Optimization based Layer-wise Magnitude-based Pruning for DNN Compression*

Guiying Li[1], Chao Qian[1], Chunhui Jiang[1], Xiaofen Lu[2], Ke Tang[3]
[1] Anhui Province Key Lab of Big Data Analysis and Application,
University of Science and Technology of China, Hefei 230027, China
[2] CERCIA, School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK
[3] Shenzhen Key Lab of Computational Intelligence, Department of Computer Science and Engineering,
Southern University of Science and Technology, Shenzhen 518055, China
{lgy147, beethove}@mail.ustc.edu.cn, chaoqian@ustc.edu.cn, {luxf, tangk3}@sustc.edu.cn

Abstract

Layer-wise magnitude-based pruning (LMP) is a very popular method for deep neural network (DNN) compression. However, tuning the layer-specific thresholds is a difficult task, since the space of threshold candidates is exponentially large and the evaluation is very expensive. Previous methods are mainly by hand and require expertise. In this paper, we propose an automatic tuning approach based on optimization, named OLMP. The idea is to transform the threshold tuning problem into a constrained optimization problem (i.e., minimizing the size of the pruned model subject to a constraint on the accuracy loss), and then use powerful derivative-free optimization algorithms to solve it. To compress a trained DNN, OLMP is conducted within a new iterative pruning and adjusting pipeline. Empirical results show that OLMP can achieve the best pruning ratio on LeNet-style models (i.e., 114 times for LeNet-300-100 and 298 times for LeNet-5) compared with some state-of-the-art DNN pruning methods, and can reduce the size of an AlexNet-style network up to 82 times without accuracy loss.

1 Introduction

Deep neural networks (DNNs) have achieved outstanding performance in recent years [Krizhevsky *et al.*, 2012; LeCun *et al.*, 2015]. However, DNNs usually suffer from their enormous number of parameters, which makes them prohibitive to be deployed on platforms with limited memory and processing units, e.g., mobile phones [Kim *et al.*, 2016]. To overcome this obstacle, a variety of approaches have been developed to compress those trained DNNs with over-parameterized structures [Denil *et al.*, 2013; Han *et al.*, 2015; Molchanov *et al.*, 2017].

Layer-wise magnitude-based pruning (LMP) is an effective DNN compression method and has achieved significant results in many applications [Han *et al.*, 2015; Guo *et al.*, 2016; See *et al.*, 2016]. The idea is to prune connections in each layer separately by removing the connections with absolute weight values lower than a layer-specific threshold. Given a threshold for each layer, LMP can prune connections in parallel, which is especially useful for DNNs with millions or billions connections. With well chosen pruning thresholds, LMP can achieve a significant reduction in the number of parameters, while maintaining a relatively low accuracy loss.

A main difficulty in applying LMP is tuning the pruning thresholds. Such a threshold tuning problem is non-trivial for two reasons. First, the solution space of all threshold combinations can be very large even for a moderate-sized DNN. Suppose a DNN has L layers and each layer has W connections, then the possible combinations of all layer-specific thresholds will be of size $(W+1)^L$. Second, the evaluation of each candidate threshold combination is very time consuming, because it needs to evaluate the performance loss of the pruned model over the training set, e.g., the accuracy loss after retraining. The commonly used approach to tune the thresholds is by hand [Guo *et al.*, 2016; Han *et al.*, 2015], which highly relies on expertise. In order to facilitate non-professional users to use LMP, an automatic threshold adjusting method is required.

It is an appealing idea to tune the thresholds of LMP from an optimization point of view. However, the performance measure of the thresholds is usually a discontinuous and non-differentiable function, which prevents the direct use of off-the-shelf optimization techniques such as gradient descent methods. An alternative way is to use derivative-free optimization methods [Goldberg, 1989; Brochu *et al.*, 2010; Qian *et al.*, 2015; Yu *et al.*, 2016], which do not require the problem to be either continuous or differentiable.

In this paper, we propose an optimization based approach, namely Optimization based LMP (OLMP), to automatically tune the pruning thresholds for LMP. Concretely, the threshold tuning problem is formulated as a constrained optimiza-

*This work was supported in part by the National Key Research and Development Program of China (2017YFB1003102), the NSFC (61603367, 61672478), the YESS (2016QNRC001), and the Science and Technology Innovation Committee Foundation of Shenzhen (ZDSYS201703031748284).

tion problem, which requires minimizing the size (i.e., the number of connections) of the pruned network, subject to a constraint on the accuracy loss. Then, a powerful derivative-free optimization algorithm is employed to solve this problem. To deal with the costly evaluation of accuracy loss, OLMP evaluates the accuracy loss of the pruned model on a randomly sampled small data set instead of the whole data set. Note that to compress a trained DNN, OLMP is conducted within a new iterative pruning and adjusting pipeline.

To empirically evaluate the performance of OLMP, a recently proposed powerful derivative-free optimization algorithm, negatively correlated search (NCS) [Tang *et al.*, 2016], is adopted. Without incurring any accuracy loss on test data, OLMP can prune **99.66%** parameters of LeNet-5 and **99.12%** parameters of LeNet-300-100, which are the best results in comparison to a number of state-of-the-art approaches [Han *et al.*, 2015; Guo *et al.*, 2016; Ullrich *et al.*, 2017; Molchanov *et al.*, 2017]. We also apply OLMP to prune an AlexNet-style deep model and **98.78%** parameters can be removed without sacrificing accuracy.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 introduces the proposed OLMP algorithm. Section 4 presents the empirical studies. Section 5 finally concludes the paper.

2 Related Work

Pruning methods [LeCun *et al.*, 1989; Hassibi and Stork, 1992] are widely used for compressing trained DNNs. They usually use some metrics to measure the saliency of parameters and then set thresholds to abandon parameters with low saliency. There are typically two types of pruning methods: neuron pruning [He *et al.*, 2014] and connection pruning [Hassibi and Stork, 1992; LeCun *et al.*, 1989]. This paper focuses on connection pruning.

LMP extends magnitude-based pruning (MP) [LeCun *et al.*, 1989] with respect to the multi-layer nature of DNNs. Given a set of connections and a positive threshold, MP prunes connections with absolute weight values smaller than the threshold. Considering the non-linear connected nature of DNNs, LMP applies MP with different thresholds to different layers. Popular LMP methods always tune the thresholds by experience, e.g., iterative pruning and retraining (ITR) [Han *et al.*, 2015], dynamic surgery (DS) [Guo *et al.*, 2016] and layer-wise optimal brain surgeon [Dong *et al.*, 2017].

There also exist some connection pruning methods without the need of tuning layer-specific thresholds. SWS [Ullrich *et al.*, 2017] extends the idea of soft weight-sharing [Nowlan and Hinton, 1992] to DNN compression. It fits a mixture of multiple Gaussian models over the weights with one Gaussian model fixed to zero mean, and then prunes the weights which are under the zero-mean distribution. Sparse VD [Molchanov *et al.*, 2017] uses a technique called variational dropout, which provides Bayesian interpretation to Gaussian dropout. It assigns an individual dropout rate to each weight and explores all possible values of dropout rates to find a sparse model.

Pruning followed by adjusting is a popular approach to getting better performance than applying pruning alone. For example, ITR and DS both use an additional model adjusting phase after pruning. DS first prunes the network drastically based on a variant of LMP, and then retrains the pruned model and randomly recovers the incorrectly removed connections by an operation called splicing. ITR iteratively applies LMP and then retrains the pruned model until the model cannot retrieve the original accuracy. Similar ideas of ITR can also be found in [Castellano *et al.*, 1997].

The proposed OLMP adopts a heuristic optimization algorithm NCS [Tang *et al.*, 2016] to optimize the pruning thresholds. Heuristic optimization algorithms have been used to evolve the network structure [Yao and Liu, 1997], and to prune the connections [Reed, 1993]. However, most of them are effective only for shallow networks, and are computational impossible for DNNs with millions of parameters. Recently, Google Research [Real *et al.*, 2017] successfully applied genetic algorithms to design the structures of DNNs by using clusters of GPU servers.

3 The Proposed Approach

In this section, we introduce OLMP, the whole framework of DNN compression with OLMP, and a realistic implementation of OLMP, respectively.

3.1 OLMP

Suppose a trained DNN model with $L+1$ layers can be represented as $W = \{W^l_{i,j} \mid W^l_{i,j} \neq 0, 1 \leq l \leq L, 1 \leq i \leq n^l, 1 \leq j \leq n^{l+1}\}$, where $W^l_{i,j}$ denotes the connection weight between the ith neuron in layer l and the jth neuron in layer $l+1$, and n^l denotes the number of neurons in layer l. Note that $W^l_{i,j} = 0$ indicates that the corresponding connection does not exist. Let $f(W)$ denote the evaluation function which measures the accuracy of model W, and let δ denote the user-defined tolerance on accuracy loss. For W, the task of finding the best pruned model with respect to δ can be formulated as

$$W^* = \arg\min_{W' \subseteq W} |W'| \quad s.t. \;\; f(W) - f(W') \leq \delta. \tag{1}$$

Note that W' is a subset of W (namely a pruned model), $|W'|$ denotes the size of W', and W^* denotes the best pruned model with the smallest size while satisfying the constraint.

In LMP, the generation of the pruned model W' is decided by the pruning thresholds at each layer. That is, the pruning in layer l of W is described by $\mathrm{MP}(W, l, \varepsilon_l) = \{W^l_{i,j} \mid |W^l_{i,j}| \geq \varepsilon_l, 1 \leq i \leq n^l, 1 \leq j \leq n^{l+1}\}$, where $|W^l_{i,j}|$ denotes the absolute value of $W^l_{i,j}$ and ε_l denotes the pruning threshold at layer l. For many LMP methods, ε_l is always determined by a heuristic function g which contains hyper-parameters to be tuned. Here, we only consider one hyper-parameter c_l to be tuned for each layer l, denoted as $\varepsilon_l = g(c_l)$. Therefore, the pruned model by LMP can be represented as follows:

$$\mathrm{LMP}(W, \boldsymbol{c}) = \bigcup_{l=1}^{L} \mathrm{MP}(W, l, g(c_l)), \tag{2}$$

where $\boldsymbol{c} = (c_1, c_2, \ldots, c_L)$. By combining Eqs. (1) and (2), the optimization based LMP (OLMP) can be formulated as:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c} \in \mathbb{R}^L, W' = \mathrm{LMP}(W, \boldsymbol{c})} |W'| \quad s.t. \;\; f(W) - f(W') \leq \delta. \tag{3}$$

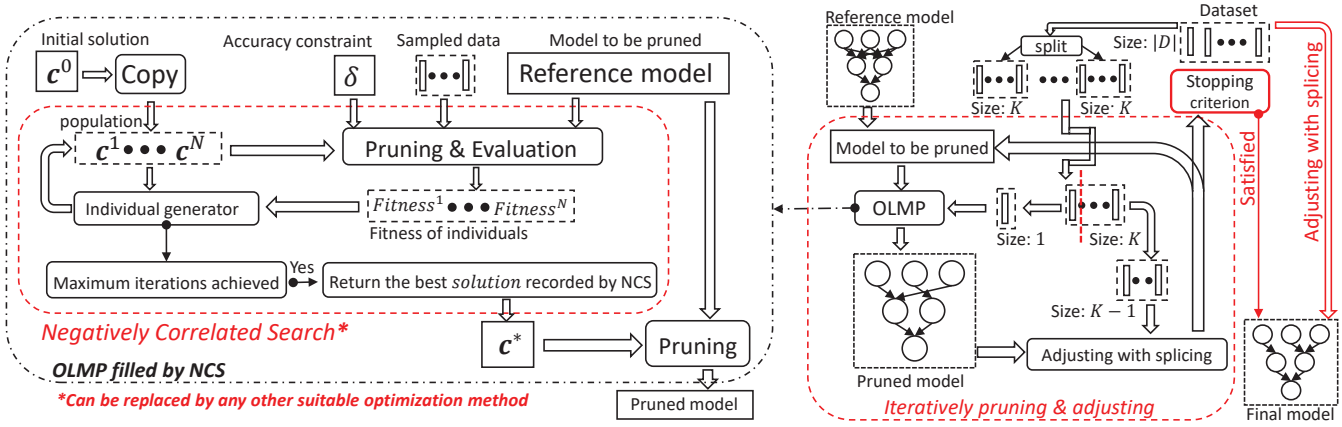

Figure 1: Illustration of DNN compression with OLMP, given an accuracy constraint δ, and a reference model as inputs. The **Right part** presents the compression process in which pruning and adjusting are conducted iteratively until the stopping criterion is fulfilled. The pruning step adopts OLMP to generate a pruned model, while the adjusting step recovers the accuracy loss of the pruned model. In each loop of pruning and adjusting, it will fetch a small set of data with K batches from the whole data set D, and use one batch for pruning while the rest for adjusting. The **Left part** illustrates the structure of OLMP using NCS. In OLMP, the threshold tuning problem is formulated as a single-objective optimization problem with an accuracy constraint δ, and then optimized using NCS to get the best found thresholds. After that, a pruned model can be derived by applying LMP on the reference model with the obtained thresholds.

The formulated problem Eq. (3) is usually difficult, which can be discontinuous and non-differentiable. Thus, we need to employ a powerful derivative-free optimization algorithm to solve it. After getting the output hyper-parameter vector $\boldsymbol{c}^*$, we can get a corresponding pruned model $\mathrm{LMP}(W, \boldsymbol{c}^*)$. Note that OLMP can be used in DNN compression together with other techniques, like accuracy adjustment.

3.2 DNN Compression with OLMP

To apply OLMP in DNN compression, we use a new iterative pruning and adjusting pipeline, as shown in the right part of Figure 1. Suppose the whole data set D contains $|D|$ batches of data, and is split into $|D|/K$ small sets, where each set contains K batches. In each iteration of the pipeline, one split data set is selected for pruning and adjusting. OLMP is first used to prune the reference model based on one batch extracted from the selected split data set, and then the pruned model is adjusted on the rest batches of the split data set to recover its accuracy. Note that the adjusting phase utilizes retraining with splicing to recover the incorrect pruned connections just the same as DS does [Guo *et al.*, 2016]. The adjusting will last one epoch on the $K-1$ batches of the split data set and the refined model will then be treated as a new reference model for pruning in the next loop. This procedure will be repeated until a limit number of loops (denoted as $pruning\_loops$) is reached. Finally, the pruned model will be retrained with splicing on the whole data until it converges.

One thing should be noted here is how to combine splicing into the pipeline. In DS, for a layer l, there are two parameters (a_l, b_l) where a_l is the pruning threshold and b_l controls the splicing operation. Through an investigation of the released code, we find that $\begin{cases} a_l = 0.9 \times \max\{\theta_l + c_l\sigma_l, 0\} \\ b_l = 1.1 \times \max\{\theta_l + c_l\sigma_l, 0\}, \end{cases}$ where θ_l and σ_l are the mean of absolute values and standard deviation of weights in layer l, and c_l is a layer-specific hyper-parameter. We use the same value for the pruning threshold $g(c_l)$ in OLMP and the pruning threshold a_l in splicing, i.e., $g(c_l) = a_l$. Thus, the hyper-parameter tuning of splicing is unified into the optimization of Eq. (3).

The iterative pipeline used in the framework is a combination of ITR and DS with the layer-wise hyper-parameters tuned by OLMP. Instead of using the whole data set in adjusting of each iteration (e.g., ITR), the proposed pipeline only uses a fraction of data for speeding up the compression. By adopting different optimization algorithms for OLMP or selecting different values for hyper-parameters (i.e., δ, K and $pruning\_loops$), this framework of DNN compression can have different implementations.

3.3 OLMP Implementation

In this subsection, we are to provide an effective implementation of the proposed DNN compression framework by using a powerful derivative-free optimization algorithm for OLMP and giving some guidelines on hyper-parameter tuning.

Here, an instantiated OLMP method based on NCS [Tang *et al.*, 2016] is established. NCS is a population-based heuristic optimization algorithm which achieves the state-of-the-art performance on many multimodal optimization problems, and does not require derivative information. It uses negative correlations to increase diversities among solutions and to encourage them to search different areas of the solution space.

To use NCS to solve the problem Eq. (3), a fitness function is needed to evaluate the qualities of solutions. A simple and direct choice is to maximize the following function $\begin{cases} \frac{|W \setminus W'|}{|W|}, & \text{if } f(W) - f(W') \le \delta \\ 0, & \text{otherwise} \end{cases}$. That is, the fitness of infeasible solutions (i.e., the constraint is violated) is assigned by the worst value 0. Note that maximizing $\frac{|W \setminus W'|}{|W|}$ is equivalent to minimizing $|W'|$ in Eq. (3). However, such a fit-

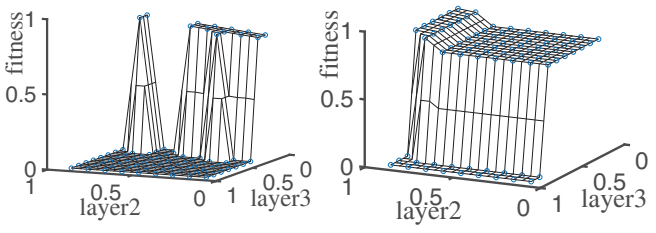

Figure 2: Landscape of solutions for iterative pruning and retraining. Left to right are results of the first to the second iteration. Each layer was pruned $10\%, 20\%, ..., 90\%$, separately. In this figure, the first layer is fixed to 90% reduced. Axis x and y indicate the percentage of reduction in layer 2 and 3, and axis z indicates the fitness values of solutions. Solutions with larger values of z are better.

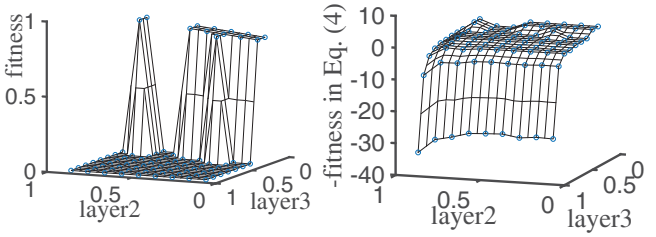

Figure 3: Transformation of the landscape by adding punishment to infeasible solutions. The left plot is the original landscape shown in Figure 2, the right plot is the landscape of function eval. It can be found that there is an obvious bias in the search space that can help the optimization algorithm to jump out of the infeasible area quickly.

ness function is usually difficult to be solved. Taking pruning LeNet-300-100 as an example, Figure 2 shows the landscape of solutions, when iterative pruning and retraining is applied. We can observe that the optimal solution is surrounded by some "plateau" structures on the landscape, which cannot provide useful searching information. Thus, we propose to minimize the fitness function Eq. (4), which prefers infeasible solutions with less constraint violation degree instead of directly assigning them by the same worst value.

$$\text{fitness}(\boldsymbol{c}) = -\text{eval}(\text{LMP}(W, \boldsymbol{c})),$$
$$\text{eval}(W^{'}) = \begin{cases} \frac{|W \setminus W^{'}|}{|W|}, & \text{if } f(W) - f(W^{'}) \leq \delta \\ -\frac{f(W)-f(W^{'})}{\delta}, & \text{otherwise} \end{cases}. \quad (4)$$

Figure 3 shows the negative fitness landscape of Eq. (4), which becomes less flat and will be easier to be optimized.

The procedure of OLMP with NCS is shown in the left of Figure 1. A reference model W, a user defined δ and a sampled data set are packed into a fitness evaluation module. In each iteration, for newly generated alternatives of $\boldsymbol{c}$, the evaluation module prunes the reference model by $\text{LMP}(W, \boldsymbol{c})$ and returns the fitness values; NCS then searches for new solutions based on these solutions and their fitness values.

As NCS is adopted, its hyper-parameters are involved into the compression process. Although NCS contains adaptive settings for its parameters, we find some manual selections for initial parameter values can speed up the search process. The parameters of NCS that we tune are σ, pop_N and T_{max} where σ decides the stride in searching solutions, pop_N decides the number of search directions and T_{max} decides the epochs of NCS. Therefore, the overall hyper-parameters consist of those used by the proposed framework (i.e., δ, K and $pruning\_loops$) and those introduced by the employed optimization algorithms (e.g., σ, pop_N and T_{max} for NCS).

In practice, we can first fix δ and σ to a small value (e.g., 0.1) and tune other parameters. The sequence of tuning was $K \rightarrow pruning\_loops \rightarrow T_{max} \rightarrow pop_N$. For each parameter, we can use grid search to get a value at which the performance stops increasing. Finally, we can tune δ and σ based on the sensitivity analysis, as provided in Section 4.3.

4 Experiments

In this section, we empirically investigate the performance of OLMP. Firstly, we compare OLMP with some state-of-the-art connection pruning methods on LeNet-5 and LeNet-300-100 [Han *et al.*, 2015], which are commonly used by all comparative methods. Secondly, OLMP is applied to compress a real deep network, i.e., an AlexNet [Krizhevsky *et al.*, 2012] style model called AlexNet-Caltech. Thirdly, we perform a sensitivity analysis of hyper-parameters. Finally, we study how OLMP performs without the proposed iterative pipeline.

For the data sets and models, LeNet-5 and LeNet-300-100 are trained on MNIST, and AlexNet-Caltech is trained on Caltech-256 [Griffin *et al.*, 2006]. Note that the output dimension of AlexNet-Caltech is 257 instead of 1000 in AlexNet, since Caltech-256 contains 257 classes (one class for unrecognized images); and this is the only difference in structures compared with original AlexNet.

In the context of all experiments, pruning ratio (PR) indicates $\frac{|W|}{|W^{'}|}$, i.e., the ratio between the number of connections of the reference model and the pruned model. All of the experiments are based on Caffe [Jia *et al.*, 2014] and released projects of DS and NCS, and run on a workstation with one Titan X pascal and dual Intel E5-2683 v3@2.0 GHz CPUs.

4.1 Comparison on LeNet-Style Models

In this subsection, we are to show that OLMP can achieve the best performance on DNN compression. We use the reference modes of LeNet-300-100 and LeNet-5 that were used in testing DS [Guo *et al.*, 2016]. Besides, SGD with the same experimental settings (e.g., base learning rate and batch size) in training these reference models is used to retrain the pruned models in the adjusting phase of OLMP. The final pruned models need to be retrained with splicing for additional $15,000$ iterations to converge. In this experiment, we set the hyper-parameters $(K, pruning\_loops, pop_N, T_{max})$ to $(1000, 15, 10, 160)$. For LeNet-300-100, (δ, σ) is set to $(8\%, 5)$; for LeNet-5, δ is set to 5%, and σ is set to 5 in the first 10 pruning loops and 0.5 since then.

The results are shown in Table 1. We can observe that OLMP achieves the best PRs on LeNet-300-100 and LeNet-5 without degrading test accuracy. Note that ITR [Han *et al.*, 2015] and DS [Guo *et al.*, 2016] are two LMP methods which manually set pruning thresholds; and SWS [Ullrich *et al.*, 2017] and Sparse VD [Molchanov *et al.*, 2017] are two non-LMP connection pruning methods.

Model	Method	Top-1 Error (%)	Accuracy Improve	PR
LeNet-300-100	ITR	1.64→**1.59**	+0.05	12
	DS	2.28→1.99	**+0.29**	56
	SWS	1.89→1.94	-0.05	23
	Sparse VD	1.64→1.92	-0.28	68
	OLMP	2.28→2.18	+0.1	**114**
LeNet-5	ITR	0.80→0.77	+0.03	12
	DS	0.91→0.91	0	108
	SWS	0.88→0.97	-0.09	200
	Sparse VD	0.80→**0.75**	**+0.05**	280
	OLMP	0.91→0.91	0	**298**

Table 1: Comparison results on LeNet-style models. OLMP gets the best PRs without accuracy loss. For each model, the best value in each column is bolded. All the results of comparative methods are the best reported ones in their original work with different reference modes. Note that OLMP and DS use the same reference models.

4.2 Application to AlexNet

In this subsection, we apply OLMP to prune a real deep network, AlexNet-Caltech with 58.33 million parameters. The reference model is trained from scratch for $10,000$ iterations using SGD which mainly follows the experimental settings in [Zeiler and Fergus, 2014] except that the batch size is 256 here; and the same settings of SGD are also used during retraining. The reference model achieves a validation accuracy of 40.18% which is similar to [Zeiler and Fergus, 2014] (i.e., $38.8 \pm 1.4\%$). The final pruned model needs to be retrained with splicing for additional $5,000$ iterations to converge. $(K, pruning\_loops, pop_N, T_{max}, \delta)$ is set to $(250, 40, 8, 200, 8\%)$, and σ is set to 5 in the first 28 pruning loops and 0.5 since then.

The results are shown in Table 2. We can see that OLMP can achieve the PR value of 82 without accuracy loss. We also give the the reduction in each layer, as shown in Table 3. We can observe that the percentages of remaining parameters in the first two layers and the last layer are much larger than that in other layers, which suggests that these three layers are less redundant than other layers.

For the computational cost, it takes 108 minutes to train the reference model, and 605 minutes to compress (433 minutes for OLMP and 172 minutes for adjusting), which is 5.6 (i.e., $605/108$) times than training the reference model. Note that NCS can be easily implemented in parallel, and the cost of NCS can be reduced by a factor of pop_N, i.e., the cost of 433 minutes can be reduced to 54 (i.e., $433/8$) minutes if enough GPUs are available.

Model	Top-1 error(%)	Parameters	PR
AlexNet-Caltech-ref	59.82	58.3M	1
AlexNet-Caltech-pruned	**59.42**	**0.7M**	**82**

Table 2: Compression results on AlexNet-Caltech. OLMP can dramatically reduce the number of parameters without accuracy loss. Note that "*-ref" indicates the reference model and "*-pruned" indicates the model pruned by OLMP. "M" indicates millions.

4.3 Analysis of Sensitivities to δ and σ

In this subsection, we investigate the impact of (δ, σ) on the performance of OLMP. LeNet-300-100 with the same settings as in subsection 4.1 is used as an example. We use $\delta \in \{1\%, 6\%, 11\%, \ldots, 51\%\}$ and $\sigma \in \{0.05,\ 5.05,\ 10.05,\ \ldots,\ 50.05\}$, and prune the model with each possible setting of (δ, σ). There are totally 121 settings of (δ, σ). For each setting, we independently run OLMP 30 times and record the number of pruned models which can recover accuracy (i.e., no accuracy loss compared with the reference model). Meanwhile, for those models which can recover accuracy, we also record their average PR. The results are shown in Table 4.

Layer	conv1	conv2	conv3	conv4	conv5	fc1	fc2	fc3	Total
Param	35K	307K	885K	664K	443K	38M	17M	1M	58.33M
Remain(%)	13.9	31.92	1.68	4.70	4.04	0.85	0.57	11.67	**1.22**

Table 3: Reductions in each layer of the pruned AlexNet-Caltech after applying OLMP. The total number of parameters in the reference model is 58.33 million and only 1.22% of the parameters are remained after compression. "K" indicates thousands.

From the distribution of bold values (i.e., the largest PR in each row) in Table 4, the PRs keep to a small value (i.e., 6) and the pruned model can always restore accuracy, when σ is very low (i.e., $\sigma = 0.05$). When σ is not very low (i.e., $\sigma \geq 5.05$), we can see that a larger δ value will possibly lead to a larger PR value. This is expected since a larger δ value implies a larger feasible solution space and thus a better optimal objective function value, i.e., a larger PR value. However, a large δ may also make the pruned model difficult to recover accuracy, which means that users need to rerun OLMP many times for finding a pruned model without accuracy loss. Thus, a large δ is prohibitive for DNNs with expensive computational cost. When δ is not very large (i.e., $\delta \leq 36\%$), the relationship between the performance of OLMP and σ is relatively stable for different δ values. That is, as σ increases, the PR tends to decrease while the number of pruned models without accuracy loss keeps within a certain range.

The above observations give us some practical guidance on tuning δ and σ. First, a suitable σ can be selected with a small fixed δ value, since the pattern between σ and the performance of OLMP is stable for small δ values. Then, we can carefully select δ for a large PR. Thus, combining with the analysis in subsection 3.3, we have shown how to tune the hyper-parameters of OLMP.

4.4 OLMP without Iterative Pruning

In this subsection, we test OLMP without iterative pruning and adjusting, to show the process of optimization in a single pruning. We also compare it with the baseline algorithm MP [LeCun *et al.*, 1989] to investigate whether optimization really brings some benefit.

Different from previous experiments, all the models used here are trained on a training set from scratch, pruned on a validation set and tested on an unseen test set. Since MNIST and Caltech-256 only have two separate data sets, a manual division is applied. For MNIST, a validation set containing $6,000$ samples is selected from the training set ($60,000$ in total) uniformly at random; the remaining samples form the new training set; the test set is untouched ($10,000$ in total). For Caltech-256, the training set is untouched ($15,420$ in total) while the validation set ($15,187$ in total) is uniformly ran-

σ \ δ	1%	6%	11%	16%	21%	26%	31%	36%	41%	46%	51%
0.05	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**	**6—30**
5.05	26—30	59—29	84—23	87—20	91—14	94—9	89—11	**99—3**	83—1	0—0	0—0
10.05	22—30	54—29	73—28	75—23	81—17	86—17	82—12	92—6	**102—3**	0—0	0—0
15.05	18—30	48—29	67—26	75—24	74—18	78—13	85—9	**96—9**	89—4	83—2	76—2
20.05	17—30	49—30	71—29	77—25	71—21	77—19	87—12	**93—6**	74—5	90—5	82—2
25.05	17—30	52—30	56—30	70—26	71—23	75—15	82—18	86—12	55—3	**101—3**	52—1
30.05	14—30	41—30	57—26	56—25	71—27	73—17	81—17	84—7	**86—8**	61—3	65—3
35.05	17—30	38—30	58—27	58—25	67—21	72—21	80—13	82—13	77—11	**86—6**	85—2
40.05	15—30	37—30	51—30	64—26	73—25	68—23	80—17	82—15	84—6	**99—6**	88—2
45.05	15—30	34—28	53—28	70—25	73—26	76—20	87—19	74—10	93—6	90—11	**100—4**
50.05	13—30	33—30	50—30	51—25	66—26	68—19	**80—15**	68—8	79—9	66—4	74—5

Table 4: Compression results of OLMP on LeNet-300-100 with different settings of (δ, σ). In each cell of the table, the value is represented as "Average PR—number of pruned models without accuracy loss". Underline values are those with the maximum PRs in each column. **Bold values** are those with largest PRs in each row.

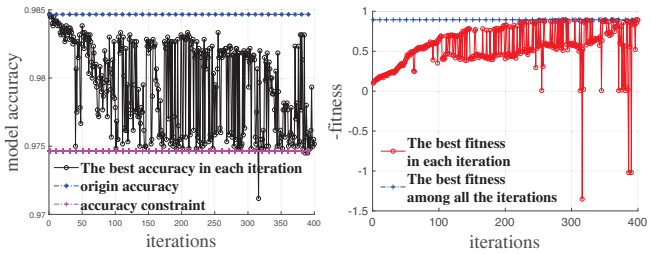

Figure 4: Visualization of the optimization process of OLMP without iterative pruning and adjusting, using LeNet-5 as the demonstration. The circles in the figures represent the corresponding results of the best found solution in each iteration. The left plot shows the variation of model accuracy and the right plot shows the evolution of $-$fitness (the larger the better).

domly divided into a new validation set ($7,530$ in total) and a new test set ($7,657$ in total). All the settings of training the reference models are the same as those used in subsections 4.1 and 4.2.

During pruning, δ is set to 1%. MP tests 100 thresholds which prune $0\%, 1\%, \ldots, 99\%$ parameters of the reference model, respectively, and then selects the pruned model with the largest PR among those without accuracy loss. For OLMP without iterative pruning and adjusting, (pop_N, σ, T_{max}) is set to $(4, 5, 400)$. Figure 4 shows the optimization process of OLMP which involves both the model accuracy and the $-$fitness value (i.e., the value of eval in Eq. (4)) of the best solution (i.e., the solution with the smallest fitness value) found in each iteration of NCS. Table 5 summarizes the results.

From the right plot in Figure 4, we can observe that OLMP can gradually improve the solution quality. To be specific, in the early stage, the found solutions always fit the accuracy constraint and the fitness values continue to increase. In the later stage, the trend of the solution quality is still becoming better, while the solutions can suddenly become worse (i.e., the curve decreases) sometimes. The reason might be that, at the early stage of pruning, the model has a lot of redundant connections and thus better solutions are easily generated; while as the optimization goes forward, the model is less redundant and it becomes easier to prune some important connections that will dramatically decrease the fitness value.

Model	Method	Val acc(%)	Val loss(%)	Test acc(%)	Test loss(%)	PR
LeNet-300-100	None	97	0	97.88	0	1
	MP	96.0	1	97.69	0.19	6.25
	OLMP	96.04±0.03	0.96±0.03	96.93±0.03	0.95±0.03	**6.41±0.06**
LeNet-5	None	98.47	0	98.42	0	1
	MP	97.67	0.8	97.72	0.7	4.76
	OLMP	97.5±0.03	0.97±0.03	98.06±0.08	0.36±0.08	**9.13±0.72**
AlexNet-Caltech	None	40.52	0	39.91	0	1
	MP	39.64	0.88	39.04	0.87	4.17
	OLMP	39.58±0.04	0.94±0.04	38.64±0.02	1.27±0.02	**4.95±0.11**

Table 5: Comparison between the baseline MP and OLMP without iterative pruning and adjusting. "None" indicates the performance of the reference model. "Val/Test acc" indicates the model accuracy on validation/test set. "Val/Test loss" indicates the accuracy of the reference model minus the accuracy of the pruned model on validation/test set. The pruned model is the best found one under the accuracy constraint 1%. Results for OLMP consist of "mean±std" which are computed from 5 independent runs. Note that pruning only considers the accuracy constraint on the validation set. **Bold values** are the best PR for pruning each DNN.

From Table 5, we can see that OLMP without iterative pruning and adjusting always achieves better PRs than MP, which implies that introducing optimization can lead to better performance. For the accuracy loss, neither OLMP nor MP has significant larger loss in testing than in validation, which shows that the pruned models on the validation data set can generalize well on the unseen test set.

5 Conclusion

State-of-the-art LMP based DNN compression methods require to manually select the pruning thresholds on each layer, and thus are hard to be applied in practice, particularly for end-users with limited expertise. This paper proposes OLMP, an optimization based LMP method which can choose thresholds automatically. OLMP first formulates the task of threshold selection as a constrained optimization problem, i.e., maximizing the pruning ratio under a constraint on the accuracy loss, and then uses a derivative-free optimization algorithm to solve the problem. To compress DNNs exhaustively, OLMP is applied within a new iterative pruning and adjusting pipeline. Empirical results show the superiority of OLMP over several state-of-the-art connection pruning methods.

References

[Brochu *et al.*, 2010] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599*, 2010.

[Castellano *et al.*, 1997] G. Castellano, A. M. Fanelli, and M. Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks*, 8(3):519–531, 1997.

[Denil *et al.*, 2013] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*, pages 2148–2156, Lake Tahoe, NV, 2013.

[Dong *et al.*, 2017] X. Dong, S.-Y. Chen, and S. J. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems 30 (NIPS'17)*, pages 4860–4874, Long Beach, CA, 2017.

[Goldberg, 1989] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, Boston, MA, 1989.

[Griffin *et al.*, 2006] G. Griffin, AD. Holub, and P. Perona. The caltech 256. *Caltech Technical Report*, 2006.

[Guo *et al.*, 2016] Y.-W. Guo, A.-B. Yao, and Y.-R. Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems 29 (NIPS'16)*, pages 1379–1387, Barcelona, Spain, 2016.

[Han *et al.*, 2015] S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28 (NIPS'15)*, pages 1135–1143, Montreal, Canada, 2015.

[Hassibi and Stork, 1992] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5 (NIPS'92)*, pages 164–171, Denver, CO, 1992.

[He *et al.*, 2014] T.-X. He, Y.-C. Fan, Y.-M. Qian, T. Tan, and K. Yu. Reshaping deep neural network for fast decoding by node-pruning. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*, pages 245–249, Florence, Italy, 2014.

[Jia *et al.*, 2014] Y.-Q. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[Kim *et al.*, 2016] Y.-D. Kim, E. Park, S.-J. Yoo, T. Choi, L. Yang, and D.-J. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proceedings of the 4th International Conference on Learning Representations (ICLR'16)*, pages 1–16, San Juan, Puerto Rico, 2016.

[Krizhevsky *et al.*, 2012] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'12)*, pages 1106–1114, Lake Tahoe, NV, 2012.

[LeCun *et al.*, 1989] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2 (NIPS'89)*, pages 598–605, Denver, CO, 1989.

[LeCun *et al.*, 2015] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Molchanov *et al.*, 2017] D. Molchanov, A. Ashukha, and D. P. Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pages 2498–2507, Sydney, Australia, 2017.

[Nowlan and Hinton, 1992] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[Qian *et al.*, 2015] C. Qian, Y. Yu, and Z.-H. Zhou. Subset selection by Pareto optimization. In *Advances in Neural Information Processing Systems 28 (NIPS'15)*, pages 1774–1782, Montreal, Canada, 2015.

[Real *et al.*, 2017] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, pages 2902–2911, Sydney, Australia, 2017.

[Reed, 1993] R. Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[See *et al.*, 2016] A. See, M.-T. Luong, and C. D. Manning. Compression of neural machine translation models via pruning. In *Proceedings of the 20th Conference on Computational Natural Language Learning (CoNLL'16)*, pages 291–301, Berlin, Germany, 2016.

[Tang *et al.*, 2016] K. Tang, P. Yang, and X. Yao. Negatively correlated search. *IEEE Journal on Selected Areas in Communications*, 34(3):542–550, 2016.

[Ullrich *et al.*, 2017] K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. In *Proceedings of the 5th International Conference on Learning Representations (ICLR'17)*, pages 1–16, Toulon, France, 2017.

[Yao and Liu, 1997] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, 1997.

[Yu *et al.*, 2016] Y. Yu, H. Qian, and Y.-Q. Hu. Derivative-free optimization via classification. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*, pages 2286–2292, Phoenix, AZ, 2016.

[Zeiler and Fergus, 2014] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Proceedings of the 13th European Conference on Computer Vision (ECCV'14)*, pages 818–833, Zurich, Switzerland, 2014.