# Extremely Low Bit Neural Network: Squeeze the Last Bit Out with ADMM

Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, Rong Jin
Alibaba Group, Hang Zhou, China
{lengcong.lc, zesheng.dzs, lihao.lh, shenghuo.zhu, jinrong.jr}@alibaba-inc.com

**Abstract**

Although deep learning models are highly effective for various learning tasks, their high computational costs prohibit the deployment to scenarios where either memory or computational resources are limited. In this paper, we focus on compressing and accelerating deep models with network weights represented by very small numbers of bits, referred to as **extremely low bit neural network**. We model this problem as a discretely constrained optimization problem. Borrowing the idea from Alternating Direction Method of Multipliers (ADMM), we decouple the continuous parameters from the discrete constraints of network, and cast the original hard problem into several subproblems. We propose to solve these subproblems using extragradient and iterative quantization algorithms that lead to considerably faster convergency compared to conventional optimization methods. Extensive experiments on image recognition and object detection verify that the proposed algorithm is more effective than state-of-the-art approaches when coming to extremely low bit neural network.

## 1 Introduction

These years have witnessed the success of convolutional neural networks (CNNs) in a wide range computer vision tasks, such as image classification, object detection and segmentation. The success of deep learning largely owes to the fast development of computing resources. Most of the deep learning models are trained on high-ended GPUs or CPU clusters. On the other hand, deeper networks impose heavy storage footprint due to the enormous amount of network parameters. For example, the 16-layers VGG involves 528 MBytes of model parameters. Both the high computational and storage cost become impediments to popularize the deep neural networks to scenarios where either memory or computational resources are limited. The great interest to deploy deep learning systems on low-ended devices motives the research in compressing deep models to have smaller computation cost and memory footprints.

Considerable efforts have been mounted to reduce the model size and speed up the inference of deep models. Denil et al. pointed out that network weights have a significant

redundancy, and proposed to reduce the number of parameters by exploiting the linear structure of network [1], which motivated a series of low-rank matrix/tensor factorization based compression algorithms, e.g. [2, 3, 4]. Alternatively, multiple studies were devoted to discritizing network weights using vector quantization methods [5, 6], which often outperformed the matrix/tensor factorization based methods [6]. Han et al. presented the deep compression method that integrates multiple compression methods to achieve a large reduction in model size [7]. Another line of work for model compression is to restrict network weights to low precision with a few bits. The advantage of this restriction is that an expensive floating-point multiplication operation can now be replaced by a sequence of cheaper and faster binary bit shift operations. This not only reduces the memory footprints but also accelerates the computation of the network. These approaches work well when pretrained weights are quantized into 4-12 bits [8, 9, 10, 11]. When coming to extremely low bit networks, i.e. only one or two bits are used to represent weights [12, 13, 14], they only work well on simple datasets (e.g. MNIST and CIFAR10), and usually incur a large loss on challenging datasets like ImageNet.

In this work, we focus on compressing and accelerating deep neural networks with extremely low bits weights, and present a unified strategy for learning such low bits networks. We overcome the limitation of the existing approaches by formulating it as a discretely constrained non-convex optimization problem, which is usually referred to as mixed integer programs (MIP). Given the NP hard nature of MIPs, we proposed a framework for learning extremely low bit neural network using the technique of alternating direction method of multipliers (ADMM) [15].

The main idea behind our method is to decouple the continuous variables from the discrete constraints using an auxiliary variable in the discrete space. This leads to a convenient form of the objective which is amenable to existing nonconvex optimization algorithms. Unlike previous low bit quantization methods [12, 16, 17] that incorporate an ad-hoc modification of the gradients for continuous weights, we simultaneously optimize in both continuous and discrete spaces, and connect the two solutions using an augmented Lagrangian. This is consistent with the previous observation from [18] that, by decoupling discrete constraints in MIP, one can use the information from the dual problem through ADMM to obtain a better upper bound. As a result of this reformulation, we can divide the problem of low bits quantized neural network into multiple subproblems which are significantly easier to solve. The main contributions of this paper are summarized as follows:

- We model the low bits neural network as a discretely constrained nonconvex optimization problem, and introduce auxiliary variables to decouple the continuous weights from the discrete constraints. With the use of ADMM, the originally hard problem are decomposed into several subproblems including proximal step, projection step and dual update.
- We show how the resulting subproblems can be efficiently solved. We utilize extragra-

dient method to accelerate the convergence of proximal step, and propose an iterative quantization algorithm to solve the projection step. The proposed algorithm enjoys a fast convergence in practice.

- We apply the proposed method to various well-known convolutional neural networks. Extensive experiments on multiple vision tasks including image classification and object detection demonstrate that the proposed method significantly outperforms the state-of-the-art approaches.

## 2 Related Work

Due to the high efficiency in both computation and memory footprints, low bits quantization of deep neural networks have received much attention in the literature. In this section, we have a brief review of the representative techniques. We also give a brief introduction to ADMM algorithm and its nonconvex extension.

### 2.1 Low bits quantization of neural network

The research of low bits quantization of neural network can be traced back to 1990s [19, 20]. Most of the benefits of low bits quantization, such as memory efficiency and multiplication free, had already been explored in these papers. However, the networks are shallow at that age so these approaches do not verify their validity in deep networks and large scale datasets.

In recent years, with the explosion of deep learning in various tasks, low bits quantization techniques have been revisited. Some early works quantize the pretrained weights with 4-12 bits and find such approximations do not decrease predictive performance [10, 8, 9, 11]. More recent works focus on training extremely low bits network from scratch with binary or ternary weights. Among these works, BinaryConnect [12] is the most representative one. BinaryConnect directly optimizes the loss of the network with weights $W$ replaced by $\text{sign}(W)$. In order to avoid the zero-gradient problem of sign function, the authors approximate it with the "hard tanh" function in the backward process. This simple idea inspired many following works. BinaryConnect only achieves good results on simple datasets such as MNIST, CIFAR10 and SVHN, but suffers a large degradation on challenging datasets like ImageNet.

Many efforts have been devoted to improve the performance of BinaryConnect. For example, Binary Weight Network (BWN) [16] proposes to improve the performance of BinaryConnect with a better approximation by introducing scale factors for the weights during binarization. Ternary Weight Network (TWN) [17] extends the idea of BWN to network with ternary weights and achieves a better performance. Inspired by BinaryConnect, in order to avoid the zero-gradient problem, both BWN and TWN modify the backward process by applying the gradients of the loss at the quantized weights.

Unlike previous works, we mathematically formulated the low bits quantization problem as a discretely constrained problem and present a unified framework based on ADMM to solve it in an efficient way. We simultaneously optimize the problem in both continuous and discrete space, and the two solutions are closely connected in the learning process.

### 2.2 ADMM and its nonconvex extension

Alternating Direction Method of Multipliers (ADMM) [15] is an algorithm that is intended to blend the decomposability of dual ascent with the superior convergence properties of the method of multipliers. The algorithm solves problems in the form:

$$\begin{aligned} \min \quad & f(\mathbf{x}) + g(\mathbf{z}) \\ \text{s.t.} \quad & A\mathbf{x} + B\mathbf{z} = \mathbf{c} \end{aligned} \tag{1}$$

with variables $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{z} \in \mathbb{R}^m$, where $A \in \mathbb{R}^{p\times n}$, $B \in \mathbb{R}^{p\times m}$ and $\mathbf{c} \in \mathbb{R}^p$.

The augmented Lagrangian of Eq.(1) can be formed as:

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^T(A\mathbf{x} + B\mathbf{z} - \mathbf{c}) + (\rho/2)\|A\mathbf{x} + B\mathbf{z} - \mathbf{c}\|_2^2 \tag{2}$$

where $\mathbf{y}$ is the Lagrangian multipliers, and ADMM consists of three step iterations:

$$\begin{aligned} \mathbf{x}^{k+1} &:= \arg\min_{\mathbf{x}} L_\rho(\mathbf{x}, \mathbf{z}^k, \mathbf{y}^k) \\ \mathbf{z}^{k+1} &:= \arg\min_{\mathbf{z}} L_\rho(\mathbf{x}^{k+1}, \mathbf{z}, \mathbf{y}^k) \\ \mathbf{y}^{k+1} &:= \mathbf{y}^k + \rho(A\mathbf{x}^{k+1} + B\mathbf{z}^{k+1} - \mathbf{c}) \end{aligned}$$

Even though ADMM was originally introduced as a tool for convex optimization problems, it turns out to be a powerful heuristic method even for NP-hard nonconvex problems. Recently, this tool has successfully been used as a heuristic to find approximate solutions to nonconvex mixed program problems [18, 21], which is very similar to our problem as noted later.

## 3 The Proposed Method

Let us first define the notion in this paper. Denote $f(W)$ as the loss function of a normal neural network, where $W = \{W_1, W_2, \cdots, W_L\}$. $W_i$ denotes the weights of the $i$-th layer in the network, which for example can be a 4-dimension tensor in convolutional layer or a 2-dimension matrix in fully connected layer. For the simplicity of notation, we regard all the entries in $W_i$ as a $d_i$-dimension vector in $\mathbb{R}^{d_i}$, and take $W$ as the concatenation of these vectors so that $W \in \mathbb{R}^d$ with $d = \sum_i d_i$.

In this work, we concentrate on training extremely low bits quantized neural networks. In specific, the weights of the network are restricted to be either zero or powers of two so

that the expensive floating-point multiplication operation can be replaced by cheaper and faster bit shift operation. In this section, we aim to mathematically model this problem and efficiently solve it.

### 3.1 Objective function

Intuitively, training a low bits neural network can be modeled as discretely constrained optimization, or in particular, mixed integer programs. For example, the weights in a ternary neural network are restricted to be $-1, 0$ or $+1$. Training such network can be mathematically formulated as mixed integer programs:

$$\min_{W} \quad f(W) \qquad \text{s.t.} \;\; W \in \mathcal{C} = \{-1, 0, +1\}^d$$

Since the weights are restricted to be zero or powers of two, we have constraints of this form

$$\mathcal{C} = \{-2^N, \cdots, -2^1, -2^0, 0, +2^0, +2^1, \cdots, +2^N\}$$

where $N$ is an integer which determines the number of bits. As in [16], we further introduce a scaling factor $\alpha$ to the constraints, i.e., instead of requiring $\mathcal{C} = \{\cdots, -2, -1, 0, +1, +2. \cdots\}$, we simply restrict $\mathcal{C}$ to $\mathcal{C} = \{\cdots, -2\alpha, -\alpha, 0, +\alpha, +2\alpha, \cdots\}$ with an arbitrary scaling factor $\alpha > 0$ that is strictly positive. It is worthy noting that the scale factor $\alpha$ in various layers can be different. In other words, for a neural network with $L$ layers, we actually introduce $L$ different scaling factors $\{\alpha_1, \alpha_2, \cdots, \alpha_L\}$. Formally, the objective function of low bits quantized neural networks can be formulated as:

$$\begin{aligned} \min_{W} \quad & f(W) \\ \text{s.t.} \quad & W \in \mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_L \end{aligned} \tag{3}$$

where $\mathcal{C}_i = \{0, \pm\alpha_i, \pm 2\alpha_i, \cdots, \pm 2^N \alpha_i\}$ and $\alpha_i > 0$. We emphasize that the scaling factor $\alpha_i$ in each layer doesn't incur more computation to the convolutional operator, because it can be multiplied after the efficient convolution with $\{0, \pm 1, \pm 2, \cdots, \pm 2^N\}$ done.

From the perspective of constrained optimization, the scaling factor $\alpha_i$ helps to expand the constraint space. As an example, Fig.1 gives an illustration of how it works for ternary network. In two dimensional space, for constraint $\{-1, 0, +1\}$, the possible solutions of ternary neural network are nine discrete points in the space. In contrast, with the scaling factor added, the constrained space is expanded to be four lines in the space. This large expansion of the constrained space will make the optimization easier.

### 3.2 Decouple with ADMM

The optimization in Eq.(3) is NP-hard in general because the weights are constrained in a discrete space. Most previous works try to directly train low bits models to minimize the

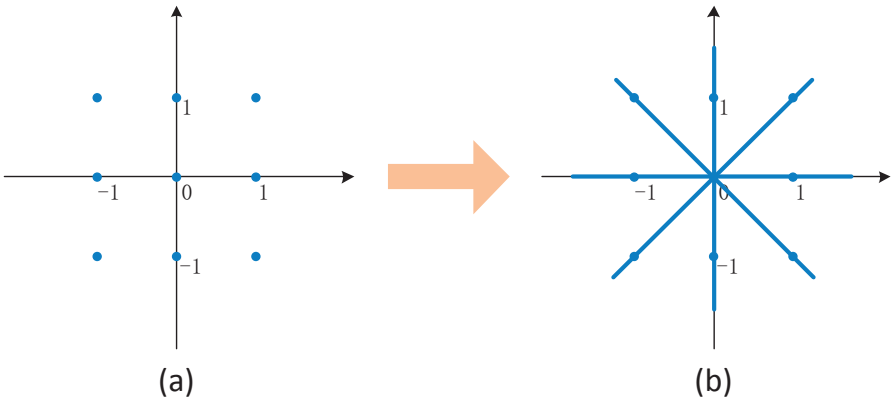


Figure 1: In ternary neural network, scaling factor expands the constrained space from (a) nice discrete points to (b) four lines in the space (two dimensional space as an example).

loss. For example, BinaryConnect [12] replace the weights $W$ with $\text{sign}(W)$ in the forward process so that the constraints will be automatically satisfied. Since the gradients of $\text{sign}(W)$ to $W$ is zero everywhere, the authors replace the sign function with "hard tanh" in the backward process. The same idea is also adopted by BWN [16] and TWN [17]. However, as indicated in [22], the use of different forward and backward approximations causes the mismatch of gradient, which makes the optimization instable.

We overcome the limitation of previous approaches by converting the problem into a form which is suitable to existing nonconvex optimization techniques. We introduce an auxiliary variable which is subject to the discrete restriction and equal to original variable. This is used with ADMM, which will result in the effect that the discrete variables being decoupled when we consider their minimization. Our basic idea is largely inspired by recent successful application of ADMM in mixed integer programs [18, 21].

First of all, defining an indicator function $I_{\mathcal{C}}$ for whether $W \in \mathcal{C}$, the objective in Eq.(3) can be written as

$$\min_{W} \quad f(W) + I_{\mathcal{C}}(W) \tag{4}$$

where $I_{\mathcal{C}}(W) = 0$ if $W \in \mathcal{C}$, otherwise $I_{\mathcal{C}}(W) = +\infty$.

By introducing an auxiliary variable $G$, we can rewrite the optimization in Eq.(4) with an extra equality constraint so that the weights is constrained to be equal to the discrete variable, but not subject to that restriction. In detail, the objective can be reformulated as:

$$\begin{aligned} \min_{W,G} \quad & f(W) + I_{\mathcal{C}}(G) \\ \text{s.t.} \quad & W = G \end{aligned} \tag{5}$$

Now we are considering a nonconvex optimization with convex linear constraints. Prob-

lems of such form can be conveniently solved with ADMM. The augmented Lagrange of Eq.(5), for parameter $\rho > 0$, can be formulated as:

$$L_\rho(W, G, \mu) = f(W) + I_{\mathcal{C}}(G) + \frac{\rho}{2}\|W - G\|^2 + \langle \mu, W - G \rangle \tag{6}$$

where $\mu$ denotes the Lagrangian multipliers and $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. With some basic collection of terms and a change of variable $\lambda = (1/\rho)\mu$, Eq.(6) can be equivalently formed as:

$$L_\rho(W, G, \lambda) = f(W) + I_{\mathcal{C}}(G) + \frac{\rho}{2}\|W - G + \lambda\|^2 - \frac{\rho}{2}\|\lambda\|^2 \tag{7}$$

Following the standard process of ADMM, this problem can be solved by repeating the following iterations:

$$W^{k+1} := \quad \arg\min_W \; L_\rho(W, G^k, \lambda^k) \tag{8}$$

$$G^{k+1} := \quad \arg\min_G \; L_\rho(W^{k+1}, G, \lambda^k) \tag{9}$$

$$\lambda^{k+1} := \quad \lambda^k + W^{k+1} - G^{k+1} \tag{10}$$

which is respectively the proximal step, projection step and dual update.

Unlike previous works, we simultaneously optimize the problem in both continuous space (i.e., proximal step) and discrete space (i.e., projection step), and the two solutions are brought together by ADMM in the learning process.

### 3.3 Algorithm subroutines

In this section, we elaborate on how the consequent subproblems in the above algorithm can be efficiently solved.

#### 3.3.1 Proximal step

For the proximal step, we optimize in the continuous space. Formally, we need to find the weights that minimize

$$L_\rho(W, G^k, \lambda^k) = f(W) + \frac{\rho}{2}\|W - G^k + \lambda^k\|^2 \tag{11}$$

Due to the decouple of ADMM, we are dealing with an unconstrained objective here. The loss can be interpreted as a normal neural network with a special regularization. Naturally, this problem can be solved with standard gradient decent method. It is easy to obtain the gradient with respect to the weights $W$:

$$\partial_W L = \partial_W f + \rho(W - G^k + \lambda^k)$$

However, we find the vanilla gradient descent method converges slowly in this problem. Since the second quadratic term occupies a large proportion of the whole lost, SGD will quickly pull the optimizer to the currently quantized weights so that the second term vanishes, and stack in that point. This results in a suboptimal solution since the loss of neural network is not sufficiently optimized.

To overcome this challenge, we resort to the extragradient method [23]. An iteration of the extragradient method consists of two very simple steps, prediction and correction:

$$\begin{aligned} W^{(p)} &:= W - \beta_p \partial_W L(W), \\ W^{(c)} &:= W - \beta_c \partial_W L(W^{(p)}) \end{aligned}$$

where $\beta_p$ and $\beta_c$ are the learning rates. A distinguished feature of the extragradient method is the use of an additional gradient step which can be seen as a guide during the optimization process. Particularly, this additional iteration allows to foresee the geometry of the problem and take the curvature information into account, which leads to a better convergency than standard gradient descent [24]. Specific to our problem, there is a more intuitive understanding of the above iterations. For the prediction step, the algorithm will quickly move to a point close to $G^k - \lambda^k$ so that the loss of quadratic regularization vanishes. Then in the correction step, the algorithm moves another step which tries to minimize the loss of neural network $f(W)$. These two steps avoid the algorithm stacking into a less valuable local minima. In practice, we find this extragradient method largely accelerate the convergence of the algorithm.

A key observation of (**??**) is that while minimizing over $G$, all the components $G_i$ are decoupled, therefore the auxiliary variables of each layer can be optimized independently. Recall that $W_i, G_i, \lambda_i, \mathcal{C}_i$ denote the weights, auxiliary variables, Lagrangian multipliers and constraints of the $i$-th layer respectively. We are essentially looking for the Euclidean projection of $(W_i^{k+1} + \lambda_i^k)$ onto a discrete set $\mathcal{C}_i$. Since the constraint is discrete and nonconvex, this optimization is nontrivial.

For convenience, we denote $(W_i^{k+1} + \lambda_i^k)$ as $V_i$. The projection of $V_i$ onto $\mathcal{C}_i$ can be formulated as

$$\begin{aligned} \min_{G_i, \alpha_i} \quad & \|V_i - G_i\|^2 \\ \text{s.t.} \quad & G_i \in \{0, \pm\alpha_i, \pm 2\alpha_i, \cdots, \pm 2^N \alpha_i\}^{d_i} \end{aligned} \tag{12}$$

Taking the scaling factor away from the constraints, the objective can be equivalently formulated as:

$$\begin{aligned} \min_{Q_i, \alpha_i} \quad & \|V_i - \alpha_i \cdot Q_i\|^2 \\ \text{s.t.} \quad & Q_i \in \{0, \pm 1, \pm 2, \cdots, \pm 2^N\}^{d_i} \end{aligned} \tag{13}$$

We propose an iterative quantization method to solve this problem. The algorithm alternates between optimizing $\alpha_i$ with $Q_i$ fixed and optimizing $Q_i$ with $\alpha_i$ fixed. In specific,

with $Q_i$ fixed, the problem becomes an univariate optimization. The optimal $\alpha_i$ can be easily obtained as

$$\alpha_i = \frac{V_i^T Q_i}{Q_i^T Q_i} \tag{14}$$

With $\alpha_i$ fixed, the optimal $Q_i$ is actually the projection of $\frac{V_i}{\alpha_i}$ onto $\{0, \pm 1, \pm 2, \cdots, \pm 2^N\}$, namely,

$$Q_i = \Pi_{\{0,\pm 1,\pm 2,\cdots,\pm 2^N\}} \left( \frac{V_i}{\alpha_i} \right) \tag{15}$$

where $\Pi$ denotes the projection operator. Moreover, the projection onto a discrete set is simply the closest point in it.

This iterative quantization algorithm is guaranteed to converge to a local minimum since we can get a decrease of loss in each step. In practice, we also find such a simple algorithm converges very fast. In most cases, we only need less than five iterations to get a stable solution.

#### 3.3.2 Dual update

In ADMM, dual update is actually gradient ascent in the dual space [15]. The iterate $\lambda^{k+1}$ in Eq.(10) can be interpreted as a scaled dual variable, or as the running sum of the error values $W^{k+1} - G^{k+1}$.

## 4 Experiments

In order to verify the effectiveness of the proposed algorithm, in this section we evaluate it on two benchmarks: ImageNet for image classification and Pascal VOC for object detection.

### 4.1 Image Classification

To evaluate the performance of our proposed method on image recognition task, we perform extensive experiments on the large scale benchmark ImageNet (ILSVRC2012), which is one of the most challenging image classification benchmarks. ImageNet dataset has about 1.2 million training images and 50 thousand validation images, and these images cover 1000 object classes. We comprehensively evaluate our method on almost all well-known deep CNN architectures, including AlexNet [25], VGG-16 [26], ResNet-18 [27], ResNet-50 [27] and GoogleNet [28].

#### 4.1.1 Experimental setup

In the ImageNet experiments, all the images are resized to $256 \times 256$. The images are then randomly clipped to $224 \times 224$ patches with mean subtraction and randomly flipping. No other data augmentation tricks are used in the learning process. We report both the

| | Accuracy | Binary | BWN | Ternary | TWN | {-2, +2} | {-4, +4} | Full Precision |
|---|---|---|---|---|---|---|---|---|
| AlexNet | Top-1 | **0.570** | 0.568 | **0.582** | 0.575 | 0.592 | 0.600 | 0.600 |
| | Top-5 | **0.797** | 0.794 | **0.806** | 0.798 | 0.818 | 0.822 | 0.824 |
| VGG-16 | Top-1 | **0.689** | 0.678 | **0.700** | 0.691 | 0.717 | 0.722 | 0.711 |
| | Top-5 | **0.887** | 0.881 | **0.896** | 0.890 | 0.907 | 0.909 | 0.899 |

Table 1: Accuracy of AlexNet and VGG-16 on ImageNet classification.

| | Accuracy | Binary | BWN | Ternary | TWN | {-2, +2} | {-4, +4} | Full Precision |
|---|---|---|---|---|---|---|---|---|
| Resnet-18 | Top-1 | **0.648** | 0.608 | **0.670** | 0.618 | 0.675 | 0.680 | 0.691 |
| | Top-5 | **0.862** | 0.830 | **0.875** | 0.842 | 0.879 | 0.883 | 0.890 |
| Resnet-50 | Top-1 | **0.687** | 0.639 | **0.725** | 0.656 | 0.739 | 0.740 | 0.753 |
| | Top-5 | **0.886** | 0.851 | **0.907** | 0.865 | 0.915 | 0.916 | 0.922 |

Table 2: Accuracy of ResNet-18 and ResNet-50 on ImageNet classification.

top-1 and top-5 classification accurate rates on the validation set, using single-view testing (single-crop on central patch only).

We study different kinds of bit width for weight quantization. Specifically, we tried binary quantization, ternary quantization, one-bit shift quantization and two-bits shift quantization. For one-bit shift quantization, the weights are restricted to be {-2$a$, -$a$, 0, +$a$, +2$a$}, which we denote as {-2, +2} in the comparison. Similarly, two-bits shift quantization are denoted as {-4, +4}. Binary quantization and ternary quantization need one bit and two bits to represent one weight respectively. Both {-2, +2} quantization and {-4, +4} quantization need three bits to represent one weight.

For binary and ternary quantization, we compare the proposed algorithm with the state-of-the-art approaches Binary Weight Network (BWN) [16] and Ternary Weight Network (TWN) [17]. Both BWN[1] and TWN[2] release their source code so we can evaluate their performance on different network architectures. Our method is implemented with Caffe [29]. The referenced full precision CNN models VGG-16, ResNet-50 and GoogleNet are taken from the Caffe model zoo[3].

#### 4.1.2 Results on AlexNet and VGG-16

AlexNet and VGG-16 are "old fashion" CNN architectures. AlexNet consists of 5 convolutional layers and 3 fully-connected layers. VGG-16 uses much wider and deeper structure than AlexNet, with 13 convolutional layers and 3 fully-connected layers. Table 1 demonstrates the comparison results on these two networks. For fair comparison with BWN, we

[1] `https://github.com/allenai/XNOR-Net`

[2] `https://github.com/fengfu-chris/caffe-twns`

[3] `https://github.com/BVLC/caffe/wiki/Model-Zoo`

| Accuracy | Binary | BWN | Ternary | TWN | {-2, +2} | {-4, +4} | Full Precision |
|---|---|---|---|---|---|---|---|
| Top-1 | **0.603** | 0.590 | **0.631** | 0.612 | 0.659 | 0.663 | 0.687 |
| Top-5 | **0.832** | 0.824 | **0.854** | 0.841 | 0.873 | 0.875 | 0.889 |

Table 3: Accuracy of GoogleNet on ImageNet classification.

report the performance of the batch normalization [30] version of AlexNet. The accuracy of the improved AlexNet is higher than the original one (Top-1 60.0% vs. 57.4%, Top-5 82.4% vs. 80.4%).

On these two architectures, the proposed algorithm achieves a lossless compression with only 3 bits compared with the full precision references. For {-2, +2} and {-4, +4} quantization, the performance of the our quantized networks is even better than the original full precision network on VGG-16. Similar results are observed in BinaryConnect on small datasets. This is because discrete weights could provide a form of regularization which can help to generalize better. These results also imply the heavy redundancy of the parameters in full precision AlexNet and VGG-16 models. This finding is consistent with that in other studies such as SqueezeNet [31]. In SqueezeNet, the authors suggest that one can achieve AlexNet-level accuracy on ImageNet with 50x fewer parameters.

Our binary quantization and ternary quantization slightly outperforms BWN and TWN on these two architectures. Comparing the accuracy of ternary quantization and binary quantization, we find that ternary network consistently works better than binary network. We also emphasize that the ternary network is more computing efficient than binary network because of the existence of many zero entries in the weights, as indicated in [32].

#### 4.1.3 Results on ResNet

The results on ResNet-18 are shown in Table 2. ResNet-18 has 18 convolutional layers with shortcut connections. For the proposed method, both the binary and ternary quantization substantially outperform their competitors on this architecture. For example, our binary network outperforms BWN by 4 points in top-1 accuracy and 3.2 points in top-5 accuracy. The proposed ternary quantization outperforms TWN by 5.2 points and 3.3 points in top-1 and top-5 accuracy respectively. All these gaps are significant on ImageNet. We also observe over two percent improvement for our ternary quantization over binary quantization.

The effectiveness of our method is also verified on very deep convolutional network such as ResNet-50. Besides significantly increased network depth, ResNet-50 has a more complex network architecture than ResNet-18. Table 2 details the results on ResNet-50. It is easy to observe the similar trends as in ResNet-18. Our method is considerably better than the compared BWN and TWN. For example, our binary quantization obtains about 5 points improvement on top-1 accuracy over BWN.

For both ResNet-18 and ResNet-50, there is a more noticeable gap between the low

bits quantized networks and full precision reference. Different from AlexNet and VGG-16, on ResNet we notice about 1 point gap in top-1 accuracy between {-4, +4} quantized network and full precision reference. These results suggest that training extremely low bits quantized network is easier for AlexNet and VGG than for ResNet, which also implies the parameters in AlexNet and VGG-16 are more redundant than those in ResNet-18 and ResNet-50.

#### 4.1.4 Results on GoogleNet

The results on GoogleNet are illustrated in Table 3. GoogleNet is a 22 layers deep network, organized in the form of the "Inception module". Similar to ResNet, GoogleNet is more compact than AlexNet and VGG-16, so it will be more difficult to compress it. There exists a gap of more than 2 points in top-1 accuracy between {-4, +4} quantized network and full precision version. The loss of binary quantization is more significant, which reaches 8 points in top-1 accuracy. Despite this, our method stills outperforms BWN[4] and TWN on this network.

#### 4.1.5 Compare with the most recent works

To our knowledge, Trained Ternary Quantization (TTN) [33] and Incremental Network Quantization (INQ) [34] are two of the most recent published works on low bits quantization of deep neural network. Instead of quantizing the ternary weights to be $\{-\alpha, 0, +\alpha\}$, TTN makes it less restrictive as $\{-\alpha, 0, +\beta\}$. Note that our method can be easily extended to deal with constraints of such form. Nevertheless, the computation of such form of ternary network is less efficient than the original one. As an example, for fast implementation the inner product between vector $\mathbf{x}$ and vector $(-\alpha, -\alpha, 0, +\beta)$ will be decomposed as $\beta\mathbf{x} \cdot (0, 0, 0, 1) - \alpha\mathbf{x} \cdot (1, 1, 0, 0)$, having to do two floating-point multiplications with $\alpha$ and $\beta$.

Since TTN only reports its results on AlexNet and ResNet-18, we compare the performance on these two architectures. Detailed results are summarized in Table 4 and Table 5. Our approach performs better than TTN on AlexNet (the results of ternary INQ on AlexNet is not available), and better than both TTN and INQ on ResNet-18. INQ shows more results on 5-bits networks in the paper. For example, the reported top-1 and top-5 accuracy of ResNet-50 with 5-bits are 73.2% and 91.2% [34]. In contrast, our method achieves such accuracy with only 3 bits.

#### 4.1.6 INT8 quantized 1×1 kernel

We notice the extremely low bits quantization of GoogleNet suffers a large degradation. We guess this may be due to the 1×1 kernel in each inception. In order to verify this

[4]Note that the GoogleNet used in BWN paper is an improved variant of the original version used in this paper.

| Method | Top-1 accuracy | Top-5 accuracy |
|---|---|---|
| TTN [33] | 0.575 | 0.797 |
| Ours (Ternary) | **0.582** | **0.806** |

Table 4: Comparison with TTN on AlexNet.

| Method | Top-1 accuracy | Top-5 accuracy |
|---|---|---|
| TTN [33] | 0.666 | 0.872 |
| INQ [34] | 0.660 | 0.871 |
| Ours (Ternary) | **0.670** | **0.875** |

Table 5: Comparison with TTN and INQ on ResNet-18.

point, we perform another experiment on GoogleNet. In this version, the 1×1 kernels in the network are quantized with relatively more bits, i.e., INT8, and kernels of other size are quantized as usual. Table 6 shows the results.

By comparing the results in Table 6 and those in Table 3, we observe a considerable improvement, especially for binary and ternary quantization. As we have discussed, discrete weights can be interpret as a strong regularizer to the network. However, the parameters in 1×1 kernel is much less than those in other kernels. Imposing a very strong regularizer to such kernels may lead to underfitting of the network. These results suggest that we should quantize different parts of the networks with different bit width in practice. Letting the algorithm automatically determine the bit width will be our future work.

### 4.2 Object Detection

In order to evaluate the proposed method on object detection task, we apply it to the state of arts detection framework SSD [35]. The models are trained on Pascal VOC2007 and VOC2012 train dataset, and tested on Pascal VOC 2007 test dataset. For SSD, we adopt the open implementation released by the authors[5]. In all experiments, we follow the same setting as in [35] and the input images are resized to $300 \times 300$.

The proposed method are evaluated on two base models, i.e., VGG-16 and Darknet reference model. Both base networks are pre-trained on ImageNet dataset. The VGG-16 network here is a variant of original one [26]. In detail, the fc6 and fc7 are converted to convolutional layers with $1 \times 1$ kernel, and the fc8 layer is removed. The parameters of fc6 and fc7 are also subsampled. The darknet reference model is borrowed from YOLO [36], which is another fast detection framework. Darknet is designed to be small yet power, which attains comparable accuracy performance as AlexNet but only with about 10% of the parameters. We utilize the base darknet model downloaded from the website[6].

[5] `https://github.com/weiliu89/caffe/tree/ssd`

[6] `https://pjreddie.com/darknet/imagenet/`

| Acc. | Binary | Ternary | {-2, +2} | {-4, +4} | Full |
|---|---|---|---|---|---|
| Top-1 | 0.654 | 0.667 | 0.674 | 0.676 | 0.687 |
| Top-5 | 0.867 | 0.877 | 0.883 | 0.883 | 0.889 |

Table 6: Accuracy of GoogleNet. 1×1 kernels are quantized with INT8.

| mAP | Darknet+SSD | VGG16+SSD |
|---|---|---|
| Ternary | 0.609 (0.621) | 0.762 |
| {-4, +4} | 0.624 (0.639) | 0.776 |
| Full Precision | 0.642 | 0.778 |

Table 7: mAP of VGG16+SSD and Darknet+SSD on Pascal VOC 2007

To the best of our knowledge, there is no other works on low bits quantization applied their algorithms to the object detection tasks. We compare our quantized network with full precision network in this experiment. We only implement ternary and {-4,+4} quantization for this experiment. Darknet has utilized many 1×1 kernels as in GoogleNet to accelerate the inference process. We implement two versions of Darknet. In the first version, the 1×1 kernels are also quantized as usual, while in the second version these kernels are quantized with INT8. Table 7 shows the mean average precision (mAP) on both models.

For {-4,+4} quantization, we find that the mAP of both modes are very close to the full precision version. On VGG16+SSD, we only suffer a loss of 0.002 in mAP. Comparing two versions of Darknet+SSD, the first version achieves a mAP of 0.624, and the second version obtains a improvement of 1.5 points. For ternary quantization, the accuracy degradation of Darknet+SSD is larger than VGG16+SSD, because the parameters of Darknet is less redundant than VGG-16. All these results indicate that our proposed method is also effective on the object detection tasks.

## 5 Conclusion

This work focused on compression and acceleration of deep neural networks with extremely low bits weight. Inspired by the efficient heuristics proposed to solve mixed integer programs, we proposed to learn low bits quantized neural network in the framework of ADMM. We decoupled the continuous parameters from the discrete constraints of network, and cast the original hard problem into several subproblems. We proposed to solve these subproblems using extragradient and iterative quantization algorithms that lead to considerably faster convergency compared to conventional optimization methods. Extensive experiments on convolutional neural network for image recognition and object detection have shown the effectiveness of the proposed method.

# References


[1] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[2] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence*, 2016.

[3] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *Advances in Neural Information Processing Systems*, 2014.

[4] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

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

[6] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[8] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[9] T. Dettmers, "8-bit approximations for parallelism in deep learning," *arXiv preprint arXiv:1511.04561*, 2015.

[10] M. Courbariaux, Y. Bengio, and J.-P. David, "Low precision arithmetic for deep learning," *arXiv preprint arXiv:1412.7024*, 2014.

[11] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," *arXiv preprint arXiv:1511.06393*, 2015.

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in Neural Information Processing Systems*, 2015.

[13] I. Hubara, D. Soudry, and R. E. Yaniv, "Binarized neural networks," *arXiv preprint arXiv:1602.02505*, 2016.

[14] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan, "Training binary multilayer neural networks for image classification using expectation backpropagation," *arXiv preprint arXiv:1503.03562*, 2015.

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[16] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *European Conference on Computer Vision*, 2016.

[17] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

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

[19] E. Fiesler, A. Choudry, and H. J. Caulfield, "Weight discretization paradigm for optical neural networks," in *The Hague'90, 12-16 April*. International Society for Optics and Photonics, 1990, pp. 164–173.

[20] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE transactions on Neural Networks*, vol. 4, no. 1, pp. 53–62, 1993.

[21] A. Alavian and M. C. Rotkowitz, "Improving admm-based optimization of mixed integer objectives," in *Information Sciences and Systems (CISS), 2017 51st Annual Conference on*. IEEE, 2017, pp. 1–6.

[22] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," *arXiv preprint arXiv:1702.00953*, 2017.

[23] G. M. Korpelevich, "An extragradient method for finding saddle points and for other problems," *Matecon*, 1976.

[24] A. Nemirovski, "Prox-method with rate of convergence o(1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems," *SIAM Journal on Optimization*, 2004.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, 2012.

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

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

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

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[31] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[32] G. Venkatesh, E. Nurvitadhi, and D. Marr, "Accelerating deep convolutional networks using low-precision and sparsity," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2861–2865.

[33] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *International Conference on Learning Representations*, 2017.

[34] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *International Conference on Learning Representations*, 2017.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *European Conference on Computer Vision*, 2016.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.