a binary mask to indicate the location and size of the head within the frame (of size 25 x 25). The output is the distance, in centimeters, from the camera. The whole model consists of 13 CONV layers and 7 FC layers. Compared with typical DNN, the eye tracking network has a more complicated architecture. It consists of three small typical DNNs taking inputs from the right eye, left eye and face respectively. The outputs get concatenated and go through 7 FC layers along with the input from face grid. The details of model structure are demonstrated in [16]. The network architecture also shows a big impact on the overall real-time speedup.

After applying our weight pruning method, the weight reduction result is shown in Table 5. The overall pruning ratio can achieve 36x. The total number of weights is reduced from 718K to 19K. The performance on three mobile devices is demonstrated in the following Table 6. The pruned model achieves up to 13x speedup compared with the non-pruned model, especially, the GEMM computing time gets accelerated up to 28x.
Table 5: Weight Pruning Result of Eye Tracking Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Weights</th>
<th>Weights after prune</th>
<th>Matrices sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv-e1</td>
<td>4875</td>
<td>3413</td>
<td>30%</td>
</tr>
<tr>
<td>conv-e2</td>
<td>102400</td>
<td>512</td>
<td>99.5%</td>
</tr>
<tr>
<td>conv-e3</td>
<td>73728</td>
<td>369</td>
<td>99.5%</td>
</tr>
<tr>
<td>conv-e4</td>
<td>8192</td>
<td>1229</td>
<td>85%</td>
</tr>
<tr>
<td>conv-f1</td>
<td>4875</td>
<td>3413</td>
<td>30%</td>
</tr>
<tr>
<td>conv-f2</td>
<td>102400</td>
<td>512</td>
<td>99.5%</td>
</tr>
<tr>
<td>conv-f3</td>
<td>73728</td>
<td>369</td>
<td>99.5%</td>
</tr>
<tr>
<td>conv-f4</td>
<td>8192</td>
<td>1229</td>
<td>85%</td>
</tr>
<tr>
<td>fc-e1</td>
<td>65536</td>
<td>3933</td>
<td>94%</td>
</tr>
<tr>
<td>fc-f1</td>
<td>32768</td>
<td>656</td>
<td>98%</td>
</tr>
<tr>
<td>fc-f2</td>
<td>8192</td>
<td>1475</td>
<td>82%</td>
</tr>
<tr>
<td>fc-fg1</td>
<td>160000</td>
<td>800</td>
<td>99.5%</td>
</tr>
<tr>
<td>fc-fg2</td>
<td>32768</td>
<td>984</td>
<td>97%</td>
</tr>
<tr>
<td>fc1</td>
<td>40960</td>
<td>615</td>
<td>98.5%</td>
</tr>
<tr>
<td>fc2</td>
<td>256</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>718870</td>
<td>19765</td>
<td>97.25%</td>
</tr>
</tbody>
</table>

Table 6: Performance of Eye Tracking Model on Mobile Devices

<table>
<thead>
<tr>
<th></th>
<th>Model</th>
<th>Pruned model</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honor 6X Overall</td>
<td>0.563s</td>
<td>0.043s</td>
<td>13.2x</td>
</tr>
<tr>
<td>GEMM</td>
<td>0.541s</td>
<td>0.019s</td>
<td>28.8x</td>
</tr>
<tr>
<td>Nexus 5X Overall</td>
<td>0.070s</td>
<td>0.009s</td>
<td>7.5x</td>
</tr>
<tr>
<td>GEMM</td>
<td>0.065s</td>
<td>0.002s</td>
<td>28x</td>
</tr>
<tr>
<td>Honor 10 Overall</td>
<td>0.142s</td>
<td>0.018s</td>
<td>7.7x</td>
</tr>
<tr>
<td>GEMM</td>
<td>0.131s</td>
<td>0.007s</td>
<td>17.4x</td>
</tr>
</tbody>
</table>

5 CONCLUSION

In this paper, we extend the prior work on systematic DNN weight pruning using ADMM. We integrate ADMM regularization with masked mapping/retraining, thereby guaranteeing solution feasibility and providing high solution quality. We develop two new applications: facial emotion detection and eye tracking, and propose a top-down framework of DNN training, model compression, and acceleration in mobile devices. The proposed method shows significant storage/memory reduction and speedup measured on mobile devices.

ACKNOWLEDGMENTS

This work is funded by the National Science Foundation Awards CNS-1704662 and CNS-1739748.

REFERENCES