Constrained Optimization Based Low-Rank Approximation of Deep Neural Networks

Chong Li(✉) and C. J. Richard Shi

University of Washington, Seattle, WA 98195, USA
{chongli,cjshi}@uw.edu

Abstract. We present COBLA—Constrained Optimization Based Low-rank Approximation—a systematic method of finding an optimal low-rank approximation of a trained convolutional neural network, subject to constraints in the number of multiply-accumulate (MAC) operations and the memory footprint. COBLA optimally allocates the constrained computation resources into each layer of the approximated network. The singular value decomposition of the network weight is computed, then a binary masking variable is introduced to denote whether a particular singular value and the corresponding singular vectors are used in low-rank approximation. With this formulation, the number of the MAC operations and the memory footprint are represented as linear constraints in terms of the binary masking variables. The resulted 0–1 integer programming problem is approximately solved by sequential quadratic programming. COBLA does not introduce any hyperparameter. We empirically demonstrate that COBLA outperforms prior art using the SqueezeNet and VGG-16 architecture on the ImageNet dataset.

Keywords: Low-rank approximation · Resource allocation
Constrained optimization · Integer relaxiation

1 Introduction

The impressive generalization power of deep neural networks comes at the cost of highly complex models that are computationally expensive to evaluate and cumbersome to store in memory. When deploying a trained deep neural network on edge devices, it is highly desirable that the cost of evaluating the network can be reduced without significantly impacting the performance of the network.

In this paper, we consider the following problem: given a set of constraints to the number of multiply-accumulate (MAC) operation and the memory footprint (storage size of the model), the objective is to identify an optimal low-rank approximation of a trained neural network, such that the evaluation of the approximated network respects the constraints. For conciseness, the number of MAC operation and the memory footprint of the approximated network will be referred to as *computation cost* and *memory cost* respectively.

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11214, pp. 746–761, 2018.
https://doi.org/10.1007/978-3-030-01249-6_45

Our proposed method, named COBLA (**C**onstrained **O**ptimization **B**ased **L**ow-rank **A**pproximation), combines the well-studied low-rank approximation technique in deep neural networks [1,9,11,13,15,21,27,28,30] and sequential quadratic programming (SQP) [2]. Low-rank approximation techniques exploit linear dependency of the network weights, so the computation cost and the memory cost of network evaluation can both be reduced. A major unaddressed obstacle of the low-rank approximation technique is in determining the target rank of each convolutional layer subject to the constraints. In a sense, determining the target rank of each layer can be considered as a *resource allocation* problem, in which constrained resources in terms of computation cost and memory cost are allocated to each layer. Instead of relying on laborious manual tuning or sub-optimal heuristics, COBLA *learns* the optimal target rank of each layer by approximately solving a constrained 0–1 integer program using SQP. COBLA enables the user to freely and optimally trade-off between the evaluation cost and the accuracy of the approximated network.

To the best knowledge of the authors, COBLA is the first systematic method that learns the optimal target ranks (which define the *structure* of the approximated network) subject to constraints in low-rank approximation of neural networks. We empirically demonstrate that COBLA outperforms prior art using SqueezeNet [12] and VGG-16 [26] on the ImageNet (ILSVRC12) dataset [23]. COBLA is independent of how the network weights are decomposed. We performed the experiments using two representative decomposition schemes proposed in [27] and [30]. A distinct advantage of COBLA is that it does not involve any hyperparameter tuning.

2 Low-Rank Approximation and Masking Variable

Matrix multiplication plays a pivotal role in evaluating convolutional neural networks [16]. The time complexity of exactly computing $A \cdot B$ where $A \in \mathbb{R}^{k \times l}$ and $B \in \mathbb{R}^{l \times p}$ is $\mathcal{O}(klp)$. Here A is some transformation of the weight tensor, and B is the input to the layer. With a pre-computed rank r approximation of A, denoted by $\widehat{A}$, it only takes $\mathcal{O}((k+l)pr)$ operations to approximately compute the matrix multiplication. The memory footprint of $\widehat{A}$ is also reduced to $\mathcal{O}((k+l)r)$ from $\mathcal{O}(kl)$.

The focus of this paper is in optimally choosing the target rank r for each layer subject to the constraints. This is a critical issue that was not adequately addressed in the existing literature.

If the target rank r was known, the rank r minimizer of $||A - \widehat{A}||$ (independent of the input data B) could be easily computed by the singular value decomposition (SVD). Let the SVD of A be $A = \sum_{\forall j} \sigma_j \cdot U_j \cdot (V_j)^T$, where σ_j is the jth largest singular value, U_j and V_j are the corresponding singular vectors. The rank r minimizer of $||A - \widehat{A}||$ is simply $\widehat{A} = \sum_{j \leq r} \sigma_j \cdot U_j \cdot (V_j)^T$. Let the set S_σ contain the indices of the singlar values and corresponding singular vectors that are included in the low-rank approximation. In this case $S_\sigma = \{j | j \leq r\}$. Unfortunately, identifying the input data dependent optimal value of $\widehat{A}$ that minimizes $||A \cdot B - \widehat{A} \cdot B||$ is significantly more difficult. As a matter of fact, the general weighted low-rank approximation problem is NP-hard [31].

2.1 Low-Rank Approximation of Neural Networks

Let the kernel of a convolution layer be $W \in \mathbb{R}^{c \times m \times n \times f}$, where c is the number of the input channels, m, n are the size of the filter, and f is the number of output channels. Let an input to the convolution layer be $Z \in \mathbb{R}^{c \times x \times y}$, where $x \times y$ is the size of the image. The output of the convolution layer $T = W * Z$ can be computed as

$$T(x, y, f) = W * Z = \sum_{c'=1}^{c} \sum_{x'=1}^{m} \sum_{y'=1}^{n} W(c', x', y', f) \cdot Z(c', x + x', y + y') \quad (1)$$

Given a trained convolutional neural network, the weight tensor of a convolution layers W can be decomposed into tensors G and H. Essentially, a convolutional layer with weight W is decomposed into two convolutional layers, whose weights are G and H respectively. The *decomposition scheme* defines how a four-dimensional weight tensor is decomposed. We focus on the decomposition schemes described in [27] and [30], which are representative works in low-rank approximation of neural networks. The dimensions of the weights of the decomposed layers are summarized in Table 1.

Table 1. Dimension of the decomposed layers in low-rank approximation of neural networks.

Decomposition scheme	Dimension of G	Dimension of H	Compute decomposed weight with
[27]	$[c, m, 1, r]$	$[r, 1, n, f]$	Eq. 3
[30]	$[c, m, n, r]$	$[r, 1, 1, f]$	Eq. 4

r is the target rank, which dictates how much computation cost and memory cost are allocated to a layer.

With the dimension of the decomposed weights defined by the target rank and the decomposition scheme, we now identify the optimal weight of the decomposed layers. The basic idea is to compute the SVD of some matricization of the four-dimensional network weight, and only use a subset of the singular values (together with their corresponding singular vectors) to approximate the network weight. In [27], the following low-rank approximation is applied to the weight tensor $W \in \mathbb{R}^{c \times x \times y \times f}$,

$$W[c', :, :, f'] = \sum_{\forall j} \sigma_j \cdot U_{f'}^{j} \cdot (V_j^{c'})^T \approx \sum_{j \in S_{\sigma,i}} \sigma_j \cdot U_{f'}^{j} \cdot (V_j^{c'})^T = \sum_{j \in S_{\sigma,i}} P_{f'}^{j} \cdot (V_j^{c'})^T \quad (2)$$

For conciseness, scalar σ_j is absorbed into the left singular vector U_j such that $P_j = \sigma_j \cdot U_j$.

Properly choosing S_σ for each layer subject to constraints is critical to the performance of the approximated network. Note that the target rank $r_i = |S_{\sigma,i}|$, where $|\cdot|$ denotes the cardinality of a set. The default technique is truncating the singular values, where $S_{\sigma,i}$ is chosen by adjusting a hyperparameter k_i such that $S_{\sigma,i} = \{j | j \leq k_i\}$ [27,30]. Obviously, truncating the singular values is suboptimal considering the NP-hardness of the weighted low-rank approximation problem [31]. It is worth emphasizing that k_i is a hyperparameter that has to be individually adjusted for each convolution layer in the network. Given the large number of layers in the network, optimally adjusting the $S_{\sigma,i}$ for each layer constitutes a challenging integer optimization problem by itself. COBLA can be considered as an automatic method to choose $S_{\sigma,i}$ for each layer subject to the constraints.

Equivalently, Eq. 2 can be re-written as

$$W[c', :, :, f'] \approx \sum_{j \in S_{\sigma,i}} P^j_{f'} \cdot (V^{c'}_j)^T = \sum_{\forall j} m_{ij} \cdot (P^j_{f'} \cdot (V^{c'}_j)^T) \tag{3}$$

where $m_{ij} \in \{0, 1\}$ is the *masking variable* of a singular value and its corresponding singular vectors, with $m_{ij} = 1$ indicating the jth singular value of the ith convolutional layer is included in the approximation, and $m_{ij} = 0$ otherwise. Obviously, for the ith convolutional layer $S_{\sigma,i} = \{j \mid m_{ij} = 1\}$. If $m_{ij} = 1$ for all (i, j), then all the singular values and the corresponding singular vectors are included in the approximation. If so, the approximated network would be identical to the original network (subject to numerical error). Let vector $\boldsymbol{m}$ be the concatenation of all m_{ij}. Also, let $\boldsymbol{m_i}$ denote the masking variables of the ith convolutional layer. See Fig. 1 for a small example illustrating how masking variables can be used to select the singular values and the corresponding singular vectors in low-rank approximation.

$$\left[\ U_1, U_2, \boxed{U_3}, U_4, \boxed{U_5}\ \right] \cdot \begin{bmatrix} m_1 \cdot \sigma_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_2 \cdot \sigma_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{m_3 \cdot \sigma_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & m_4 \cdot \sigma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{m_5 \cdot \sigma_5} & 0 \end{bmatrix} \cdot \begin{bmatrix} V_1^T \\ V_2^T \\ \boxed{V_3^T} \\ V_4^T \\ \boxed{V_5^T} \\ V_6^T \end{bmatrix}$$

Fig. 1. An example of utilizing masking variables to select the singular values and the corresponding singular vectors in low-rank approximation. In this example $W \in \mathbb{R}^{5\times6}$, the SVD of W is $W = U\varSigma V^T$. The values of the masking variables $m_{1..5}$ are [1,1,0,1,0], thus $S_\sigma = \{1, 2, 4\}$. $\widehat{W} = \varSigma_{j\in\{1,2,4\}} \sigma_j \cdot U_j \cdot (V_j)^T$, where $\widehat{W}$ is a rank 3 approximation of W.

We can apply the masking variables formulation to the decomposition scheme described in [30] in a similar fashion. Recall that in most mainstream deep learning frameworks, the convolution operation is substituted by matrix multiplication via the `im2col` subroutine [16]. To compute convolution as matrix multiplication, the network weight $W \in \mathbb{R}^{c \times m \times n \times f}$ is reshaped into a two dimensional matrix $W_M \in \mathbb{R}^{f \times c \cdot m \cdot n}$. In [30], low-rank approximation is applied to W_M. With a slight abuse of notations,

$$W_M = \sum_{\forall j} P_j \cdot (V_j)^T \approx \sum_{j \in S_{\sigma,i}} P_j \cdot (V_j)^T = \sum_{\forall j} m_{ij} \cdot (P_j \cdot (V_j)^T) \tag{4}$$

It is worth emphasizing that the input to the layer is not considered in Eq. 3 or Eq. 4. Much effort has been made to approximately compute the optimal weight of the decomposed layers (G and H) conditioned on the distribution of the input to the layer [11,28,30]. However, our experiment and prior work [27] indicate that the accuracy improvement enabled by data dependent decomposition vanishes after the fine-tuning process. For this reason, we simply use the data independent decomposition, and focus on identifying an optimal allocation of the constrained computation resources.

3 Problem Statement and Proposed Solution

Let $N_C(\boldsymbol{m})$ and $N_M(\boldsymbol{m})$ be the computation cost and the memory cost associated with evaluating the network. Also, let $N_{C,O}$ and $N_{M,O}$ denote the computation cost and the memory cost of the original convolutional neural network.

Consider a general empirical risk minimization problem,

$$E(\mathcal{W}) = \frac{1}{N_S} \sum_{n=1}^{N_S} \{\mathcal{L}(f(I^n, \mathcal{W}), O^n)\} \tag{5}$$

where $\mathcal{L}(\cdot)$ is the loss function, $f(\cdot)$ is the non-linear function defined by the convolutional neural network, I^n and O^n are the input and output of the nth data sample, N_S is the number of training samples, and $\mathcal{W}$ is the set of weights in the neural network.

Assuming that a convolutional neural network has been trained and low-rank approximation of the weights is performed as in Eq. 3 or Eq. 4, the empirical risk of the approximated neural network is

$$E(\boldsymbol{m}, \mathcal{P}, \mathcal{V}) = \frac{1}{N_S} \sum_{n=1}^{N_S} \{\mathcal{L}(f(I^n, \boldsymbol{m}, \mathcal{P}, \mathcal{V}), O^n)\} \tag{6}$$

where $\mathcal{P}$ and $\mathcal{V}$ are the sets of P and V vectors of all convolutional layers.

Given a system-level budget defined by the upper limit of computation cost $N_{C,max}$ and the upper limit of memory cost $N_{M,max}$, the problem can be formally stated as

$$\begin{aligned} \underset{\boldsymbol{m},\mathcal{P},\mathcal{V}}{\text{minimize}} \quad & E(\boldsymbol{m},\mathcal{P},\mathcal{V}) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \le N_{C,max} \\ & N_M(\boldsymbol{m}) \le N_{M,max} \\ & m_{i,j} \in \{0,1\} \end{aligned} \tag{7}$$

In this 0–1 integer program, the computation cost and the memory cost associated with evaluating the approximated network are expressed in terms of $\boldsymbol{m}$.

If we were given an optimal solution $\{\boldsymbol{m^*}, \mathcal{P}^*, \mathcal{V}^*\}$ to Eq. 7 by an oracle, then the optimal target rank r_i^* for the ith convolutional layer subject to the constraints is simply $\mathit{\Sigma}\boldsymbol{m_i^*}$. In other words, with the masking variable formulation, we are now able to *learn* the optimal *structure* of the approximated network subject to constraints by solving a constrained optimization problem. This is a key innovation of the proposed method.

However, exactly solving the 0–1 integer program in Eq. 7 is intractable. We propose to approximately solve Eq. 7 in a two-step process: in the first step, we focus on $\boldsymbol{m}$, while keeping $\mathcal{P}$, $\mathcal{V}$ as constants. The value of $\mathcal{P}$, $\mathcal{V}$ are computed using SVD as in Eq. 3 or Eq. 4. To approximately compute $\boldsymbol{m^*}$, we resort to integer relaxation [22], which is a classic method in approximately solving integer programs. The 0–1 integer variables are relaxed to continuous variables in the interval $[0, 1]$. Essentially, we solve the following program in the first step

$$\begin{aligned} \underset{\boldsymbol{m}}{\text{minimize}} \quad & E_{\mathcal{P},\mathcal{V}}(\boldsymbol{m}) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \le N_{C,max} \\ & N_M(\boldsymbol{m}) \le N_{M,max} \\ & 0 \le m_{i,j} \le 1 \end{aligned} \tag{8}$$

A locally optimal solution of Eq. 8, denoted by $\hat{\boldsymbol{m}}^*$, can be identified by a constrained non-linear optimization algorithm such as SQP. Intuitively, $\hat{\boldsymbol{m}}^*$ quantifies the relative importance of each singular value (and its corresponding singular vectors) in the approximation with a scalar between 0 and 1. The resulted target rank of the ith layer $r_i = \lfloor \mathit{\Sigma}\hat{\boldsymbol{m}}_{\boldsymbol{i}}{}^* \rceil$, where $\lfloor \cdot \rceil$ operator randomly rounds [25] a real number to an integer, such that

$$\lfloor x \rceil = \begin{cases} \lceil x \rceil & \text{with probability } x - \lfloor x \rfloor \\ \lfloor x \rfloor & \text{otherwise} \end{cases} \tag{9}$$

Here $\lfloor \mathit{\Sigma}\hat{\boldsymbol{m}}_{\boldsymbol{i}}{}^* \rceil$ serves as a surrogate for $\mathit{\Sigma}\boldsymbol{m_i^*}$. $\hat{\boldsymbol{m}}^{\boldsymbol{*}}$ scales the corresponding singular values. We therefore let $S_{\sigma,i}$ contain the j index of the r_i largest elements in set $\{\hat{m^*_{i,j}} \cdot \sigma_{i,j} | \forall_j\}$. A binary solution $\boldsymbol{m'}$ due to $\hat{\boldsymbol{m}}^*$ can be expressed as $m'_{i,j} = \mathbb{1}_{S_{\sigma,i}}(j)$ where $\mathbb{1}(\cdot)$ is the indicator function. If $\boldsymbol{m'}$ violates the constraints, the random rounding procedure is repeated until the constraints are satisfied.

In the second step, we incorporate the scaling effect of $\hat{\boldsymbol{m}}^{\boldsymbol{*}}$ in $\mathcal{P}$ as follows: for the ith convolutional layer, let $P_j \leftarrow \hat{m}^*_{ij} \cdot P_j$. The resulted low-rank approximation of the network is defined by $\{\boldsymbol{m'}, \mathcal{P}, \mathcal{V}\}$. With the structure of the approximated network determined by $\boldsymbol{m}'$, $\mathcal{P}$ and $\mathcal{V}$ can be further fine-tuned by simply running back-propagation.

3.1 Sequential Quadratic Programming

In the proposed method, Eq. 8 is solved using the SQP algorithm, which is arguably the most widely adopted method in solving constrained non-linear optimization problems [2]. At each SQP iteration, a linearly constrained quadratic programming (QP) subproblem is constructed and solved to move the current solution closer to a local minimum. To construct the QP subproblem, the gradient of the objective function and the constraints, as well as the Hessian have to be computed. The gradients can be readily computed by an automatic differentiation engine, such as TensorFlow. An approximation of the Hessian is iteratively refined by the BFGS algorithm [3] using the gradient information.

The scalability of the SQP algorithm is not a concern in our method. The number of decision variables (masking variables) in Eq. 8 is generally on the order of thousands, which is significantly smaller than the number of the weight parameters.

With a large training dataset, averaging over the entirety of the dataset to compute the gradient in each SQP iteration can be extremely time-consuming. In such cases, an estimation of the gradient by sub-sampling the training dataset has to be used in lieu of the true gradient. To address the estimation error of the gradients due to sub-sampling, we employed non-monotonic line search [4]. Non-monotonic line search ensures the line search iterations in the SQP algorithm can terminate properly despite the estimation error due to sub-sampled gradients. Note that a properly regularized Hessian estimation due to BFGS is positive semidefinite by construction, even with sub-sampled gradients [18]. Thus the QP subproblem is guaranteed to be convex.

Mathematically rigorous analysis of the convergence property of the SQP algorithm with sub-sampled gradient is the next step of this research. Recent theoretical results [6,7] could potentially provide insights into this problem. We empirically evaluated the numerical stability of SQP with sub-sampled gradients in Sect. 5.2

4 Prior Works

In this section, we thoroughly review the heuristics in the literature that are closely related to our proposed method. These heuristics will serve as the baseline to demonstrate the effectiveness of COBLA.

In [27], the target rank for each layer is identified by trial-and-error. Each trial involves fine-tuning the approximated network, which is highly time-consuming.

The following heuristic is discussed in [27] and earlier works: for the ith convolutional layer $S_{\sigma,i} = \{j | j \leq k_i\}$ is chosen such that the first k_i singular values and their corresponding singular vectors account for a certain percentage of the total variations. Thus, for the ith convolutional layer k_i is chosen to be the largest integer subject to

$$\sum_{j=1}^{k_i} \sigma_{i,j}^2 \leq \beta \cdot \sum_{\forall j} \sigma_{i,j}^2 \tag{10}$$

where β is the proportion of the total variations accounted by the low-rank approximation, and $\sigma_{i,j}$ is the jth largest singular value of the ith convolutional layer. It is obvious that the computation cost and the memory cost of the approximated network are monotonic functions of β. The largest β that satisfies the constraints in computation cost and memory cost, denoted by β^*, can be easily computed by bisection. Then the k_i value for each layer can be identified by plugging β^* into Eq. 10. We call this heuristic CPTV (Certain Percentage of Total Variation).

Another heuristics proposed in [30] identifies $S_{\sigma,i} = \{j | j \leq k_i\}$ by maximizing the following objective function subject to the constraints in computation cost.

$$\prod_{\forall i} (\sum_{j=1}^{k_i} \sigma_{i,j}^2) \tag{11}$$

In [30], a greedy algorithm is employed to approximately solve this program. We call this heuristic POS-Greedy (Product Of Sum-Greedy). See Sect. 2.4 of [30] for details. Due to the use of the greedy algorithm, only a single constraint can be considered by POS-Greedy.

We can improve the POS-Greedy heuristic by noting that the program in Eq. 11 can be solved with provable optimality and multiple constraints support by using the masking variable formulation. Equation 11 can be equivalently stated as

$$\begin{aligned} \underset{\boldsymbol{m}}{\text{maximize}} \quad & \prod_{\forall i} (\sum_{\forall j} m_{i,j} \cdot \sigma_{i,j}^2) \\ \text{subject to} \quad & N_C(\boldsymbol{m}) \leq N_{C,max} \\ & N_M(\boldsymbol{m}) \leq N_{M,max} \\ & m_{i,j} \in \{0, 1\} \end{aligned} \tag{12}$$

Note that the masking variables and the singular values can only take non-negative values, thus the objective in Eq. 12 is equivalent to maximizing the *geometric mean*. If the 0–1 integer constraint were omitted, the objective function and the constraints in Eq. 12 are concave in $\boldsymbol{m}$. Even with the 0–1 integer constraint, modern numerical optimization engines can efficiently solve this mixed integer program with provable optimality. The heuristic of exactly solving Eq. 12 is called POS-CVX (Product of Sum-Convex). In our experiment, we observe that the numerical value of the objective function due to POS-CVX is consistently 1.5 to 2 times higher than that due to POS-Greedy.

In [13], variational Bayesian matrix factorization (VBMF) [20] is employed to estimate the target rank. Given an observation V that is corrupted by additive noise $V = U + \sigma Z$, VBMF takes a Bayesian approach to identify a decomposition of matrix U whose rank is no larger than r, such that $U = BA^T$. We refer to this heuristic as R-VBMF (Rank due to VBMF). It is worth emphasizing that with R-VBMF, the user cannot arbitrarily set $N_{C,max}$ or $N_{M,max}$. Rather, the heuristic will decide the computation and the memory cost of the approximated network.

We also experimented with the low-rank signal recovery [5] to estimate the target rank for each layer. This groundbreaking result from the information theory community states that given a low-rank signal of unknown rank r which is contaminated by additive noise, one can optimally recover the low-rank signal in the Minimum-Square-Error (MSE) sense by truncating the singular values of the data matrix to $2.858 \cdot y_{med}$, where y_{med} is the median empirical singular values of the data matrix. This impressive result was not previously applied in the context of low-rank approximation of neural networks.

5 Numerical Experiments

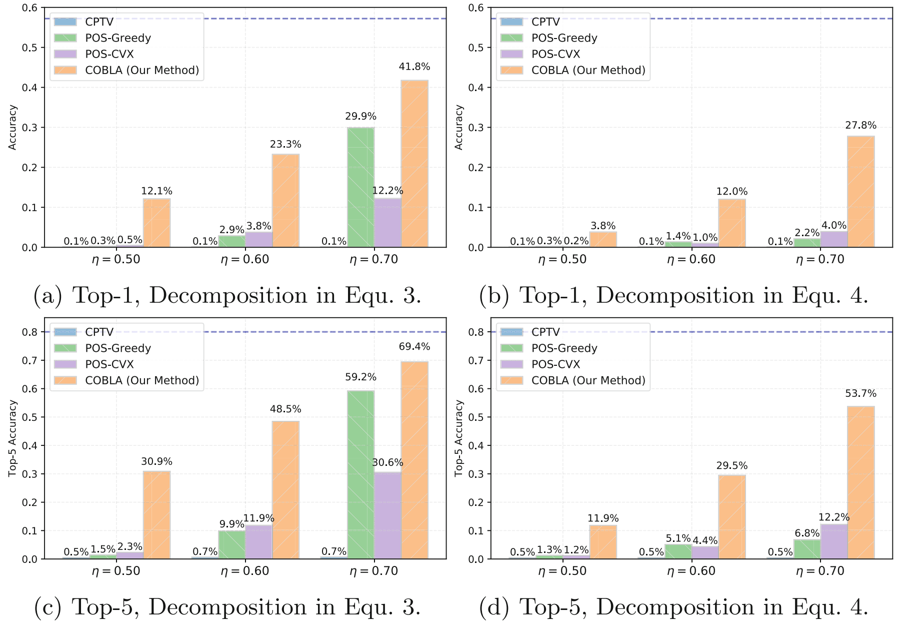
In this section, we compare the performance of COBLA to the previously published heuristics discussed in Sect. 4. Image classification experiments are performed using the SqueezeNet and the VGG-16 architecture on the ImageNet dataset [23]. SqueezeNet is a highly optimized architecture that achieves AlexNet-level accuracy with 50X parameter reduction. Further compressing such a compact and efficient architecture is a challenging task. We report the results using the decomposition scheme in both Eqs. 3 and 4.

The constraints to the computation cost and the memory cost of the approximated network, $N_{C,max}$ and $N_{M,max}$, are expressed in terms of the cost of the original network, denoted by $N_{C,O}$ and $N_{M,O}$. In the experiment $N_{C,max} = \eta \cdot N_{C,O}$ and $N_{M,max} = \eta \cdot N_{M,O}$, for $\eta = \{0.5, 0.6, 0.7\}$. The results in Fig. 2 are compiled by evaluating the approximated network due to each methods, before any fine-tuning is performed.

5.1 Effect of Fine-Tuning

We fine-tune the resulted network approximation due to POS-CVX and COBLA for 50 epochs. The experiment is repeated using the decomposition schemes in Eqs. 3 and 4. The hyperparameters used in the training phase are re-used in the fine-tuning phase, except for learning rate and momentum, which are controlled by the YellowFin optimizer [29]. The fine-tuning results are reported in Table 2.

Before fine-tuning, COBLA performs much better using the decomposition scheme in Eq. 3 (Fig. 2(a)(c)) than Eq. 4 (Fig. 2(b)(d)). Interestingly, the difference is reduced to within 1% after fine-tuning. This observation not only demonstrates that the effectiveness of COBLA is independent of the decomposition scheme, but also suggests that the choice of decomposition scheme is not critical to the success of low-rank approximation techniques.



(a) Top-1, Decomposition in Equ. 3.

(b) Top-1, Decomposition in Equ. 4.

(c) Top-5, Decomposition in Equ. 3.

(d) Top-5, Decomposition in Equ. 4.

Fig. 2. Comparison of Top-1 and Top-5 accuracy of the network approximation of SqueezeNet before fine-tuning. The right-hand side of the constraints in Eq. 8 are set to $N_{C,max} = \eta \cdot N_{C,O}$ and $N_{M,max} = \eta \cdot N_{M,O}$, where $N_{C,O}$ and $N_{M,O}$ are the computation cost and the memory cost of the original network without low-rank approximation. The Top-1 and Top-5 accuracy of the original SqueezeNet are 57.2% and 80.0% respectively.

Table 2. Accuracy of the approximated network at various constraint conditions using the SqueezeNet architecture on ImageNet dataset after 50 epochs of fine-tuning. The baseline method is POS-CVX. The Top-1 and Top-5 accuracy of the original SqueezeNet are 57.2% and 80.0% respectively.

	$N_{C,max}$	$N_{M,max}$	Decomposition scheme	Top-1 COBLA	Top-1 baseline	Top-5 COBLA	Top-5 baseline
1	$0.7 \cdot N_{C,O}$	$0.7 \cdot N_{M,O}$	Eq. 3	55.7%	−2.0%	79.2%	−1.1%
2	$0.7 \cdot N_{C,O}$	$0.7 \cdot N_{M,O}$	Eq. 4	55.4%	−2.4%	78.8%	−1.6%
3	$0.6 \cdot N_{C,O}$	$0.6 \cdot N_{M,O}$	Eq. 3	54.4%	−4.1%	78.2%	−2.7%
4	$0.6 \cdot N_{C,O}$	$0.6 \cdot N_{M,O}$	Eq. 4	54.3%	−3.8%	77.9%	−2.7%
5	$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Eq. 3	52.6%	−7.4%	77.0%	−5.7%
6	$0.5 \cdot N_{C,O}$	$0.5 \cdot N_{M,O}$	Eq. 4	51.7%	−5.5%	76.2%	−4.1%

5.2 Comparison with R-VBMF and Low-Rank Signal Recovery

Section 3.2 of [13] suggests that R-VBMF could function as a general solution for identifying the target rank of each layer in low-rank approximation of neural networks. We compared COBLA to R-VBMF. In our experiment, low-rank approximation is applied to all layers. This is different from the experiment setup in [13], where low-rank approximation is applied to a manually selected subset of layers. The reasoning behind applying R-VBMF to all layers is that if R-VBMF was indeed capable of recovering the true rank of the weight, it would just return the full rank of the weight if no low-rank approximation should be applied to a layer.

R-VBMF returns $N_{C,max} = 0.25 \cdot N_{C,O}$ and $N_{M,max} = 0.19 \cdot N_{M,O}$ on SqueezeNet using the decomposition scheme in Eq. 3. With such tight constraints, the accuracy of the approximated networks due to R-VBMF and COBLA both dropped to chance level before fine-tuning. Even after 10 epochs of fine-tuning, R-VBMF is still stuck close to chance level, while COBLA achieves a Top-1 accuracy of 15.9% and Top-5 accuracy of 36.2%. This experiment demonstrates that COBLA is a more effective method, even with severely constrained computation cost and memory cost. The low-rank signal recovery technique [5] also dramatically underestimated the target ranks.

The ineffectiveness of these rigorous signal processing technique in estimating the target rank in neural networks is not surprising. First of all, the non-linear activation functions between the linear layers are crucial to the overall dynamic of the network, but they cannot be easily considered in R-VBMF or low-rank signal recovery. Also, the low-rank approximation problem is not equivalent to recovering a signal from noisy measurements. Some unjustified assumptions have to be made regarding the distribution of the noise. More importantly, the target rank of each layer should not be analyzed in an isolated and layer-by-layer manner. It would be more constructive to study the approximation error with the dynamic of the entire network in mind. COBLA avoids the aforementioned pitfalls by taking a data-driven approach to address the unique challenges in this constrained optimization problem.

5.3 Effect of Sub-sampled Gradients in SQP Iterations

As discussed in Sect. 3.1, when the dataset is large, it is computationally prohibitive to exactly compute the gradient in each SQP iteration, and a sub-sampled estimation of the gradient has to be used. To investigate the effect of sub-sampled gradients in SQP, we conducted experiments using the NIN architecture [17] on the CIFAR10 dataset [14].

CIFAR10 is a small dataset on which we can afford to exactly compute the gradient in each SQP iteration. Although the CIFAR10 dataset is no longer considered a state-of-the-art benchmark, the 11-layer NIN architecture we used is relatively recent and ensures that the experiment is not conducted on a trivial example.

In Fig. 3, we compare the accuracy of the approximated network due to previously published heuristics and COBLA. The experiment using COBLA is conducted under two conditions. In the first case, labeled COBLA (sub-sampled gradient), 5% of the training dataset is randomly sampled to estimate the gradient in each SQP iteration. In the second case, labeled COBLA (exact gradient), the entire training dataset is used to exactly compute the gradient in each SQP

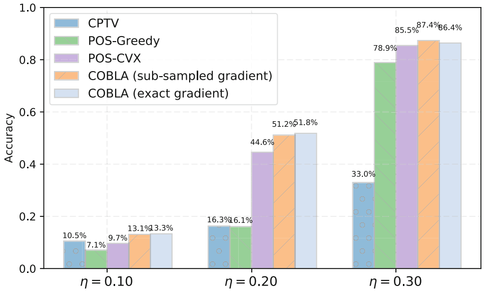


Fig. 3. Comparision of Top-1 accuracy of NIN architecture on the CIFAR10 dataset. The constraints are $N_{C,max} = \eta \cdot N_{C,O}$ and $N_{M,max} = \eta \cdot N_{M,O}$, for $\eta = \{0.1, 0.2, 0.3\}$. The accuracy of the original CIFAR-10 NIN is 91.9%.

iteration. As shown in Fig. 3, accuracies in the two cases are very similar (within 1%). This experiment provides some empirical evidence for the numerical stability of SQP with sub-sampled gradients.

5.4 COBLA on VGG-16

We compared COBLA to [27] using the VGG-16 architecture. We make the note that VGG-16 is an architecture that is over-parameterized by design. Such over-parameterized architectures are not suitable for studying model compression methods, as the intrinsic redundancy of the architecture would allow ineffective methods to achieve significant compression as well [10]. Optimized architectures that are designed to be computationally efficient (e.g. SqueezeNet) are more reasonable benchmarks [10]. The purpose of this experiment is to demonstrate the scalability of COBLA (VGG-16 is 22X larger than SqueezeNet in terms of computation cost). This experiment also provides a side-by-side comparison of COBLA to the results reported in [27].

In [27], the computation cost and the memory cost of the approximated network are $0.33 \cdot N_{C,O}$ and $0.36 \cdot N_{M,O}$ respectively. The resource allocation defined by the target rank of each layer is identified manually by trial-and-error. As shown in Table 3, COBLA *further* reduces the computation *and* the memory cost of the *compressed* VGG-16 in [27] by 12% with no accuracy drop (by 30% with negligible accuracy drop).

6 System Overview of COBLA

In Fig. 4, we present the system overview of COBLA. The centerpiece of COBLA is the SQP algorithm (which solves Eq. 8). The two supporting components are TensorFlow for computing gradients (of the empirical risk w.r.t. the masking variables) and MOSEK [19] for solving the convex QP in each SQP step. Given

Table 3. Comparision of COBLA to [27] using the VGG-16 architecture on ImageNet. The Top-5 accuracy of the original VGG-16 is 89.8%.

	Computation (reduction)	Memory (reduction)	Top-5 accuracy	Target rank of decomposed layers
Baseline [27]	$0.33 \cdot N_{C,O}$ -	$0.36 \cdot N_{M,O}$ -	89.8% -	`5, 24, 48, 48, 64, 128, 160 192, 192, 256, 320, 320, 320`
COBLA	$0.29 \cdot N_{C,O}$ (−12%)	$0.32 \cdot N_{M,O}$ (−12%)	89.8% (+0.0%)	`5, 17, 41, 54, 77, 109, 133 155, 180, 239, 274, 283, 314`
COBLA	$0.23 \cdot N_{C,O}$ (−30%)	$0.25 \cdot N_{M,O}$ (−30%)	88.9% (−0.9%)	`5, 16, 32, 48, 64, 81, 95 116, 126, 203, 211, 215, 232`

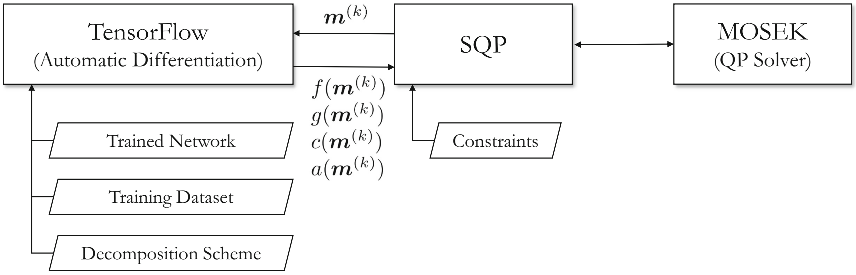

TensorFlow
(Automatic Differentiation)
$\boldsymbol{m}^{(k)}$
SQP
MOSEK
(QP Solver)
$f(\boldsymbol{m}^{(k)})$
$g(\boldsymbol{m}^{(k)})$
$c(\boldsymbol{m}^{(k)})$
$a(\boldsymbol{m}^{(k)})$
Trained Network
Training Dataset
Decomposition Scheme
Constraints

Fig. 4. System overview of COBLA. $\boldsymbol{m}^{(k)}$ is the value of the masking variables at the kth SQP iteration. $f(\boldsymbol{m}^{(k)})$ is the loss, $g(\boldsymbol{m}^{(k)})$ is the gradient of the loss with respect to the masking variables, $c(\boldsymbol{m}^{(k)})$ is the value of the constraint functions, and $a(\boldsymbol{m}^{(k)})$ is the Jacobian of the constraints.

$\boldsymbol{m}^{(k)}$, the value of the masking variables at the kth SQP iteration, TensorFlow computes the loss and the gradients based on the trained network and user-defined decomposition scheme. COBLA is available at https://github.com/chongli-uw/cobla.

6.1 Quantifying Parameter Redundancy of Each Layer

Given an approximated network identified by COBLA subject to the constraint of $N_{C,max} = 0.5 \cdot N_{C,O}$ and $N_{M,max} = 0.5 \cdot N_{M,O}$, we visualize the topology of the SqueezeNet and label the reduction of the computation cost of each layer in Fig. 5. For example, 28.9% of the computation cost of layer `conv1` is reduced by COBLA, so the computation cost of `conv1` in the approximated network is 71.1% of that in the original SqueezeNet network.

In the approximated network identified by COBLA, the allocation of the constrained computation resources is highly inhomogeneous. For most of the 1×1 layers, including the `squeeze` layers and the `expand/1` $\times$ `1` layers, the computation cost is not reduced at all. This indicates that there is less linear dependency in the 1×1 layers. However, the output layer `conv10` is an exception. `conv10` is a 1×1 layer that maps the high-dimensional output from previous layers to a vector of size 1000 (the number of classes in ImageNet). As shown in Fig. 5, 66% of the computation in `conv10` can be reduced. This coincides with the design choice that was identified manually in [8], where the author found that the output layer has high parameter redundancy.

In [24], it is hypothesized that the parameter redundancy of a layer is dependent on its relative position in the network and follows certain trends (increasing, decreasing, convex and concave are explored). Figure 5 indicates that the parameter redundancy of each layer is more complex than previously hypothesized and has to be analyzed on a case-by-case basis.

7 Conclusion

In this paper, we presented a systematic method named COBLA, to identify the target rank of each layer in low-rank approximation of a convolutional neural network, subject to the constraints in the computation cost and the memory cost of the approximated network. COBLA optimally allocates constrained computation resources into each layer. The key idea of the COBLA is in applying a binary masking variable to the singular values of the network weights to formulate a constrained 0–1 integer program. We empirically demonstrate that our method outperforms previously published works using the SqueezeNet and VGG-16 on the ImageNet dataset.

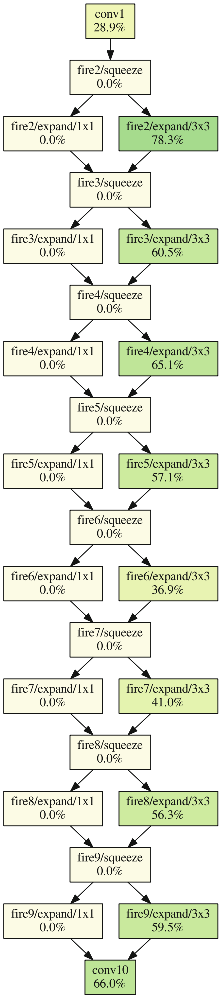


Fig. 5. Per-layer computation cost reduction in the approximated SqueezeNet.

Acknowledgment. The authors would like to thank the anonymous reviewers, particularly Reviewer 3, for their highly constructive advice. This work is supported by an Intel/Semiconductor Research Corporation Ph.D. Fellowship.

References

1. Alvarez, J.M., Salzmann, M.: Compression-aware training of deep networks. In: Neural Information Processing Systems (2017). http://papers.nips.cc/paper/6687-compression-aware-training-of-deep-networks.pdf

2. Boggs, P.T., Tolle, J.W.: Sequential quadratic programming. Acta Num. **4**, 1 (1995). https://doi.org/10.1017/S0962492900002518. http://www.journals.cambridge.org/abstract_S0962492900002518
3. Dai, Y.H.: Convergence properties of the BFGS algoritm. SIAM J. Optim. **13**(3), 693–701 (2002). https://doi.org/10.1137/S1052623401383455
4. Dai, Y.H., Schittkowski, K.: A sequential quadratic programming algorithm with non-monotone line search. Pac. J. Optim. **4**, 335–351 (2008)
5. Gavish, M., Donoho, D.L.: The optimal hard threshold for singular values is $4/\sqrt{3}$. IEEE Trans. Inf. Theory **60**(8), 5040–5053 (2014). https://doi.org/10.1109/TIT.2014.2323359. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6846297
6. Ge, R., Huang, F., Jin, C., Yuan, Y.: Escaping from saddle points - online stochastic gradient for tensor decomposition. J. Mach. Learn. Res. **40** (2015)
7. Gower, R.M., Goldfarb, D., Richtarik, P.: Stochastic block BFGS: squeezing more curvature out of data. In: International Conference on Machine Learning (2016). https://doi.org/10.1016/j.camwa.2005.08.006
8. Han, S., Mao, H., Dally, W.J.: Deep compression - compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (2016)
9. Ioannou, Y., Robertson, D., Shotton, J., Cipolla, R., Criminisi, A.: Training CNNs with low-rank filters for efficient image classification. In: International Conference on Learning Representations (2016). http://arxiv.org/abs/1511.06744
10. Jacob, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. ArXiv (2017). http://arxiv.org/abs/1712.05877
11. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: British Machine Vision Conference (BMVC) (2014). https://doi.org/10.5244/C.28.88, http://arxiv.org/abs/1405.3866
12. Keutzer, F.N.I., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Kurt: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. In: International Conference on Learning Representations (2017). https://doi.org/10.1007/978-3-319-24553-9, http://arxiv.org/abs/1602.07360
13. Kim, Y.D., Park, E., Yoo, S., Choi, T., Yang, L., Shin, D.: Compression of deep convolutional neural networks for fast and low power mobile applications. In: International Conference on Learning Representations (2016). http://arxiv.org/abs/1511.06530
14. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report (2009)
15. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In: International Conference on Learning Representations (2015). http://arxiv.org/abs/1412.6553
16. Lebedev, V., Lempitsky, V.: Fast ConvNets using group-wise brain damage. In: Conference on Computer Vision and Pattern Recognition (2016). https://doi.org/10.1109/CVPR.2016.280, http://arxiv.org/abs/1506.02515
17. Lin, M., Chen, Q., Yan, S.: Network in network. In: International Conference on Learning Representations (2013). https://doi.org/10.1109/ASRU.2015.7404828, http://arxiv.org/abs/1312.4400
18. Mokhtari, A.: Efficient methods for large-scale empirical risk minimization. Ph.D. thesis, University of Pennsylvania (2017)
19. MOSEK: the MOSEK optimization toolbox for MATLAB manual. Technical report (2017)

20. Nakajima, S., Tomioka, R., Sugiyama, M., Babacan, S.D.: Condition for perfect dimensionality recovery by variational bayesian PCA. J. Mach. Learn. Reas. **16**, 3757–3811 (2016)
21. Novikov, A., Vetrov, D., Podoprikhin, D., Osokin, A.: Tensorizing neural networks. In: Neural Information Processing Systems (2015), http://arxiv.org/pdf/1509.06569v1.pdf
22. Nowak, I.: Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming. Birkhäuser Basel (2005). https://doi.org/10.1007/3-7643-7374-1
23. Olga, R., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. (2015)
24. Park, E., Ahn, J., Yoo, S.: Weighted-entropy-based quantization for deep neural networks. In: Conference on Computer Vision and Pattern Recognition (2017). https://doi.org/10.1109/CVPR.2017.761
25. Raghavan, P., Tompson, C.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**(4), 365–374 (1987). https://doi.org/10.1007/BF02579324
26. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
27. Tai, C., Xiao, T., Zhang, Y., Wang, X., E, W.: Convolutional neural networks with low-rank regularization. In: International Conference on Learning Representations (2016). http://arxiv.org/abs/1511.06067
28. Yu, X., Liu, T., Wang, X., Tao, D.: On compressing deep models by low rank and sparse decomposition. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2017). https://doi.org/10.1109/CVPR.2017.15
29. Zhang, J., Mitliagkas, I., Ré, C.: YellowFin and the art of momentum tuning. arXiv preprint (2017). http://arxiv.org/abs/1706.03471
30. Zhang, X., Zou, J., Ming, X., He, K., Sun, J.: Efficient and accurate approximations of nonlinear convolutional networks. In: Conference on Computer Vision and Pattern Recognition (2015). http://arxiv.org/abs/1411.4229
31. Zhou, G.: Rank-constrained optimization: a Riemannian manifold approach. Ph.D. thesis, Florida State University (2015)