Pruning Deep Neural Networks with ℓ_0-constrained Optimization

Dzung T. Phan, Lam M. Nguyen, Nam H. Nguyen, Jayant R. Kalagnanam
IBM Research, Thomas J. Watson Research Center, Yorktown Heights, NY, 10598, USA
phandu@us.ibm.com, LamNguyen.MLTD@ibm.com, nnguyen@us.ibm.com, jayant@us.ibm.com

***Abstract*—Deep neural networks (DNNs) give state-of-the-art accuracy in many tasks, but they can require large amounts of memory storage, energy consumption, and long inference times. Modern DNNs can have hundreds of million parameters, which make it difficult for DNNs to be deployed in some applications with low-resource environments. Pruning redundant connections without sacrificing accuracy is one of popular approaches to overcome these limitations. We propose two ℓ_0-constrained optimization models for pruning deep neural networks layer-by-layer. The first model is devoted to a general activation function, while the second one is specifically for a ReLU. We introduce an efficient cutting plane algorithm to solve the latter to optimality. Our experiments show that the proposed approach achieves competitive compression rates over several state-of-the-art baseline methods.**

I. Introduction

In recent years, deep neural networks (DNNs) have shown significant accuracy improvements in a wide variety of applications including pattern recognition, image classification, and speech recognition. With modern large networks consisting of hundreds of millions parameters [1], DNNs can require large amounts of storage, massive computing power, and long inference times. Hence, deploying very large learned networks to resource-limited devices can be impractical due to their extensively computational and storage requirements. Furthermore, it has been shown that neural networks can be heavily over-parametrized [2] and are prone to over-fitting [3]. Recent works show that their parameters can be reduced by more than 90% without accuracy drop [4], [5]. It is a vitally important task to prune deep neural networks without a considerable loss in accuracy so that DNNs can be utilized for real-world applications including low-resource environments such as mobile devices, IoT edge devices, and real-time prediction. The network pruning problem has received dramatically increasing attention in the deep learning community [6].

There are two main lines of neural network pruning research by removing parameters from an existing network: regularization [7] and parameter pruning [4], [8]. We particularly focus on layer-wise pruning methods, which is a powerful technique to compress neural networks. It excerpts the intermediate outputs at each layer of the well-trained network, and enforces consistency between the resulting response and the pre-trained network response. One of the first layer-wise methods is Net-Trim [8], which proposes a convex quadratically constrained ℓ_1 program. Following this idea, layer-wise optimal brain surgeon method (L-OBS) [9] eliminates parameters based on second-order derivatives of a layer-wise error function.

A common approach using the non-convex ℓ_0 norm is to prune the neural network during training by encouraging redundant weights to become zero in the training problem [7], [10], [11]. Our work is the first to apply the ℓ_0 norm technique for a layer-wise pruning method. Ideally, the pruning problem should be formulated as a single optimization problem where we optimize a training loss and enforce sparsity simultaneously. However, getting a high-quality solution with two goals: a small training loss and a very sparse vector solution is changeling due to non-convexity from both the objective and sparsity constraint.

In this paper, we propose an ℓ_0-constrained optimization framework for layer-wise pruning neural networks. For a general activation function, we formulate the weight pruning task as the well-known best subset selection problem. For layers with a rectified linear unit (ReLU), we introduce an ℓ_0-constrained quadratic programming. We provide a cutting plane method to efficiently solve the structured quadratic problem to optimality with theoretical convergence guarantee. Compared to Net-Trim and L-OBS, for the same sparsity level, the proposed model guarantees a smaller reconstructed error bound for each layer. Our numerical experiments demonstrate a state-of-the-art compression rate for both fully-connected and convolutional neural networks.

II. Related Work

In a seminal work, the authors in [12] developed a trimming technique, called optimal brain damage (OBD), to remove parameters based on the minimal change in training error. The second derivative of the error function is approximated by a diagonal matrix and calculated by back-propagation. The optimal brain surgeon (OBS) method [13] makes no restrictive assumption on the Hessian, and is able to prune of more weights than OBD. Due to Hessian inverse computation, these methods are not scalable for large deep networks. Layer-wise OBS [9] mitigates the issue by considering a simpler layer-wise error function related to the input of activation, and pruning each layer of a trained network independently. The layer-wise pruning idea is originally proposed in [8], where a convex ℓ_1 norm model with performance guarantee, specially designed for ReLU activation, is introduced.

Magnitude-based methods delete weights with the smallest magnitudes from a threshold. Han *et al.* [4] propose to include a retrain step in the framework. Li *et al.* [14] calculate the layer specific thresholds by solving a constrained optimization via a derivative-free optimization. The authors in [15] use the idea of soft weight-sharing to compress the network. Dynamic network surgery [16] involves two stages: pruning and splicing. Variational dropout techniques have also been utilized to sparsify neural networks [17].

Another class of pruning methods are based on enforcing sparsity during training phase such as regularization methods: ℓ_1 norm [18], ℓ_0 norm [7], and a combination of (ℓ_1, ℓ_2)-norm [19]. Some ℓ_0-constrained models have been proposed including training the constrained model by augmented-Lagrangian [10] and connection sensitivity [11].

III. General Activation Functions

One of popular frameworks for compressing a neural network consists of three steps: pre-training the network, removing unimportant components, and re-training the remaining structure [4], [8], [5]. In this paper, we introduce a new layer-wise pruning method for the second step.

A. Overview of layer-wise pruning

Consider a fully-connected feedforward network with L layers for supervised learning. Given a training data set $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$, we train the neural network using the input $\mathbf{x}_i^{(0)} = \mathbf{x}_i \in \mathbb{R}^{m_0}$ and minimize the discrepancy between the true label $\mathbf{y}_i \in \mathbb{R}^{m_L}$ and the output $\mathbf{x}_i^{(L)}$ at the final layer. The network can be represented as a nested function, where the output at the ℓ-th layer is $\mathbf{x}^{(\ell)} = \sigma(\mathbf{W}_\ell^\mathsf{T}\mathbf{x}^{(\ell-1)} + \mathbf{b}^{(\ell-1)})$, $\ell = 1, \ldots, L$, where σ is an activation function, $\mathbf{W}_\ell = [\mathbf{w}_1, \ldots, \mathbf{w}_{m_\ell}] \in \mathbb{R}^{m_{\ell-1} \times m_\ell}$ is a weight matrix, and $\mathbf{b}^{(\ell-1)} \in \mathbb{R}^{m_\ell}$ is the bias. By stacking an additional row and a constant term to $\mathbf{W}_\ell$ and $\mathbf{x}^{(\ell-1)}$ respectively, we can recast the network as $\mathbf{X}^{(\ell)} = \sigma(\mathbf{Z}^{(\ell)})$, $\mathbf{Z}^{(\ell)} = \mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}$, $\ell = 1, \ldots, L$, where $\mathbf{X}^{(\ell)} = [\mathbf{x}_1^{(\ell)}, \ldots, \mathbf{x}_n^{(\ell)}] \in \mathbb{R}^{m_\ell \times n}$ and $\mathbf{Z}^{(\ell)} \in \mathbb{R}^{m_\ell \times n}$. Here, m_ℓ is the number of neurons at the ℓ-th layer.

Suppose that a pre-trained network parameterized by the set of weights $\{\mathbf{W}_\ell\}_{\ell=1}^L$ is given. Our goal of pruning the network is to force many elements of $\mathbf{W}_\ell$ to be zero and adjust the values of non-zero entries so that the sparsified network has a comparable predictive capability. By using the original training data $\mathbf{X}$ and pre-trained weights $\{\mathbf{W}_\ell\}_{\ell=1}^L$, we wish to find the set of sparsest weights $\hat{\mathbf{W}}_\ell$ such that

$$\sigma(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}) \approx \sigma(\mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}), \quad \text{while } \|\hat{\mathbf{W}}_\ell\|_0 \ll \|\mathbf{W}_\ell\|_0, \tag{1}$$

for all $\ell = 1, \ldots, L$, where $\|.\|_0$ is the ℓ_0 norm, counting the number of nonzero entries. However, solving this problem directly is very challenging because of the non-linearity of σ, and sparsity requirement. We can overcome the issue by looking for an approximation problem for (1). When the activation function is ReLU, the authors in [8] propose a convex ℓ_1-norm problem for approximately solving (1) by imposing a bound on the reconstructed error. They relax the nonconvex constraint by using a number of quadratic constraints and linear inequalities. We propose a better approximation for model (1), i.e., resulting a smaller reconstructed error $\|\sigma(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}) - \sigma(\mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})\|_F$. Moreover, it can deal with a general activation function.

B. An ℓ_0 optimization model for pruning

Assume that σ is Lipschitz continuous; that is, there exists $C > 0$ such that $|\sigma(x) - \sigma(y)| \leq C|x - y|$, $\forall x, y \in \mathbb{R}$. Most activation functions such as ReLU $\sigma(x) = \max(0, x)$, sigmoid $\sigma(x) = 1/(1 + e^{-x})$, hyperbolic tangent $\sigma(x) = \tanh(x)$, and $\sigma(x) = \arctan(x)$ satisfy the assumption with $C = 1$. For these functions, we have

$$\|\sigma(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}) - \sigma(\mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})\|_F \leq \|\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)} - \mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}\|_F.$$

With the Lipschitz continuity assumption, we can achieve the goal in Eq. (1) by considering the proximity requirement for the input of the activation function

$$\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)} \approx \mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}, \text{ while } \|\hat{\mathbf{W}}_\ell\|_0 \ll \|\mathbf{W}_\ell\|_0 \tag{2}$$

for all $\ell = 1, \ldots, L$. More precisely, we propose to solve the following ℓ_0-constrained optimization problem for each layer ℓ to get a sparse weight matrix $\hat{\mathbf{W}}_\ell$

$$\min_{\mathbf{W}} \quad \tfrac{1}{2}\|\mathbf{W}^\mathsf{T}\mathbf{X}^{(\ell-1)} - \mathbf{Z}^{(\ell)}\|_F^2 \quad \text{s.t.} \|\mathbf{W}\|_0 \leq \kappa, \tag{3}$$

where κ is the maximally allowable number of nonzeros. Note that $\mathbf{X}^{(\ell-1)}$ and $\mathbf{Z}^{(\ell)}$ come from the pre-trained network. The model (3) is the well-known best subset selection problem. It can be efficiently solved by some exact methods [20] or local methods such as stochastic gradient descent.

It has been known that we can vectorize filters of each channel in convolutional neural networks and treat them as fully-connected feedforward layers, see for example, Section 3.4.2 of [9]. Hence, we have only presented our techniques for feedforward networks, but will also show the numerical results for convolutional networks.

IV. ReLU Activation Function

We notice that ReLU is one of the most widely-used activation functions, thus we are particularly interested in handling this function. When ReLU is employed as the activation for each layer, we can further tighten the model (3). That is, for the same layer-wise error $\|\sigma(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)}) - \sigma(\mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})\|_F$, we could get a sparser solution by using another model proposed in this section rather than the one obtained from (3).

We note that $\sigma(Z_{i,j}^{(\ell)})$ is non-negative. In some applications, many elements $\sigma(Z_{i,j}^{(\ell)})$ are zero. If $(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})_{i,j} \leq$

0 and $Z_{i,j}^{(\ell)} \leq 0$, then $\sigma((\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})_{i,j}) = \sigma(Z_{i,j}^{(\ell)})$. It suggests that we should not impose $(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})_{i,j}$ close to $Z_{i,j}^{(\ell)}$ for this case. For notational simplicity, we will omit the index ℓ in some cases such as $\mathbf{X}^{(\ell-1)} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ if there is no confusion. Define two index sets:

$$\mathcal{I} = \{(i,j) : Z_{i,j}^{(\ell)} > 0\}, \quad \mathcal{A} = \{(i,j) : Z_{i,j}^{(\ell)} \leq 0\}.$$

We propose to solve the following problem (4) when σ is the ReLU function:

$$\begin{array}{ll} \min\limits_{\mathbf{W},\mathbf{y}} & F(\mathbf{W},\mathbf{y}) = \|[\mathbf{W}^\mathsf{T}\mathbf{X}^{(\ell-1)} - \mathbf{Z}^{(\ell)}]_\mathcal{I}\|^2 + \sum\limits_{(i,j)\in\mathcal{A}} y_{i,j}^2 \\ \text{s.t.} & \mathbf{w}_i^\mathsf{T}\mathbf{x}_j \leq y_{i,j}, \ \ \forall (i,j) \in \mathcal{A} \\ & y_{i,j} \geq 0, \ \ \forall (i,j) \in \mathcal{A} \\ & \|\mathbf{W}\|_0 \leq \kappa, \end{array} \tag{4}$$

where $[\mathbf{X}]_\mathcal{I}$ is a vectorized form whose entries are $X_{i,j}$'s for all $(i,j) \in \mathcal{I}$. The inputs for activation functions are encouraged to be close to each other only over the set $\mathcal{I}$. We allow some entries $(\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})_{i,j}$ in set $\mathcal{A}$ associated with $Z_{i,j}^{(\ell)} = (\mathbf{W}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})_{i,j} \leq 0$ to become strictly positive, but incurring a cost $y_{i,j}^2 = (\hat{\mathbf{W}}_\ell^\mathsf{T}\mathbf{X}^{(\ell-1)})_{i,j}^2$.

Remark 1. *The value $y_{i,j}$ can act as the hinge loss for $\mathbf{w}_i^\mathsf{T}\mathbf{x}_j$. We can obtain an equivalent formulation for* (4) *by directly considering a squared hinge loss on $\mathbf{w}_i^\mathsf{T}\mathbf{x}_j$ values*

$$\begin{array}{ll} \min\limits_{\mathbf{W}} & \|[\mathbf{W}^\mathsf{T}\mathbf{X}^{(\ell-1)} - \mathbf{Z}^{(\ell)}]_\mathcal{I}\|^2 + \sum\limits_{(i,j)\in\mathcal{A}} \max(\mathbf{w}_i^\mathsf{T}\mathbf{x}_j, 0)^2 \\ \textit{s.t.} & \|\mathbf{W}\|_0 \leq \kappa. \end{array} \tag{5}$$

However, the non-differentiability of the the hinge loss makes the optimization problem (5) *difficult to solve. Existing algorithms such as sub-gradient descent algorithms and SGD are often slow. The advantage of our model* (4) *is the smoothness property and the decomposable structure. Our setting is similar to the well-known support vector machine (SVM) models [21]. The non-smooth hinge loss version of SVM is neat, but challenging to solve it. Most practical algorithms for SVM are devoted to the other smooth SVM models. Problem* (4) *plays the same role as the primal version of SVM. It is also a smooth quadratic program, our next outer-approximation algorithm can efficiently exploit its special structure.*

Our model (4) is tighter than existing layer-wise models. Particularly, for the same number of nonzero weights, the reconstructed error bound for each layer of the pruned network obtained from (4) is less than or equal to those given by Net-Trim and L-OBS. Formally, we have the following theorem.

Theorem 1. *Consider a normalized pre-trained network $(\{\mathbf{W}_\ell\}_{\ell=1}^L; \mathbf{X})$ such that $\|\mathbf{W}_\ell\|_1 = 1$ for $\ell = 1, \ldots, L$. Suppose $(\{\bar{\mathbf{W}}_\ell\}_{\ell=1}^L; \mathbf{X})$ is the pruned network provided by Net-Trim, where the error in each layer ℓ is $\bar{\epsilon}_\ell$ (defined in Eq.* (3) *of [8]) and $\sum_{\ell=1}^L \|\bar{\mathbf{W}}_\ell\|_0 = \bar{\kappa}$. Then there exists a network $(\{\hat{\mathbf{W}}_\ell\}_{\ell=1}^L; \mathbf{X})$ pruned by our model Eq.* (4) *with non-negative values $0 \leq \hat{\epsilon}_\ell \leq \bar{\epsilon}_\ell$ such that $\sum_{\ell=1}^L \|\hat{\mathbf{W}}_\ell\|_0 \leq \bar{\kappa}$ and the reconstructed error obeys*

$$\|\hat{\mathbf{X}}^{(\ell)} - \mathbf{X}^{(\ell)}\|_F \leq \sum_{j=1}^{\ell} \hat{\epsilon}_\ell, \ \forall \ell = 1, \ldots, L.$$

Here $\hat{\mathbf{X}}^{(\ell)} = ReLU(\hat{\mathbf{W}}_\ell^\mathsf{T}\hat{\mathbf{X}}^{(\ell-1)})$ and $\hat{\mathbf{X}}^{(0)} = \mathbf{X}$. Furthermore, it holds that $\sum_{j=1}^{\ell} \hat{\epsilon}_\ell \leq \sum_{j=1}^{\ell} \bar{\epsilon}_\ell$ and strict inequality occurs if there exists $y_{i,j} > 0$ for some $(i,j) \in \mathcal{A}$ in (4)*.*

Note that the reconstructed error in Net-Trim obeys $\|\bar{\mathbf{X}}^{(\ell)} - \mathbf{X}^{(\ell)}\|_F \leq \sum_{j=1}^{\ell} \bar{\epsilon}_\ell$ and $\bar{\epsilon}_\ell \geq \hat{\epsilon}_\ell$. Hence, our model (4) gives a better upper bound for the reconstructed error. The claim is still true when we compare our model (3) with L-OBS since L-OBS uses a heuristic block coordinate search for reducing the error. L-OBS can reach a sub-optimal solution. As a consequense, we can apply performance guarantee results from Net-Trim and L-OBS to our models (3) and (4).

A. Cutting Plane Algorithm

The problem (4) is a large-scale constrained optimization problem, existing methods such as interior-point methods and augmented Lagrangian method find it hard to solve. We introduce an efficient algorithm for solving (4), that can exploit its special structure. Observe that $|\mathcal{A}|$ is often very large, e.g. $\mathcal{O}(nm_\ell)$, we need to deal with a large number of constraints (i.e., $2|\mathcal{A}|$) as well as additional variables $y_{i,j}$ (i.e., $|\mathcal{A}|$) for problem (4). As an example, the pre-trained network for LeNet-300-100 applied to MNIST [22] has the following values for $|\mathcal{I}|$ and $|\mathcal{A}|$ for each layer as shown in Table I.

Table I: Cardinality for index sets $\mathcal{I}$ and $\mathcal{A}$

Layer	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{A}\rvert/(\lvert\mathcal{A}\rvert + \lvert\mathcal{I}\rvert)$ (%)
1	4140K	13680K	76
2	2340K	3660K	61

We could expect that at the optimal solution of (4), only a very small portion of entries $y_{i,j}$ is strictly positive and most of them are zero. That is, to maintain a small reconstruction error with a sparse solution, a very few entries $\mathbf{w}_i^\mathsf{T}\mathbf{x}_j, (i,j) \in \mathcal{A}$, may need to be changed to positive. We will predict these nonzero elements and gradually add them into our model.

We recast (4) as a two-stage problem, whose objective is represented as a function of $\mathbf{W}$:

$$\min_{\|\mathbf{W}\|_0 \leq \kappa} R(\mathbf{W}) = \|\Big[\mathbf{W}^\mathsf{T}\mathbf{X}^{(\ell-1)} - \mathbf{Z}^{(\ell)}\Big]_\mathcal{I}\|^2 + \sum_{i=1}^{m_\ell} \phi(\mathbf{w}_i) \tag{6}$$

$$\text{where } \phi(\mathbf{w}_i) = \begin{array}{ll} \min\limits_{\mathbf{y}} & \sum_{(i,j)\in\mathcal{A}} y_{i,j}^2 \\ \text{s.t.} & \mathbf{w}_i^\mathsf{T}\mathbf{x}_j \leq y_{i,j}, \ \ \forall (i,j) \in \mathcal{A} \\ & y_{i,j} \geq 0, \ \ \forall (i,j) \in \mathcal{A}. \end{array} \tag{7}$$

Our key idea is to begin with a starting point $\mathbf{W}^{(0)}$ and successively minimize an approximation of $R(\mathbf{W})$ with a sequence of first-order surrogate functions for $\phi(\mathbf{w}_i)$. We take advantage of the fact that most of $\{\phi(\mathbf{w}_i)\}_{i=1}^{m_\ell}$ take zero value at the optimal solution.

We can show that $\phi(\mathbf{w}_i)$ is the convex function and a subgradient can be evaluated quickly. We will approximate $\phi(\mathbf{w}_i)$ from below by a collection of linear inequalities.

Theorem 2. *$\phi(\mathbf{w}_i)$ is convex and its subgradient is*

$$\partial\phi(\mathbf{w}_i) = \begin{cases} 2\sum\limits_{(i,j)\in\hat{\mathcal{A}}} (\mathbf{w}_i^\mathsf{T}\mathbf{x}_j)\mathbf{x}_j, & \text{if } |\hat{\mathcal{A}}| > 0 \\ \mathbf{0}, & \text{otherwise}, \end{cases}$$

where $\hat{\mathcal{A}} = \{(i,j) \in \mathcal{A} : \mathbf{w}_i^\mathsf{T}\mathbf{x}_j > 0\}$.

At the k-th iteration of our algorithm described below, for each neuron $i \in \{1,\dots,m_\ell\}$, piece-wise linear lower approximations for $\phi(\mathbf{w}_i)$ can be generated as

$$\phi(\mathbf{w}_i^{(t)}) + \partial\phi(\mathbf{w}_i^{(t)})^\mathsf{T}(\mathbf{w}_i - \mathbf{w}_i^{(t)}) \le u_i$$

for every $t = 1,\dots,k$, where u_i is a surrogate of $\phi(\mathbf{w}_i)$. It is equivalent to

$$\big(2\sum_{(i,j)\in\hat{\mathcal{A}}} (\mathbf{w}_i^{(t)\mathsf{T}}\mathbf{x}_j)\mathbf{x}_j\big)^\mathsf{T}\mathbf{w}_i - \sum_{(i,j)\in\hat{\mathcal{A}}} (\mathbf{w}_i^{(t)\mathsf{T}}\mathbf{x}_j)^2 \le u_i,$$

in short, $\mathbf{a}_i^{(t)\mathsf{T}}\mathbf{w}_i + b_i^{(t)} \le u_i$, $\forall t = 1,\dots,k$. For any $k > 0$, let us define $\mathcal{C}_k = \{(i,t) : 1 \le t \le k, 1 \le i \le m_\ell\}$. An approximation problem for (6) is

$$\begin{array}{ll} \min\limits_{\mathbf{W},\mathbf{u}} & E(\mathbf{W},\mathbf{u}) = \| \left[\mathbf{W}^\mathsf{T}\mathbf{X} - \mathbf{Z}\right]_\mathcal{I} \|^2 + \sum_{i=1}^{m_\ell} u_i \\ \text{s.t.} & \|\mathbf{W}\|_0 \le \kappa \\ & \mathbf{a}_i^{(t)\mathsf{T}}\mathbf{w}_i + b_i^{(t)} \le u_i, \forall (i,t) \in \mathcal{C}_k \\ & u_i \ge 0, \forall i = 1,\dots,m_\ell. \end{array} \tag{8}$$

Problem (8) is much simpler to be solved than (4) when $|\mathcal{C}_k| \ll |\mathcal{A}|$. For our network pruning application with negligible accuracy loss requirement, we expect that a very few elements u_i are strictly positive at the optimal solution of (6). As a result, it leads to a small cardinality $|\mathcal{C}_k|$. (It will be numerically confirmed later in the experimental section V-B.) As shown in Table I, the value $|\mathcal{A}|$ is often very large. Our proposed cutting plane method for solving (4), described in Algorithm 1, will make use of the property $|\mathcal{C}_k| \ll |\mathcal{A}|$.

We can prove that the algorithm converges to an optimal solution of (4).

Theorem 3. *If Algorithm 1 finitely terminates at the k-th iteration, then $(\mathbf{W}^{(k)}, \sqrt{\mathbf{u}^{(k)}})$ is an optimal solution of (4). Otherwise, if we assume that $\mathbf{W}^{(k)}$ is uniformly bounded, then the sequence of iterates $\{(\mathbf{W}^{(k)}, \mathbf{u}^{(k)})\}_{k\ge1}$ satisfies*

$$\lim_{k\to\infty} F(\mathbf{W}^{(k)}, \sqrt{\mathbf{u}^{(k)}}) = F(\mathbf{W}^*, \mathbf{y}^*),$$

where $(\mathbf{W}^*, \mathbf{y}^*)$ is an optimal solution of (4).

Algorithm 1: Cutting Plane Algorithm - CPA

Initialization: Solve the following to obtain $\mathbf{W}^{(1)}$

$$\begin{array}{ll} \min\limits_{\mathbf{W}} & \| \left[\mathbf{W}^\mathsf{T}\mathbf{X} - \mathbf{Z}\right]_\mathcal{I} \|^2 \\ \text{s.t.} & \|\mathbf{W}\|_0 \le \kappa \end{array} \tag{9}$$

and set $u_i^{(1)} = 0, \forall i = 1,\dots,m_\ell$.

for $k = 1, 2, \dots$ **do**

(a) If $\sum_{i=1}^{m_\ell} \phi(\mathbf{w}_i^{(k)}) = \sum_{i=1}^{m_\ell} u_i^{(k)}$, then terminate.

(b) For each $i \in \{1,\dots,m_\ell\}$, we generate a new cut for problem :$\mathbf{a}_i^{(k)\mathsf{T}}\mathbf{w}_i + b_i^{(k)} \le u_i$, where

$$\mathbf{a}_i^{(k)} = 2\sum_{(i,j)\in\hat{\mathcal{A}}} (\mathbf{w}_i^{(k)\mathsf{T}}\mathbf{x}_j)\mathbf{x}_j,\ b_i^{(k)} = -\sum_{(i,j)\in\hat{\mathcal{A}}} (\mathbf{w}_i^{(k)\mathsf{T}}\mathbf{x}_j)^2$$

(c) Solve problem (8) to get $\mathbf{W}^{(k+1)}$ and $\mathbf{u}^{(k+1)}$.

B. Solving Subproblems

In this section, we investigate algorithms to solve subproblems (8), (9), and (3). We can solve the problem (8) by the alternating direction method of multipliers (ADMM). We reformulate the linear inequalities as follows

$$\begin{array}{l} \mathbf{a}_i^{(t)\mathsf{T}}\mathbf{w}_i + b_i^{(t)} - u_i + v_{i,t} = 0, \\ v_{i,t} \ge 0,\ (i,t) \in \mathcal{C}_k. \end{array} \tag{10}$$

We note that the problems (9), (3), and the ones in ADMM can be expressed in the following finite-sum form

$$\min_{\mathbf{w}\in\mathbb{R}^d:\|\mathbf{w}\|_0\le\kappa} \psi(\mathbf{w}) = \sum_{n=1}^{N} f_n(\mathbf{w}) = \big((\mathbf{a}_n^\mathsf{T}\mathbf{w} - b_n)^2 + \mathbf{c}^\mathsf{T}\mathbf{w}\big)$$

for some vectors $\mathbf{c}, \mathbf{a}_n \in \mathbb{R}^d$ and $b_n \in \mathbb{R}$. As explained in [23], stochastic gradient decent algorithms will work well if the variance of gradients $\nabla f_n(\mathbf{w})$ evaluated at the optimal solution of $\psi(\mathbf{w})$ is relatively small. This is what we could expect since the losses $\| \left[\mathbf{W}^\mathsf{T}\mathbf{X} - \mathbf{Z}\right]_\mathcal{I} \|^2$ and $\big(\mathbf{a}_i^{(t)\mathsf{T}}\mathbf{w}_i + b_i^{(t)} - u_i^{(k)} + v_{i,t}^{(k)}\big)^2$ should be small in our applications. As a result, we use SARAH [24] with a sparsity projection step for solving these subproblems.

V. Numerical Experiments

This section first shows the benefit of the proposed models (3) and (4) for layer-wise pruning in deep neural

Table II: Model reconstruction error results

Size (m,n)	**Method**	1 (%)	3 (%)	5 (%)	7 (%)	9 (%)
(2000, 5000)	Net-Trim	4.218	3.712	2.969	1.249	0.712
	`Model` (3)	1.731	0.576	0.216	0.071	0.011
	`Model` (4)	1.674	0.453	0.165	0.042	0.006
(2000, 10000)	Net-Trim	6.332	5.675	3.123	1.416	0.164
	`Model` (3)	2.431	0.860	0.299	0.102	0.0154
	`Model` (4)	2.255	0.506	0.079	0	0
(2000, 15000)	Net-Trim	7.730	6.500	2.827	0.092	0.016
	`Model` (3)	3.086	0.949	0.291	0.076	0.007
	`Model` (4)	2.614	0.312	0.0001	0	0
(5000, 20000)	Net-Trim	8.055	6.720	3.316	0.957	0.081
	`Model` (3)	2.140	0.705	0.269	0.083	0.011
	`Model` (4)	1.984	0.469	0.106	0.010	0

networks in terms of the reconstruction error quality. We then demonstrate a good scalability of the proposed CPA algorithm for solving large-scale problems (4). Finally, we test pruning performance for popular neural networks using real-world data sets.

For CPA, we terminated the algorithm when $|\sum_{i=1}^{m_\ell} \phi(\mathbf{w}_i^{(k)}) - \sum_{i=1}^{m_\ell} u_i^{(k)}| \leq 10^{-4}$. The penalty parameter β for the augmented Lagrangian term in ADMM is set to be 1.0. Starting points $\mathbf{u}^{(1)}, \mathbf{v}^{(1)}$ and $\boldsymbol{\lambda}^{(1)}$ for ADMM are initialized at zero. For SARAH, we set $\tilde{\mathbf{w}}^{(0)} = \mathbf{0}, \eta = 10^{-4}$ and $h = 0.5d$. Our code is written in Python and run on a machine with Nvidia Titan V graphics card and 32G memory.

A. Reconstructed Error Comparison

In this subsection, we numerically compare the reconstructed error of our pruning models (3) and (4) with Net-Trim [8], and validate the theoretical result given in Theorem 1. We use synthetic data with various sizes for these experiments.

For a given matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, a vector $\mathbf{b} \in \mathbb{R}^m$, and a fixed sparsity level κ, we compare the reconstructed error for each layer computed by

$$error = \|\sigma(\mathbf{A}\hat{\mathbf{x}}) - \sigma(\mathbf{b})\|^2/m. \tag{11}$$

Here, $\hat{\mathbf{x}} \in \mathbb{R}^n$ is the sparse solution obtained from each model (3), (4), and Net-Trim. We tune the ℓ_1 penalty parameter for Net-Trim so that the cardinality of its solution is about κ.

We now discuss how to generate the data for $\mathbf{A}$ and $\mathbf{b}$. The entries of $\mathbf{A}$ are randomly generated from the standard uniform distribution on the open interval $(-10, 10)$. As observed in [4], the pruned weights can form a bimodal distribution; hence the non-zero entries for ground-truth weight vector $\mathbf{x}_{true}$ are sampled from a bimodal Gaussian distribution. The means are -0.5 and 1, and standard deviations are 0.75 and 1.2, respectively. The sparsity for the ground-truth weight vector $\mathbf{x}_{true}$ is set to be 10% (i.e., $\frac{\|\mathbf{x}_{true}\|_0}{n} = 0.1$). The observed vector is $\mathbf{b} = \mathbf{A}\mathbf{x}_{true} + \boldsymbol{\delta}$, where the noise $\boldsymbol{\delta}$ is sampled from a Gaussian distribution with mean zero and variance of 10^{-3}. As shown in Table I, practical values for $|\mathcal{A}| = |\{i : b_i \leq 0\}|$ are often large, dominate $|\mathcal{I}| = |\{i : b_i > 0\}|$. Thus we choose $|\mathcal{A}| = 0.7m$. We can achieve this goal by randomly selecting positions for negative b_i's and simultaneously adjusting the sign of row $\mathbf{A}(i,:)$ and δ_i to $(-\mathbf{A}(i,:), -\delta_i)$ if needed.

In Table II, we report the reconstructed error (11) for three models with different sparsity levels $100 * \frac{\|\hat{\mathbf{x}}\|_0}{n}(\%)$ for various problem sizes (m, n). As we can see, the ℓ_1-based model Net-Trim gets the highest errors, they are much larger than those given by (3) and (4). We notice that even model (3) is designed for a general activation function and Net-Trim is customized only for ReLU, but (3) still gets a better performance over Net-Trim. Our model (4) has the smallest error, it can exploit the special structure of ReLU and outperforms the general-purpose model (3). In some cases when retaining 7% or 9% of non-zero entries, model (4) incurs no reconstructed error.

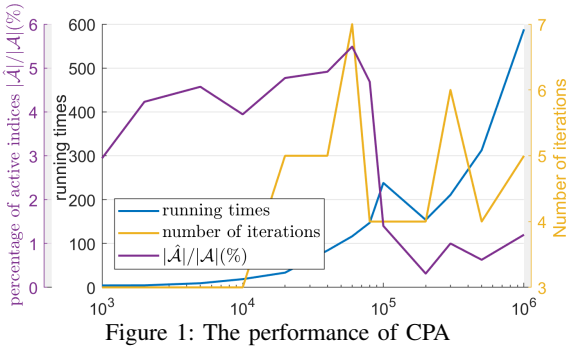

Figure 1: The performance of CPA

B. Performance of CPA

Now we investigate the performance of CPA algorithm using the same synthetic data generation mechanism proposed in Subsection V-A. We fix the sparsity level constraint at 3%, i.e. $\|\mathbf{x}\|_0 = 0.03 * n$, $m = 10000$, and vary the value of n from 10^3 to 10^6. In Figure 1 we present the running times in seconds and the number of iterations for CPA. As we mentioned, the main motivation of two-stage CPA is to handle cases when $|\mathcal{A}|$ is large (relative to $|\mathcal{I}|$), and at the optimal solution many sub-problems of the second stage (7) are inactive, that is, $\mathbf{w}_i^\mathsf{T}\mathbf{x}_j \leq 0$ (then $\phi(\mathbf{w}_i) = 0$). The latter is related to the number of cuts added to problem (8). More precisely, we expect that the ratio $\frac{|\hat{\mathcal{A}}|}{|\mathcal{A}|}$ is small, where $\hat{\mathcal{A}} = \{i \in \mathcal{A} : \mathbf{A}(i,:)\hat{\mathbf{x}} > 0\}$ and $\mathcal{A} = \{i : b_i \leq 0\}$. Hence, we also show the ratio $|\hat{\mathcal{A}}|/|\mathcal{A}|$ in percentage. They are always less than 5.5% in these test cases. From Figure 1 we can see that the number of iterations for CPA is small, varying from 3 to 7. The algorithm attempts to predict the indices $i \in \mathcal{A}$ so that $\mathbf{A}(i,:)\mathbf{x} > 0$ at the optimal solution, after a few iterations CPA correctly predicts the active set $\hat{\mathcal{A}}$. The ratio $|\hat{\mathcal{A}}|/|\mathcal{A}|$ is small, which can explain why only a small number of iterations for CPA is needed. Finally, we observe that the algorithm scales well to large-sized problems with respect to running times.

C. Pruning Neural Networks

We conducted our experiments for pruning four popular neural network architectures consisting of a fully-connected network **LeNet-300-100** [22] and three convolutional neural networks: **LeNet-5-Caffe** [17], **VGG-16** [1], and **ResNet-50** [25] for classification tasks. We used three standard datasets, which are **`MNIST`** for training LeNet-300-100 and LeNet-5-Caffe networks, **`CIFAR-10`** for training VGG-16 network, and **`ImageNet`** ILSVRC-2012 for training ResNet-50.

To obtain sparse networks by pruning, we first well trained these networks by using Adam in TensorFlow with a learning rate of 10^{-4} and a batch size of 128. After using

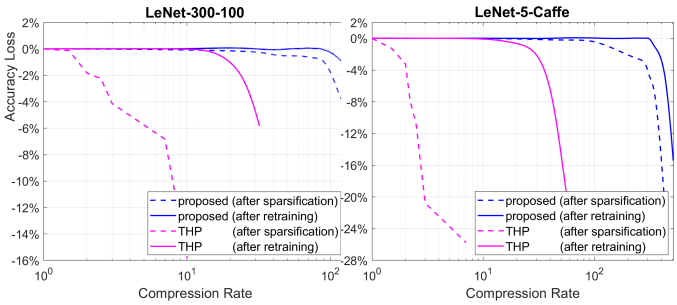

Figure 2: Compression rate and accuracy loss

our layer-wise pruning method in the second step, we also retrained the network for nonzero weights as suggested in [4]. An ℓ_2 regularizer with a regularization parameter of 10^{-5} was used in both pre-trained and re-trained networks. In particular, we applied our pruning technique to these networks by solving the problem (4) for ReLU activation layers by the cutting plane algorithm and problem (3) for other activation functions or loss layers. These models can handle skip connections as in ResNet-50 by sparsifying inputs of activation functions. Our proposed pruning method is denoted by "CPA".

We compare the proposed CPA with state-of-the-art baseline methods: threshold pruning (THP) [4], layer-wise optimal brain surgeon (L-OBS) [9], dynamic network surgery (DNS) [16], soft weight-sharing (SWS) [15], group ordered weighted $\ell_1\_\ell_2$ (GrOWL) [26], single-shot network pruning (SNIP) [11], sparse variational dropout (SparseVD) [17], and global sparse momentum (GSM) [27]. We are interested in the compression performance for the networks, where each method results in without or with little drop of top-1 test accuracy.

First, we test the behavior of our pruning method for different levels of sparsity on two networks: LeNet-300-100 and LeNet-5-Caffe. We show in Figure 2 the trade-off curve between accuracy drop and number of retained parameters, which is represented as a compression rate, for the proposed method CPA and THP. The compression rate is defined by the ratio of number of nonzero weights in the original model versus the compressed network. For each method, we present the test accuracy for both after layer-wise sparsification and retraining steps. We observe that our model (4) takes into account minimizing the reconstruction error when pruning, thus CPA can compress these networks with a moderate compression rate without accuracy loss even no retraining is needed. For a very high compression rate, a retraining step only helps to increase a few percentages of accuracy. Nevertheless, the accuracy for magnitude-based pruning THP after sparsification without retraining drops quickly as sparsity level increases, and the test accuracy gaps between sparsification and retraining are often large.

Next, in Table III we report the final compression rate ("Compression Rate") for all above-mentioned methods together with the number of model parameters for pre-trained networks ("#Params"), the test error for the original and pruned networks ("Top-1 Error"), the accuracy loss ("Error Drop"), and sparsity for each layer ("Sparsity per Layer"). We can see that our ℓ_0-based optimization approach CPA achieves the highest compression rate compared to other state-of-the-art methods for 3 out of 4 neural networks. For ResNet-50, GSM has a slightly better compression rate than that of our method (5.0 versus 4.8), but it incurs a higher accuracy drop than CPA (1.42 versus 0.06). Obviously, CPA outperforms L-OBS, one of the best layer-wise pruning methods.

Table III: Comparison of pruning results (ED = Error Drop, CR = Compression Rate)

Method	Top-1 Error (%)	ED	Sparsity per Layer (%)	CR
MNIST · LeNet-300-100 · 267K				
THP	1.64 → 1.59	-0.05	92.0 - 91.0 - 74.0	12
L-OBS	1.76 → 1.82	0.06		14
SWS	1.89 → 1.94	0.05		23
GrOWL	1.70 → 1.90	0.20		24
SNIP	1.70 → 2.40	0.70		50
DNS	2.28 → 1.99	-0.29	98.2 - 98.2 - 94.5	56
SparseVD	1.64 → 1.94	0.28	98.9 - 97.2 - 62.0	68
GSM	1.81 → 1.82	0.01		60
CPA	1.77 → 1.73	-0.04	99.2 - 98.0 - 65.0	**85**
MNIST · LeNet-5-Caffe · 431K				
THP	0.80 → 0.77	-0.03	34.0 - 88.0 - 92.0 - 81.0	12
L-OBS	1.27 → 1.27	0.00		14
SWS	0.88 → 0.97	0.09		200
SNIP	0.90 → 1.10	0.20		100
DNS	0.91 → 0.91	0.00	86.0 - 97.0 - 99.3 - 96.0	111
SparseVD	0.80 → 0.75	-0.05	67.0 - 98.0 - 99.8 - 95.0	280
GSM	0.79 → 0.94	0.15		300
CPA	0.81 → 0.79	-0.02	79.0 - 99.0 - 99.8 - 95.6	**317**
CIFAR-10 · VGG-16 · 15.9M				
GrOWL	6.60 → 7.30	0.70		15
SparseVD	7.30 → 7.30	0.00		48
CPA	7.22 → 7.21	-0.01		**56**
ImageNet · ResNet-50 · 25.5M				
L-OBS				2.2
GSM	24.28 → 25.70	1.42		**5.0**
CPA	23.82 → 23.88	0.06		4.8

VI. Conclusions

In this paper, we proposed a novel layer-wise pruning method CPA for a deep neural network based on ℓ_0-constrained optimization. For a general activation function, we formulate the task of removing unimportant weights as solving the best subset selection problem. For a layer employing the rectifier, we have introduced an ℓ_0-constrained quadratic programming and a cutting plane algorithm. These models have a smaller reconstructed error for each layer than existing methods Net-Trim and L-OBS. Furthermore, we have provided a convergence analysis for the cutting plane algorithm with a global convergence guarantee.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *NIPS*, 2013, pp. 2148–2156.

[3] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *ICLR*, 2017.

[4] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.

[5] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," in *ICLR*, 2020.

[6] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" in *The 3rd MLSys Conference*, 2020.

[7] C. Louizos, M. Welling, and D. Kingma, "Learning sparse neural networks through ℓ_0 regularization," in *ICLR*, 2018.

[8] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: Convex pruning of deep neural networks with performance guarantee," in *NIPS*, 2017.

[9] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *NIPS*, 2017, pp. 4857–4867.

[10] M. Carreira-Perpinan and Y. Idelbayev, ""Learning-compression" algorithms for neural net pruning," in *CVPR*, 2018.

[11] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *ICLR*, 2019.

[12] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *NIPS*, 1990, pp. 598–605.

[13] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *NIPS*, 1993, pp. 164–171.

[14] G. Li, C. Qian, C. Jiang, X. Lu, and K. Tang, "Optimization based layer-wise magnitude-based pruning for dnn compression," in *IJCAI*, 2018.

[15] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *ICLR*, 2017.

[16] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *NIPS*, 2016.

[17] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *ICML*, 2017, pp. 2498–2507.

[18] S. Nowlan and G. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural Comput.*, vol. 4, no. 4, pp. 473–493, 1992.

[19] J. Yoon and S. Hwang, "Combined group and exclusive sparsity for deep neural networks," in *ICML*, 2017.

[20] D. Bertsimas, A. King, and M. R, "Best subset selection via a modern optimization lens," *The Annals of Statistics*, pp. 813–852, 2016.

[21] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] L. Nguyen, N. Nguyen, D. Phan, J. Kalagnanam, and K. Scheinberg, "When does stochastic gradient algorithm work well?" *arXiv:1801.06159*, 2018.

[24] L. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, "SARAH: A novel method for machine learning problems using stochastic recursive gradient," *ICML*, 2017.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[26] D. Zhang, H. Wang, M. Figueiredo, and L. Balzano, "Learning to share: simultaneous parameter tying and sparsification in deep learning," in *ICLR*, 2018.

[27] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum sgd for pruning very deep neural networks," in *NeurIPS*, 2019.