Deep Neural Network Quantization via Layer-Wise Optimization Using Limited Training Data

Shangyu Chen,[1] **Wenya Wang,**[1] **Sinno Jialin Pan**[1]
[1]Nanyang Technological University
schen025@e.ntu.edu.sg, wangwy@ntu.edu.sg, sinnopan@ntu.edu.sg

Abstract

The advancement of deep models poses great challenges to real-world deployment because of the limited computational ability and storage space on edge devices. To solve this problem, existing works have made progress to prune or quantize deep models. However, most existing methods rely heavily on a supervised training process to achieve satisfactory performance, acquiring large amount of labeled training data, which may not be practical for real deployment. In this paper, we propose a novel layer-wise quantization method for deep neural networks, which only requires limited training data (1% of original dataset). Specifically, we formulate parameters quantization for each layer as a discrete optimization problem, and solve it using Alternative Direction Method of Multipliers (ADMM), which gives an efficient closed-form solution. We prove that the final performance drop after quantization is bounded by a linear combination of the reconstructed errors caused at each layer. Based on the proved theorem, we propose an algorithm to quantize a deep neural network layer by layer with an additional weights update step to minimize the final error. Extensive experiments on benchmark deep models are conducted to demonstrate the effectiveness of our proposed method using 1% of CIFAR10 and ImageNet datasets. Codes are available in: https://github.com/csyhhu/L-DNQ

Introduction

Deep neural networks have been extensively employed with promising results in various applications especially in computer vision (Krizhevsky, Sutskever, and Hinton 2012; Simonyan and Zisserman 2014; He et al. 2016). However, high performance comes with huge cost brought by enormous amount of parameters and tremendous computational cost. Consider a very deep model which is fully well-trained and deployed, to use it for making predictions, most of the computations involve multiplications of a real-valued weight by a real-valued activation in forward propagation. These multiplications are expensive as they are all float-point to float-point multiplication operations. To alleviate this problem, a number of approaches have been proposed to compress deep models by pruning or quantization. Han et al. (2015) proposed to sparsify weights to reduce the number of multiplications directly. Courbariaux, Bengio, and David (2015) and Hubara et al. (2016) proposed to binarize weights to be in $\{\pm 1\}$. Rastegari et al. (2016) introduced a float value α_l known as the *scaling factor* in layer l, turning binarized weights as $\alpha_l \times \{\pm 1\}$. To accelerate inference, α_l is multiplied by layer input, such that weights become integer values $\{\pm 1\}$, converting float-point multiplication into float-point to integer computation. Li, Zhang, and Liu (2016) extended binary weights to ternary values, and Leng et al. (2017) further proposed to quantize the models with more bits in order to provide more flexibility. Apart from above mentioned training-based method, direct quantization methods also exist that don't require training data: Gong et al. (2014) clustered weights using Kmeans. Jacob et al. (2017), Vanhoucke, Senior, and Mao (2011) projected weights to nearest discrete points. Li et al. (2017) proposed convergence analysis of quantization training.

However, most prevailing low bits compression methods relied on a heavy training process with large amount of labeled data. Specifically, given a pre-trained full-precision deep neural network as the initial parameters, which is usually trained in a cloud environment, most methods carry the compression process in a supervised learning manner with sufficient training data by minimizing the error between the compressed network outputs and the ground-truth labels. However, practical scenarios pose more strict challenges. On the one hand, excessive training data for a specific application is inaccessible due to data privacy, especially in commercial models with high confidential requirement (Wu et al. 2016). On the other hand, many real cases require the compression conducted on edge devices, which greatly limit the storage space to deploy large amount of training data (e.g. ImageNet dataset occupies over 500Gb). Therefore, compression under **limited instances** is crucial in practice.

In these situations, existing training-based compression methods fail to proceed as they need abundant data to train a compressed model. Direct quantization methods show considerable gap. To overcome this limitation, we develop a quantization method, which only uses a small portion of training data while is still able to preserve the performance of the original network after quantization. The proposed method is named **L**ayer-wise/**L**imited training data **D**eep Neural **N**etwork **Q**uantization (L-DNQ), which aims to achieve the following goals: 1) For each layer, parameters are quantized while the layer output is similar to that of the original full-precision parameters. 2) The quantization can be obtained in a closed-form solution without gradient-based training. 3) There is a theoretical guarantee on the overall prediction performance after quantization. 4) The whole process consumes only a small portion of instances from training data (1% of training dataset in experiments).

To achieve the first goal, we formulate a quadratic optimization problem for each layer to minimize the error between the quantized layer output and the full-precision layer output. The quantized weights serve as discrete constraints, leading to a discrete optimization problem. This problem is solved by Alternative Direction Methods of Multipliers (ADMM) (Boyd et al. 2011) that decouples the continuous variables and discrete constraints to update them separately. ADMM is highly efficient and provides a closed-form solution to our quantization problem which contributes to our second goal. Regarding our third goal, inspired by (Aghasi et al. 2017; Dong, Chen, and Pan 2017) which provide theoretical bounds for network pruning, we derive a bound for network quantization. Most importantly, to avoid inefficient usage of data as is common in existing works, we approximate full-precision network under quantized constraints without the need for label supervision for optimization except a light retraining process. Experiments verify that the quantized networks learned by our proposed method with limited training data only bring a slight drop of the prediction performance, which achieves our fourth goal.

L-DNQ is able to achieve promising results with only limited training data compared with existing quantization methods. In practice, L-DNQ shows tremendous efficiency in data usage compared with some state-of-the-art methods.

Related Work

Regarding deep networks quantization, Courbariaux, Bengio, and David (2015) proposed network binarization through deterministic and stochastic rounding for parameters update after backpropagation. Hubara et al. (2016) and Rastegari et al. (2016) extended the idea by introducing binary activation. However, they all fail to recover a closing accuracy of the full-precision models. Li, Zhang, and Liu (2016) assumed a prior distribution for parameters to find an approximated threshold for ternarization. However, a proper prior is difficult to define. Lin, Zhao, and Pan (2017) approximated the full-precision weight with a linear combination of multiple binary weight bases. Zhu et al. (2016) set different scaling factors for positive and negative parameters ternarization. To update these factors, they first performed gradient descent, then gathered and averaged gradients for different factors based on some heuristic thresholds. Both of these methods relied on retraining and manually designed thresholds. Leng et al. (2017) modified the quantization retraining objective function using ADMM, which separates the processes on training real-valued parameters and quantizing the updated parameters. Zhou et al. (2017) proposed to incrementally quantize a portion of parameters based on weight partition. Polino, Pascanu, and Alistarh (2018) used distillation to assist training quantized network. All the above approaches relied on heavy supervised training process. Hou and Kwok (2018) extended neural network binarization from (Hou, Yao, and Kwok 2016) to quantization. By directly quantizing weights without training, Gong et al. (2014) clustered weights using Kmeans, Jacob et al. (2017), Vanhoucke, Senior, and Mao (2011) projected weights to nearest discrete points, Kundu et al. (2017) approximated full-precision weights by addition of a set of discrete values. Lin, Talathi, and Annapureddy (2016) converted pre-trained floating point models based on ignal-to-quantization-noise-ratio (SQNR). Still, such direct quantization can hardly cover performance drop. Other related works to ours include (Aghasi et al. 2017) and (Dong, Chen, and Pan 2017), which are also layer-wise deep compression approaches. However, they focused on pruning unimportant weights of the original networks rather than quantizing the real values of the weights into discrete bits.

Problem Statement and Preliminary

Problem Statement

Given a training set of n instances of d dimensions, $\{\mathbf{x}_j, y_j\}_{j=1}^n$, and a well-trained deep neural network with L layers (excluding the input layer)[1]. The well-trained network is considered as a reference or teaching network. Denote the input and the final output of the well-trained deep neural network by $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m_L \times n}$, respectively. For a layer l, we denote the input and the output of the layer by $\mathbf{Y}^{l-1} = [\mathbf{y}_1^{l-1}, ..., \mathbf{y}_n^{l-1}] \in \mathbb{R}^{m_{l-1} \times n}$ and $\mathbf{Y}^l = [\mathbf{y}_1^l, ..., \mathbf{y}_n^l] \in \mathbb{R}^{m_l \times n}$, respectively, where $\mathbf{y}_i^l$ can be considered as a representation of $\mathbf{x}_i$ in layer l, and $\mathbf{Y}^0 = \mathbf{X}$, $\mathbf{Y}^L = \mathbf{Y}$, $m_0 = d$. Using one forward-pass step, we have $\mathbf{Y}^l = \sigma(\mathbf{Z}^l)$, where $\mathbf{Z}^l = \bar{\mathbf{W}}_l^\top \mathbf{Y}^{l-1}$ with $\bar{\mathbf{W}}_l \in \mathbb{R}^{m_{l-1} \times m_l}$ denoting the matrix of full-precision parameters for layer l of the well-trained neural network, and $\sigma(\cdot)$ is the activation function. For convenience in presentation and proof, we define the activation function $\sigma(\cdot)$ as the rectified linear unit (ReLU). We further denote by $\bar{\mathbf{\Theta}}_l \in \mathbb{R}^{m_{l-1} m_l \times 1}$ the vectorization of $\bar{\mathbf{W}}_l$. For a well-trained neural network, $\mathbf{Y}^l$, $\mathbf{Z}^l$ and $\bar{\mathbf{\Theta}}_l$ are all fixed matrices and contain most information of the neural network. The goal of quantization is to discretize the values of all elements of $\bar{\mathbf{\Theta}}_l$ for each layer into a finite set $\mathbf{\Omega}_l$, e.g. symmetric: $\mathbf{\Omega}_l = \{-\alpha_l, 0, \alpha_l\}$ or random: $\mathbf{\Omega}_l = \{\alpha_l, \beta_l, \gamma_l\}$. We denote the quantized $\bar{\mathbf{\Theta}}_l$ by $\hat{\mathbf{\Theta}}_l$.

Alternative Direction Methods of Multipliers

ADMM is a widely-used optimization method (Aghasi et al. 2017; Takapoui et al. 2017). It combines the decomposability of dual ascent and convergence properties of the methods of multipliers. Given the following minimization problem:

$$\min f(\mathbf{x}) + g(\mathbf{z}), \quad \text{s.t.} h(\mathbf{x}, \mathbf{z}) = \mathbf{0}.$$

In ADMM, we first reformulate (1) with augmented Lagrangian as:

$$L_\rho(\mathbf{x}, \mathbf{y}, \mathbf{z}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^\top h(\mathbf{x}, \mathbf{z}) + \frac{\rho}{2} ||h(\mathbf{x}, \mathbf{z})||_2^2, \quad (1)$$

[1]For simplicity in presentation, we suppose the neural network is a feed-forward (fully-connected) network. However, our method works beyond feed-forward networks, and can be applied to deep models with CNN layers, such as VGG and ResNet.

where $\mathbf{y}$ is the Lagrangian multipliers. In practice, $\mathbf{y}$ is replaced by $\boldsymbol{\lambda} = (\frac{1}{\rho})\mathbf{y}$ (Boyd et al. 2011) to convert (1) to

$$L_\rho(\mathbf{x},\mathbf{y},\mathbf{z}) = f(\mathbf{x}) + g(\mathbf{z}) + \frac{\rho}{2}||h(\mathbf{x},\mathbf{z}) + \boldsymbol{\lambda}||_2^2 - \frac{\rho}{2}||\boldsymbol{\lambda}||^2. \quad (2)$$

We then breaks (2) into subproblems with respect to $\mathbf{x}, \mathbf{z}, \lambda$, respectively, and solve them iteratively using the following updates:

$$\text{Proximal step} : \mathbf{x}^{k+1} = \text{argmax}_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{z}^k, \boldsymbol{\lambda}^k) \quad (3)$$

$$\text{Projection step} : \mathbf{z}^{k+1} = \text{argmax}_{\mathbf{z}} L_\rho(\mathbf{x}^{k+1}, \mathbf{z}, \boldsymbol{\lambda}^k) \quad (4)$$

$$\text{Dual update Step} : \boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1} \quad (5)$$

Layer-Wise Quantization

Our proposed L-DNQ is a layer-wise cascade algorithm, where the output of a quantized layer is used as the input for quantizing the subsequent layer. Specifically, suppose one has quantized the well-trained network up to the $(l-1)$-th layer. Then, to quantize the l-th layer, we consider $\hat{\mathbf{Y}}^{l-1} = f(\mathbf{Y}^0; \hat{\boldsymbol{\Theta}}_{[1,\dots,l-1]})$ as the input, where $f(\mathbf{Y}^0; \hat{\boldsymbol{\Theta}}_{[1,\dots,l-1]})$ denotes the output of the $(l-1)$-th layer with the first $(l-1)$ layers being quantized, given input $\mathbf{Y}^0$. In the subsequent steps, L-DNQ aims at quantize the weights from layer l to the last layer L, denoted by $\hat{\boldsymbol{\Theta}}_{[l,\dots,L]}$, such that the divergence of the final layer output between quantized network and pre-trained network is minimized:

$$\min_{\hat{\boldsymbol{\Theta}}_{[l,\dots,L]}} ||f(\hat{\mathbf{Y}}^{l-1}; \hat{\boldsymbol{\Theta}}_{[l,\dots,L]}) - f(\mathbf{Y}^{l-1}; \bar{\boldsymbol{\Theta}}_{[l,\dots,L]})||_F^2, \quad (6)$$
$$\text{s.t.} \quad \hat{\boldsymbol{\Theta}}_{[l,\dots,L]} \in \boldsymbol{\Omega}_{[l,\dots,L]}.$$

Directly solving the above problem is difficult as the inputs to quantized network ($\hat{\mathbf{Y}}^{l-1}$) and the reference network ($\mathbf{Y}^{l-1}$) are different. Here, instead we propose to optimize the upper bound of the objective in (6), which is given by the following triangle inequality:

$$\begin{aligned} &||f(\hat{\mathbf{Y}}^{l-1}; \hat{\boldsymbol{\Theta}}_{[l,\dots,L]}) - f(\mathbf{Y}^{l-1}; \bar{\boldsymbol{\Theta}}_{[l,\dots,L]})||_F^2 \\ &\le \underbrace{||f(\hat{\mathbf{Y}}^{l-1}; \hat{\boldsymbol{\Theta}}_{[l,\dots,L]}) - f(\hat{\mathbf{Y}}^{l-1}; \bar{\boldsymbol{\Theta}}^{new}_{[l,\dots,L]})||_F^2}_{\text{Quantization}} \\ &+ \underbrace{||f(\hat{\mathbf{Y}}^{l-1}; \bar{\boldsymbol{\Theta}}^{new}_{[l,\dots,L]}) - f(\mathbf{Y}^{l-1}; \bar{\boldsymbol{\Theta}}_{[l,\dots,L]})||_F^2}_{\text{Weights Update}}, \quad (7) \end{aligned}$$

where $f(\hat{\mathbf{Y}}^{l-1}; \hat{\boldsymbol{\Theta}}_{[l,\dots,L]})$ presents the final output with quantized weights $\hat{\boldsymbol{\Theta}}_{[l,\dots,L]}$ and input $\hat{\mathbf{Y}}^{l-1}$. $\bar{\boldsymbol{\Theta}}^{new}_{[l,\dots,L]}$ is introduced as updated full precision weights learned during training. The objective in (6) is upper bounded by the summation of the **Quantization** term and the **Weights Update** term. To optimize its upper bound, we present an ADMM algorithm to minimize the Quantization term for each layer and a back-propagation algorithm to minimize the Weights Update term right after the quantization on each layer. These two procedures are conducted alternatingly until all layers are quantized.

Layer-Wise Error

During layer-wise quantization in layer l, which is the quantization term in (7), suppose $\hat{\boldsymbol{\Theta}}_l$ is the quantized parameters of $\bar{\boldsymbol{\Theta}}^{new}_l$ (when $l = 1$, $\bar{\boldsymbol{\Theta}}^{new}_1 = \bar{\boldsymbol{\Theta}}_1$; matrix form as $\bar{\mathbf{W}}^{new}_l$) to be learned. With the cascade input $\hat{\mathbf{Y}}^{l-1}$ and $\hat{\boldsymbol{\Theta}}_l$, we obtain a new outcome of the weighted sum before performing the activation function $\sigma(\cdot)$ for layer l, denoted by $\hat{\mathbf{Z}}^l$. Compared with the weighted sum of updated weights and cascade input: $\mathbf{Z}^l_* = (\bar{\mathbf{W}}^{new}_l)^\top \hat{\mathbf{Y}}^{l-1}$, we define an error function $E(\cdot)$ as:

$$E^l = E(\hat{\mathbf{Z}}^l) = \frac{1}{n}\left\|\hat{\mathbf{Z}}^l - \mathbf{Z}^l_*\right\|_F^2, \quad (8)$$

where $\|\cdot\|_F$ is the Frobenius Norm. Based on the definition of the error function (8), $E(\mathbf{Z}^l_*) = \frac{1}{n}\left\|\mathbf{Z}^l_* - \mathbf{Z}^l_*\right\|_F^2 = 0$. Using this property and following (Hassibi and Stork 1993; LeCun, Denker, and Solla 1990), (8) can be approximated by functional Taylor series as follows,

$$\begin{aligned} E^l &= E(\hat{\mathbf{Z}}^l) - E(\mathbf{Z}^l_*) \quad (9) \\ &= \left(\frac{\partial E^l}{\partial \boldsymbol{\Theta}_l}\right)^\top \delta\boldsymbol{\Theta}_l + \frac{1}{2}\delta\boldsymbol{\Theta}_l{}^\top \mathbf{H}_l \delta\boldsymbol{\Theta}_l + O(\|\delta\boldsymbol{\Theta}_l\|_2^3), \end{aligned}$$

where $\delta\boldsymbol{\Theta}_l$ denotes a perturbation of $\bar{\boldsymbol{\Theta}}^{new}_l$, $\mathbf{H}_l \equiv \partial^2 E^l / \partial\boldsymbol{\Theta}_l{}^2$ is the Hessian matrix w.r.t. $\boldsymbol{\Theta}_l$. It can be proven that with the error function defined in (8), the first (linear) term $\frac{\partial E^l}{\partial \boldsymbol{\Theta}_l}\Big|_{\boldsymbol{\Theta}_l = \bar{\boldsymbol{\Theta}}^{new}_l} = \mathbf{0}$, and $O(\|\delta\boldsymbol{\Theta}_l\|_2^3)$ vanishes. Hence we can rewrite (9) as $E^l = \frac{1}{2}\delta\boldsymbol{\Theta}_l{}^\top \mathbf{H}_l \delta\boldsymbol{\Theta}_l$. Since our goal is to quantize the weights to obtain $\hat{\boldsymbol{\Theta}}$, by replacing $\delta\boldsymbol{\Theta}_l$ with $\hat{\boldsymbol{\Theta}}_l - \bar{\boldsymbol{\Theta}}^{new}_l$, the final objective becomes:

$$\min_{\hat{\boldsymbol{\Theta}}_l} f(\hat{\boldsymbol{\Theta}}_l) = \tfrac{1}{2}(\hat{\boldsymbol{\Theta}}_l - \bar{\boldsymbol{\Theta}}^{new}_l)^\top \mathbf{H}_l (\hat{\boldsymbol{\Theta}}_l - \bar{\boldsymbol{\Theta}}^{new}_l), \quad (10)$$
$$\text{s.t.} \quad \hat{\boldsymbol{\Theta}}_l \in \boldsymbol{\Omega}_l,$$

where $\boldsymbol{\Omega}_l$ is a discrete set of all possible values of the quantized weights in layer l. To solve (10), which is a discrete optimization problem, we develop a ADMM-based algorithm.

Quantization with ADMM

In our problem setting, we apply ADMM to separately optimize continuous variables and discrete variables in (10). To be specific, we introduce an auxiliary parameter $\mathbf{G}$ and reformulate (10) as follows,

$$\min_{\hat{\boldsymbol{\Theta}},\mathbf{G}} f(\hat{\boldsymbol{\Theta}}) + I_{\boldsymbol{\Omega}}(\mathbf{G}), \quad \text{s.t.} \quad \hat{\boldsymbol{\Theta}} = \mathbf{G}, \quad (11)$$

where $I_{\boldsymbol{\Omega}}(\mathbf{G})$ is an indicator function that induces great penalty if $\mathbf{G} \notin \boldsymbol{\Omega}$. Here we drop the subscript l for simplification in presentation. By introducing $\mathbf{G}$ and applying the ADMM algorithm, the optimization problem (11) can be converted to

$$\begin{aligned} &L_\rho(\hat{\boldsymbol{\Theta}}, \mathbf{G}, \boldsymbol{\lambda}) \\ &\quad = f(\hat{\boldsymbol{\Theta}}) + I_{\boldsymbol{\Omega}}(\mathbf{G}) + \frac{\rho}{2}\|\hat{\boldsymbol{\Theta}} - \mathbf{G} + \boldsymbol{\lambda}\|_2^2 - \frac{\rho}{2}\|\boldsymbol{\lambda}\|_2^2. (12) \end{aligned}$$

(12) can be broken into 3 subproblems that are solved alternatingly and iteratively by repeating the following steps.

Proximal Step At iteration $k+1$, the proximal step involves the update on $\hat{\mathbf{\Theta}}$ via

$$\hat{\mathbf{\Theta}}^{k+1} = \arg\min_{\hat{\mathbf{\Theta}}} L_\rho(\hat{\mathbf{\Theta}}, \mathbf{G}^k, \boldsymbol{\lambda}^k), \tag{13}$$

where

$$L_\rho(\hat{\mathbf{\Theta}}, \mathbf{G}^k, \boldsymbol{\lambda}^k) = f(\hat{\mathbf{\Theta}}) + \frac{\rho}{2}\|\hat{\mathbf{\Theta}} - \mathbf{G}^k + \boldsymbol{\lambda}^k\|_2^2. \tag{14}$$

Since $f(\hat{\mathbf{\Theta}})$ is a quadric function with continuous variable, setting the gradient to $\mathbf{0}$ leads to the optimal solution by solving the following linear equation:

$$\big(\mathbf{H}+\text{diag}(\rho)\big)\hat{\mathbf{\Theta}}^{\boldsymbol{k+1}} = \mathbf{H}\bar{\mathbf{\Theta}}^{new}+\text{diag}(\rho)(\mathbf{G}^k-\boldsymbol{\lambda}^k). \tag{15}$$

This is far more efficient than gradient descent (Leng et al. 2017), and costs only a few seconds even if $\mathbf{H}$ is large. Besides, gradient descent requires fine-tuning a number of hyper-parameters and has unpredictable convergence. These issues are avoided using (15). As we can observe in (14), ρ acts as an importance weight for the discrete term. A large ρ leads $\hat{\mathbf{\Theta}}^{k+1}$ to approach discrete feasible solution $\mathbf{G}^k - \boldsymbol{\lambda}^k$, while a small ρ guides it to original weights $\bar{\mathbf{\Theta}}^{new}$.

Projection Step In projection step, we optimize $\mathbf{G}$ by solving the following optimization problem:

$$\min_{\mathbf{G}} \|\hat{\mathbf{\Theta}}^{k+1} - \mathbf{G} + \boldsymbol{\lambda}^k\|_2^2, \quad \text{s.t.} \quad \mathbf{G} \in \mathbf{\Omega}. \tag{16}$$

We define $\mathbf{V}^k = \hat{\mathbf{\Theta}}^{k+1} + \boldsymbol{\lambda}^k$, which is fixed in the projection step. The goal of (16) is to find $\mathbf{G}$ that is closest to $\mathbf{V}^k$ and lie in the discrete set $\mathbf{\Omega}$. Take $\mathbf{\Omega} = \{-\alpha, 0, \alpha\}$ as an example. We further denote by $\mathbf{Q} \in \{-1, 0, 1\}$ an intermediate variable such that $\mathbf{G} = g(\alpha, \mathbf{Q}) = \alpha \cdot \mathbf{Q}$. Here $g(\cdot)$ represents a mapping from integers to discrete real values. Thus, (16) can be rewritten as:

$$\min_{\mathbf{G},\alpha} \|\mathbf{V}^k - \alpha \cdot \mathbf{Q}\|_2^2, \quad \text{s.t.} \quad \mathbf{Q} \in \{-1, 0, 1\}, \tag{17}$$

which consists of two types of variables to be optimized: the scaling factor α that is continuous and the discrete constraints $\mathbf{Q}$. The problem is non-convex and non-smooth. We propose to solve it alternatingly: optimize α with $\mathbf{Q}$ fixed and vice versa. Specifically, given a fixed $\mathbf{Q}$, (17) is a quadric function w.r.t α, which can be easily solved by $\alpha = \frac{\mathbf{V}^\top\mathbf{Q}}{\mathbf{Q}^\top\mathbf{Q}}$. With α fixed, the optimal $\mathbf{Q}$ is obtained by projecting $\mathbf{V}^k$ to the nearest feasible solution as $\mathbf{Q} = \text{Proj}_{\{-1,0,1\}}\left(\frac{\mathbf{V}^k}{\alpha}\right)$, where $\text{Proj}_{\mathbf{\Omega}}(x)$ denotes the nearest point in the set $\mathbf{\Omega}$ for x. The projection step is efficient to compute. Empirical experiments show that α and $\mathbf{Q}$ could reach a stable range in less than 10 iterations. Finally we can have $\mathbf{G}^{k+1} = g(\alpha, \mathbf{Q})$.

Dual Update Step After obtaining $\hat{\mathbf{\Theta}}^{\boldsymbol{k+1}}$ and $\mathbf{G}^{k+1}$, the dual variable $\boldsymbol{\lambda}$ is updated using the following rule:

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \hat{\mathbf{\Theta}}^{k+1} - \mathbf{G}^{k+1}. \tag{18}$$

The ADMM-based layer-wise optimization saves much effort compared to existing gradient-descent-based methods. Computation complexity for the three steps are: $O(n^3)$,$O(n)$,$O(n)$, where n is the number of weights in one kernel. Most importantly, solving the optimization objective requires no label information, which is beneficial when labeled data is not available in real world applications.

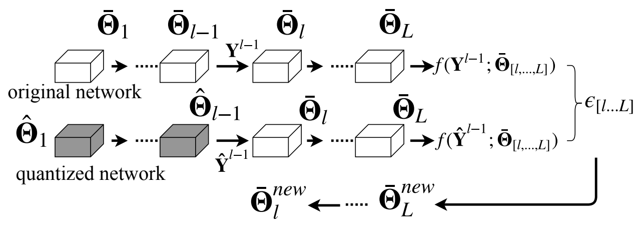
$\bar{\mathbf{\Theta}}_1$
$\bar{\mathbf{\Theta}}_{l-1}$
$\bar{\mathbf{\Theta}}_l$
$\bar{\mathbf{\Theta}}_L$
$\mathbf{Y}^{l-1}$
$f(\mathbf{Y}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]})$
original network
$\hat{\mathbf{\Theta}}_1$
$\hat{\mathbf{\Theta}}_{l-1}$
$\bar{\mathbf{\Theta}}_l$
$\bar{\mathbf{\Theta}}_L$
$\hat{\mathbf{Y}}^{l-1}$
$f(\hat{\mathbf{Y}}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]})$
$\epsilon_{[l...L]}$
quantized network
$\bar{\mathbf{\Theta}}_l^{new}$
$\bar{\mathbf{\Theta}}_L^{new}$

Figure 1: Weights Update: After layer $l-1$ is quantized from $\bar{\mathbf{\Theta}}_{l-1}$ to $\hat{\mathbf{\Theta}}_{l-1}$. Input $\mathbf{Y}^{l-1}$ and $\hat{\mathbf{Y}}^{l-1}$ are fed into two networks to generate $f(\mathbf{Y}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]})$ and $f(\hat{\mathbf{Y}}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]})$, respectively. Squared difference is backpropated to update weights in higher layers: $\bar{\mathbf{\Theta}}_{l,...,L}^{new}$.

Remaining Non-quantized Layers Update

In order to optimize the weights update term in (7), we first obtain the network where the previous $l-1$ layers are quantized while the remaining ones are not. Denote the difference or error of the final layer output between the current partially-quantized network and the original network by $\epsilon_{[l...L]} = \|f(\hat{\mathbf{Y}}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]}^{new}) - f(\mathbf{Y}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]})\|_F^2$. Here, we consider $f(\mathbf{Y}^{l-1}; \bar{\mathbf{\Theta}}_{[l,...,L]})$ as the ground-truth, and use back-propagation to learn $\bar{\mathbf{\Theta}}_{[l,...,L]}^{new}$. After we update $\bar{\mathbf{\Theta}}_{[l,...,L]}$ to $\bar{\mathbf{\Theta}}_{[l,...,L]}^{new}$, we can apply the ADMM algorithm introduced in the previous section to quantize layer l by replacing $\bar{\mathbf{\Theta}}_l$ with $\bar{\mathbf{\Theta}}_l^{new}$ in (10). Fig.1 demonstrates the weights update procedure.

Algorithm 1 Layer-wise Unsupervised Network Quantization

```
Require: $\bar{\mathbf{\Theta}} = \{\bar{\mathbf{\Theta}}_l\}_{1\le l\le L}$, $\boldsymbol{C} = \{\mu_l, \sigma_l, \gamma_l, \beta_l\}_{1\le l\le L}$:
    well-trained network, $\mathbf{X}$: training data
Ensure: $\hat{\mathbf{\Theta}} = \{\hat{\mathbf{\Theta}}_l\}_{1\le l\le L}$, $\hat{\boldsymbol{C}} = \{\hat{\mu}_l, \hat{\sigma}_l, \hat{\gamma}_l, \hat{\beta}_l\}_{1\le l\le L}$,
    $\{\boldsymbol{\alpha}_l\}_{1\le l\le L}$: quantized network
1: for $l = 1, 2, ...L$ do
2:     Calculate Hessian matrix $\mathbf{H}_l$ of layer $l$ from $\bar{\mathbf{\Theta}}_l^{new}$
       and $\mathbf{X}$. (If $l = 1$, $\bar{\mathbf{\Theta}}_1^{new} = \bar{\mathbf{\Theta}}_1$)
3:     Perform ADMM to obtain quantized weight $\hat{\mathbf{\Theta}}_l$ by
       solving (12)
4:     Learn $\bar{\mathbf{\Theta}}_{[l+1,...,L]}^{new}$ by minimizing $\epsilon_{[l+1...L]} =$
       $\|f(\hat{\mathbf{Y}}^l; \bar{\mathbf{\Theta}}_{[l+1,...,L]}^{new}) - f(\mathbf{Y}^l; \bar{\mathbf{\Theta}}_{[l+1,...,L]})\|_F^2$, and up-
       date $\bar{\mathbf{\Theta}}_{[l+1,...,L]} \leftarrow \bar{\mathbf{\Theta}}_{[l+1,...,L]}^{new}$.
5: end for
6: Retrain.
```

L-DNQ in Practice

The overall procedure of L-DNQ is summarized in Algorithm 1. In the algorithm, $\boldsymbol{C}$ denotes the set of parameters for batch normalization, which is also updated in the quantized network. Steps 1-5 corresponds to the quantization procedure using ADMM. After quantization, models can be further boosted by retraining: using label information to fine-tune parameters in batch normalization layers and unquantized layer. In practice, we adopt approximated Hessian calculation in (Dong, Chen, and Pan 2017), which only needs to calculate one block matrix of the original Hessian, making our Hessian computation suitable to handle. About 200 images are sufficient to generate Hessian matrix for each layer.

Theoretical Analysis

Recall that our goal is to control the consistency of the network's final output $\mathbf{Y}^L$ before and after quantization. In the following, we show how the layer-wise errors propagate to the final output layer. We prove that the accumulated error over multiple layers is upper bounded by a constant.

Theorem 1. *Given a quantized network via layer-wise quantization introduced in Section , each layer has its own layer-wise error ε^l for $1 \leq l \leq L$, then the accumulated error of ultimate network output $\hat{\varepsilon}^L = \frac{1}{\sqrt{n}}\|\hat{\mathbf{Y}}^L - \mathbf{Y}^L\|_F$ obeys:*

$$\hat{\varepsilon}^L \leq \sum_{k=1}^{L-1}\left(\prod_{l=k+1}^{L}\mathcal{A}\sqrt{\varepsilon^k}\right) + \sqrt{\varepsilon^L}, \tag{19}$$

where $\hat{\mathbf{Y}}^l = \sigma(\hat{\mathbf{W}}_l^\top \hat{\mathbf{Y}}^{l-1})$ for $2 \leq l \leq L$ denotes the "accumulated pruned output" of layer l. The term $\mathcal{A} = \|\hat{\mathbf{\Theta}}_l\|_F$ is upper bounded according to different quantizations.

Consider $\mathbf{\Omega} = \{\pm\alpha, 0\}$ as an example. It can be proven that $\mathcal{A} \leq \alpha^2 \times (m_l \times m_{l-1})$. Moreover, empirical experiments shows that 0 occupies 50%-70% of the quantized parameters, thus $\mathcal{A}$ is much smaller in practice. In summary, Theorem 1 shows that: 1) Layer-wise error for layer l will be scaled by continued multiplication of parameters' Frobenius Norm over the following layers when it propagates to final output. For quantization, this Frobenius Norm is upper bounded by a constant determined by the quantization intervals. 2) The final error of the ultimate network output is bounded by the weighted sum of layer-wise errors.

Proof. We prove Theorem 1 via induction. First, for $l = 1$:

$$\hat{\varepsilon}^l = \varepsilon^l = \sqrt{\delta E^l}. \tag{20}$$

Then suppose that Theorem 1 holds up to layer l:

$$\hat{\varepsilon}^l \leq \sum_{h=1}^{l-1}(\prod_{k=h+1}^{l}\|\hat{\mathbf{\Theta}}_k\|_F\sqrt{\delta E^h}) + \sqrt{\delta E^l}. \tag{21}$$

In order to show that (21) holds for layer $l+1$ as well, we refer to $\tilde{\mathbf{Y}}^{l+1} = \sigma(\hat{\mathbf{W}}_{l+1}^\top \mathbf{Y}^l)$ as 'layer-wise quantized output', where the input $\mathbf{Y}^l$ is fixed as the same as the originally well-trained network. An accumulated input $\hat{\mathbf{Y}}^{l+1} = f(\mathbf{Y}^0; \hat{\mathbf{\Theta}}_{[1,\ldots,l]})$, and have the following theorem. □

Theorem 2. *Consider layer $l+1$ in a quantized deep network, the difference between its accumulated quantized output, $\hat{\mathbf{Y}}^{l+1}$, and layer-wise quantized output, $\tilde{\mathbf{Y}}^{l+1}$, is bounded by:*

$$\|\tilde{\mathbf{Y}}^{l+1} - \hat{\mathbf{Y}}^{l+1}\|_F^2 \leq \sqrt{n}\|\hat{\mathbf{\Theta}}^{l+1}\|_F^2\hat{\varepsilon}^l. \tag{22}$$

By using (20), (22) and the triangle inequality, we are now able to extend (21) to layer $l+1$:

$$\begin{aligned}\hat{\varepsilon}^{l+1} &= \frac{1}{\sqrt{n}}\|\hat{\mathbf{Y}}^{l+1} - \mathbf{Y}^{l+1}\|_F^2 \\ &\leq \frac{1}{\sqrt{n}}\|\tilde{\mathbf{Y}}^{l+1} - \hat{\mathbf{Y}}^{l+1}\|_F^2 + \frac{1}{\sqrt{n}}\|\tilde{\mathbf{Y}}^{l+1} - \mathbf{Y}^{l+1}\|_F^2 \\ &\leq \sum_{h=1}^{l}\left(\prod_{k=h+1}^{l+1}\|\hat{\mathbf{\Theta}}^{k+1}\|_F^2 \cdot \sqrt{\delta E^h}\right) + \sqrt{\delta E^{l+1}}.\end{aligned}$$

Finally, we prove that (21) holds up for all layers, and Theorem 1 is a special case when $l = L$. We further set $\mathcal{A} = \|\hat{\mathbf{\Theta}}_l\|_F$, which is the Frobenius norm of quantized weights. $\mathcal{A}$ is upper bounded according to different quantization. Consider $\mathbf{\Omega} = \{\pm\alpha, 0\}$ as an example. It can be proven that $\mathcal{A} \leq \alpha^2 \times (m_l \times m_{l-1})$.

Experiment

Experimental Setup

We conduct comparison experiments with the following baseline approaches: 1) Extremely Low Bit Neural Network (ExNN) (Leng et al. 2017) 2) Trained Ternary Quantization (TTQ) (Zhu et al. 2016), 3) Incremental Network Quantization (INQ) (Zhou et al. 2017) 4) Loss-Aware weight Ternarized network (LAT) (Hou and Kwok 2018). 5) Compressing Deep Convolutional Networks using Vector Quantization (VQ) (Gong et al. 2014). 6) Direct Quantization (DQ) which is commonly used in production (Jacob et al. 2017), (Vanhoucke, Senior, and Mao 2011). 7) Ternary Residual Networks (TRN) (Kundu et al. 2017). 8) Model compression via distillation and quantization (DistilQuant) (Polino, Pascanu, and Alistarh 2018). Two benchmark datasets are used including ImageNet ILSVRC-2012 and CIFAR-10. Regarding deep architectures, we experiment with ResNet-18 (He et al. 2016), AlexNet [2] (Krizhevsky, Sutskever, and Hinton 2012) on ImageNet dataset, and with CIFARNet[3], VGG, WRN[4], ResNet-20, ResNet-32, ResNet-56 on CIFAR-10. All these deep models are well trained at the first place. Due to different deep learning framework, performance of well-trained networks show slight difference.

As L-DNQ pioneers in limited-instance quantization, few works have been published for comparison. VQ conducts compression based on original pre-trained model by using K-means clustering directly; DQ essentially projects full-precision weights into nearest discrete points; TRN approximated full-precision weights by a combination of quantized weights. After quantization, training instances are used

[2]AlexNet with batch normalization layers is adopted.

[3]The network architecture is: $(2 \times 128C3) - MP2 - (2 \times 256C3) - MP2 - (2 \times 512C3) - MP2 - (2 \times 1024FC) - 10SVM$, where $C3$ is a 3×3 ReLU convolution layer, $MP2$ is a 2×2 max-pooling layer.

[4]Widen factor:20, depth:28, dropout rate:0.3

for retraining and fine-tunning. These methods can be considered as baselines for limited-instance quantization. For fair comparison with training-based quantization, we reduce training data to 1% of the original training dataset. Specifically, we re-implement ExNN, TTQ, INQ, LAT and DistilQuant to generate their reported results and then apply the same sets of parameters to produce the results with limited instances. 500 training instances in CIFAR-10 and 12,800 in ImageNet are randomly sampled to simulate the scenario of limited instances. All experiments are conducted 5 times and the average result is reported. Note that all methods use different initial pre-trained models. For fair comparison, we record the percentage of the improvements for these quantized models over their corresponding pre-trained models, which is positive for improvement while negative for degradation after quantization (The higher the better).

L-DNQ adopts the following quantization intervals: $\mathbf{\Omega}_l = \alpha_l \times \{0, \pm 2^0, \pm 2^1, \pm 2^2 ... \pm 2^b\}$ for each layer. Using power of 2 is efficient for inference, because quantized weights can be stored and calculated as integer $(\pm 1, 2, ...)$, with layer output multiplying by α_l to retrieve the actual layer output. To facilitate notation, we use $(2b+3)$-bit to denote the above quantization set, which can be interpreted as the total number of different values in the quantization set, e.g., $\alpha \times \{0, \pm 2^0, \pm 2^1, \pm 2^2\}$ is denoted as 7-bit. In INQ (Zhou et al. 2017), (Jacob et al. 2017) and (Kundu et al. 2017), a different presentation form for bits is used. Here we convert the number of bits using our notation for fair comparison.

Network	Method	bits	Imp*(%)	FP**
ResNet20	TTQ	3	-77.25	91.77
	INQ	15	-48.48	90.02
	ExNN	3	-11.15	91.5
	VQ	3	-11.27	
	DQ	3	-19.92	
	L-DNQ	3	**-4.30**	
ResNet32	TTQ	3	-79.99	92.33
	INQ	15	-48.02	86.83
	ExNN	3	-12.03	92.13
	VQ	3	-8.98	
	DQ	3	-21.07	
	L-DNQ	3	**-3.66**	
ResNet56	TTQ	3	-80.64	93.20
	INQ	15	-15.84	93.40
	ExNN	3	-12.15	92.66
	VQ	3	-11.43	
	DQ	3	-18.67	
	L-DNQ	3	**-3.49**	
CIFARNet	LAT	3	-11.62	89.62
	VQ	3	-11.83	92.27
	DQ	3	-21.72	
	L-DNQ	3	**-1.96**	
VGG***	DistilQuant	3	-53.9	90.77
	L-DNQ	3	**-1.47**	89.42
WRN	DistilQuant	3	-6.57	92.25
	L-DNQ	3	**-2.22**	91.43

Table 1: Comparison on CIFAR-10. All methods use 1% (500 images) of training instances. * indicates improvement. ** represents **F**ull **P**recision (pre-trained model) Accuracy. *** is a VGG-like model adopted by DistilQuant.

Network	Method	bits	Improvement(%)	FP Accuracy
ResNet18	TTQ	3	-69.48/-88.49	69.6/89.2
	INQ	15	-61.27/-64.22	68.27/88.69
	ExNN	3	-43.53/-37.82	69.76/89.02
	VQ	3	-35.69/-29.08	
	DQ	3	-61.22/-65.64	
	L-DNQ	3	**-16.43/-10.67**	
	DQ	8	-56.78/-58.92	
		16	-13.81/-8.68	
		32	-2.82/-1.51	
	L-DNQ	9	**-2.73/-0.90**	
ResNet34	DistilQuant	3	-32.03/24.3	56.55/79.09
	L-DNQ	3	**-29.31/18.37**	
AlexNet[1]	TTQ	3	-56.18/-78.26	57.2/80.3
	DistilQuant		-55.07/-72.79	56.55/79.09
	ExNN		-28.34/-26.06	58.34/80.80
	VQ		-35.95/-36.06	
	DQ		-57.04/-76.49	
	L-DNQ		**-17.78/-14.00**	
AlexNet	INQ	15	-42.82/-46.83	57.24/79.80
	TRN	12.5	≈ -1	N.A.
	ExNN	9	-13.47/-10.33	58.34/80.80
	VQ		-5.94/-4.22	
	DQ		-7.84/-5.72	
	L-DNQ		**-0.88/-0.57**	

Table 2: Comparison on ImageNet. All methods use 1% (12,800 images) training instances. AlexNet[1]: in TTQ, the weights of the first and final layer remain full precision. L-DNQ, ExNN, DQ, VQ, DistilQuant are under the same setting.

Overall Experimental Results and Analysis

On CIFAR-10, we compare our method with ExNN, TTQ, INQ, LAT, VQ, DQ, DistilQuant in Table 1. ExNN incorporated ADMM quantization in training. TTQ ternarized the full-precision model into 3 different values: $\{\beta_l, 0, \alpha_l\}$ in layer l. INQ converted weights into either power of two or zero. LAT quantized weights into 3 bits. DistilQuant utilize distillation to train quantized model. We reproduce TTQ, INQ experiments on ResNet-20, ResNet-32, ResNet-56, LAT experiments on CIFARNet and DistilQuant on VGG, WRN with the source code released by (Zhu et al. 2016) , (Zhou et al. 2017) , (Hou and Kwok 2018) and (Polino, Pascanu, and Alistarh 2018), respectively. ExNN is reimplemented by us since its source code is not released. After VQ and DQ generate quantized weights, we use limited instances for retraining. As Table 1 shows, all training-based methods (ExNN, TTQ, INQ, DistilQuant) experienced great degradation using limited instances. VQ and DQ performed slightly better, but still showed considerable drop. As a contrast, L-DNQ can preserve original performance, even under a dramatic reduction of training data down to 1%.

On ImageNet, which is a much larger dataset than CIFAR-10, we compare our model with ExNN, INQ, TTQ, VQ, DQ, DistilQuant using ResNet18/34 and AlexNet, with the results showing in Table 2. Similar to CIFAR-10, different methods use different pre-trained models and the improvements are shown. When only 1% instances are used, L-DNQ outperforms other methods by a large margin. Training-based methods suffer under-fitting problem in limited-instance scenario. Direct quantization-based methods fail to capture data distribution for better compression.

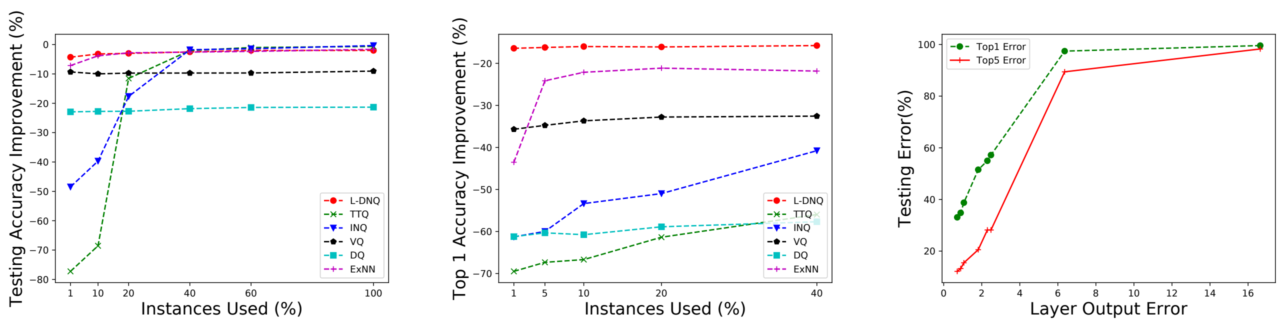

Figure 2: Fig.(a)/(b): Performance among L-DNQ, ExNN, TTQ, INQ, VQ, DQ using ResNet20/18 in CIFAR10/ImageNet with increasing instances. X-axis presents portion of training data used, Y-axis represents performance improvement after quantization (Higher the better). Fig.(c): Performance under different layer output error. X-axis presents final layer output error, Y-axis represents testing error after quantization (the lower the better).

Network	3-bit	5-bit	7-bit	9-bit	Full Precision
ResNet18	-16.43/-10.67	-8.61/-4.92	-4.67/-2.40	-2.73/-0.90	69.76/89.02
ResNet34	-29.31/-18.37	-11.22/-6.10	-3.69/-1.88	-2.10/-0.87	73.30/91.42
ResNet50	-19.66/-11.32	-7.55/-4.10	-2.79/-1.24	-1.71/-0.53	76.15/92.87

Table 3: Overall experimental results of L-DNQ in various models using 1% of ImageNet dataset. Column 2-5 represent the quantization improvement (Top1/Top5) under different bits quantization. The last column represents full precision accuracy.

Dataset	Network	Method	kbits	TopK	Improvement	Full Precision (%)
CIFAR10	ResNet20	L-DNQ$^-$	3	1	-13.66	91.5
		L-DNQ	3	1	-4.3	
ImageNet	AlexNet	L-DNQ$^-$	3	1/5	-56.80/-79.30	58.34/80.80
		L-DNQ	3	1/5	-17.78/-14.00	

Table 4: Comparison between L-DNQ and L-DNQ$^-$.

For TRN, since it doesn't release the source code, we compare with its reported result using AlexNet. As Table 2 shows, L-DNQ achieves much better performance compared with other methods using 3 bits. In practice, DQ uses more bits in quantization, hence we also compare our model using 9 bits with DQ using more bits from 8 to 32 in ResNet18 and other baselines with more bits in AlexNet. Clearly, the baseline models retrieve better performances as the number of bits increases, but still hardly achieve comparable performances as L-DNQ, which almost approximates the full-precision model with 9 bits.

We also compare L-DNQ, ExNN, TTQ, INQ, VQ and DQ in ResNet20/18 using increasing number of training instances on CIFAR-10 and ImageNet. As Fig 2(a) and 2(b) show, L-DNQ maintains high performances even when only a few training instances are used, while training-based methods degrade severely if data is scarce. We conjecture that L-DNQ minimizes the divergence from original network under quantized constraints instead of regenerating a new network given label supervisions, which enables L-DNQ to utilize much less data to preserve the original performance.

Analysis of L-DNQ

Bits' Effect Towards Performance We conduct experiments on various quantization levels on ImageNet using 1% instances. As Table 3 shows, the prediction accuracy increases as the number of bits increases. It can be observed that L-DNQ approximately recovers the original accuracy under 9-bit quantization.

Layer Output Error V.S. Performance L-DNQ aims at minimizing the final layer output error between original and quantized networks. It is interesting to explore the relationship between layer output error and performance of quantized network. We measure the performance of quantized ResNet18 network on ImageNet dataset under different final output errors. As Fig. 2(c) shows, testing error increases as the final output error increases. Empirically, testing error is positively correlated with final output error. Hence, by minimizing the final output error, L-DNQ is capable of attaining good performance.

Effectiveness of Weights Update To verify the effectiveness of weights update, we generate another baseline called L-DNQ$^-$ by removing weights update in L-DNQ, which can be considered as a reduction of the proposed L-DNQ. Comparison results between L-DNQ and L-DNQ^{-} are shown in Table 4: weights update in L-DNQ brings significant performance gain by narrowing the divergence between quantized network and the unquantized one.

Conclusion

In this paper, we propose a novel layer-wise quantization framework, L-DNQ, and provide a theoretical guarantee on the overall error. We conduct extensive experiments on two benchmark datasets to demonstrate that L-DNQ is able to quantize deep models without big performance drop using only limited training data Therefore, L-DNQ is very effective in the cases where training instances are difficult to attain because of data privacy issue and when quantization is deployed in edge devices with limited storage space.

Acknowledgements

This work is supported by NTU Singapore Nanyang Assistant Professorship (NAP) grant M4081532.020, Singapore MOE AcRF Tier-1 grant 2016-T1-001-159 and Singapore MOE AcRF Tier-2 grant MOE2016-T2-2-060.

References

Aghasi, A.; Abdi, A.; Nguyen, N.; and Romberg, J. 2017. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *NIPS*. 3180–3189.

Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; and Eckstein, J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.* 3(1):1–122.

Courbariaux, M.; Bengio, Y.; and David, J. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR* abs/1511.00363.

Dong, X.; Chen, S.; and Pan, S. 2017. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NIPS*. 4860–4874.

Gong, Y.; Liu, L.; Yang, M.; and Bourdev, L. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *NIPS*, 1135–1143.

Hassibi, B., and Stork, D. G. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 164–171.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Hou, L., and Kwok, J. T. 2018. Loss-aware weight quantization of deep networks.

Hou, L.; Yao, Q.; and Kwok, J. T. 2016. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *NIPS*, 4107–4115.

Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; and Kalenichenko, D. 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv preprint arXiv:1712.05877*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105.

Kundu, A.; Banerjee, K.; Mellempudi, N.; Mudigere, D.; Das, D.; Kaul, B.; and Dubey, P. 2017. Ternary residual networks. *arXiv preprint arXiv:1707.04679*.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *NIPS*. 598–605.

Leng, C.; Li, H.; Zhu, S.; and Jin, R. 2017. Extremely low bit neural network: Squeeze the last bit out with admm. In *AAAI*.

Li, H.; De, S.; Xu, Z.; Studer, C.; Samet, H.; and Goldstein, T. 2017. Training quantized nets: A deeper understanding. In *NIPS*, 5811–5821.

Li, F.; Zhang, B.; and Liu, B. 2016. Ternary weight networks. *arXiv preprint arXiv:1605.04711*.

Lin, D.; Talathi, S.; and Annapureddy, S. 2016. Fixed point quantization of deep convolutional networks. In *ICML*, 2849–2858.

Lin, X.; Zhao, C.; and Pan, W. 2017. Towards accurate binary convolutional neural network. In *NIPS*, 344–352.

Polino, A.; Pascanu, R.; and Alistarh, D. 2018. Model compression via distillation and quantization.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 525–542. Springer.

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

Takapoui, R.; Moehle, N.; Boyd, S.; and Bemporad, A. 2017. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control* 1–11.

Vanhoucke, V.; Senior, A.; and Mao, M. Z. 2011. Improving the speed of neural networks on cpus. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; and Cheng, J. 2016. Quantized convolutional neural networks for mobile devices. In *CVPR*, 4820–4828.

Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; and Chen, Y. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*.

Zhu, C.; Han, S.; Mao, H.; and Dally, W. J. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*.