CBP: Backpropagation with constraint on weight precision using pseudo-Lagrange multiplier method

Guhyun Kim, Doo Seok Jeong[*]
Division of Materials Science and Engineering
Hanyang University, Republic of Korea
`guhyunkim01@gmail.com, dooseokj@hanyang.ac.kr`

Abstract

Backward propagation of errors (backpropagation) is a method to minimize objective functions (e.g., loss functions) of deep neural networks by identifying optimal sets of weights and biases. Imposing constraints on weight precision is often required to alleviate prohibitive workloads on hardware. Despite the remarkable success of backpropagation, the algorithm itself is not capable of considering such constraints unless additional algorithms are applied simultaneously. To address this issue, we propose the constrained backpropagation (CBP) algorithm based on the pseudo-Lagrange multiplier method to obtain the optimal set of weights that satisfy a given set of constraints. The defining characteristic of the proposed CBP algorithm is the utilization of a Lagrangian function (loss function plus constraint function) as its objective function. We considered various types of constraints — binary, ternary, one-bit shift, and two-bit shift weight constraints. As a post-training method, CBP applied to AlexNet, ResNet-18, ResNet-50, and GoogLeNet on ImageNet, which were pre-trained using the conventional backpropagation. For most cases, the proposed algorithm outperforms the state-of-the-art methods on ImageNet, e.g., 66.6%, 74.4%, and 64.0% top-1 accuracy for ResNet-18, ResNet-50, and GoogLeNet with binary weights, respectively. This highlights CBP as a learning algorithm to address diverse constraints with the minimal performance loss by employing appropriate constraint functions. The code for CBP is publicly available at `https://github.com/dooseokjeong/CBP`.

1 Introduction

Currently, deep learning-based methods are applied in a variety of tasks, including the classification of static data, e.g., image recognition [1, 2]; classification of dynamic data, e.g., speech recognition [3–6]; function approximations, which require the output of precise predictions, e.g., electronic structure predictions [7] and nonlinear circuit predictions [8]. All of the aforementioned tasks require discriminative models. Additionally, generative models, including generative adversarial networks [9] and variants [10–13], comprise another type of deep neural network. Despite the diversity in application domain and model type used, almost all deep learning-based methods use backpropagation as a common learning algorithm.

Recent developments in deep learning have primarily focused on increasing the size and depth of deep neural networks (DNNs) to improve their learning capabilities as in the case of state-of-the-art DNNs like VGGNet [14] and ResNet [15]. Given that the memory capacity required by a DNN is proportional to the number of parameters (weights and biases), memory usage for DNN becomes severe. Additionally, a significant number of multiply-accumulate operations during the

---

[*]Corresponding author

training and inference stages impose prohibitive workload on hardware. Thus, efficient hardware-resource consumption is critical to the optimal performance of deep learning. One way to address this requirement is the use of weights of limited precision, such as binary [16, 17] and ternary weights [18, 19]. To this end, particular constraints are applied to weights during training, and additional algorithms for weight quantization are used in conjunction with backpropagation. This is because such constraints are not considered during the minimization of the objective function (loss function) when backpropagation is executed.

We adopt the Lagrange multiplier method (LMM) to combine basic backpropagation with additional constraint algorithms and produce a single constrained backpropagation (CBP) algorithm. We refer to the adopted method as pseudo-LMM because the constraint functions $cs\left(\boldsymbol{x}\right)$ are nondifferentiable at $\boldsymbol{x}_m\left(=\arg\min_{\boldsymbol{x}}cs\left(\boldsymbol{x}\right)\right)$, rendering LMM inapplicable. Nevertheless, pseudo-LMM successfully attains the optimal point under particular conditions as for LMM. In the CBP algorithm, the optimal weights satisfying a given set of constraints are evaluated via a basic backpropagation algorithm. It is implemented by simply replacing the conventional objective function (loss function) with a Lagrangian function $\mathcal{L}$ that comprises the loss and constraint functions as sub-functions that are subjected to simultaneous minimization. Therefore, this method is perfectly compatible with conventional deep learning frameworks. The primary contributions of this study are as follows.

- We introduce a novel and simple method to incorporate given constraints into backpropagation by using a Lagrangian function as the objective function. The proposed method is able to address any set of constraints on the weights insomuch as the constraint functions are mathematically well-defined.
- We introduce pseudo-LMM with constraint functions $cs\left(w\right)$ that are nondifferentiable at $w_m\left(=\arg\min_{w}cs\left(w\right)\right)$ and analyze the kinetics of pseudo-LMM in the continuous time domain.
- We introduce optimal (sawtooth-like) constraint functions with gradually vanishing unconstrained-weight windows and provide a guiding principle for the stable co-optimization of weights and Lagrange multipliers in a quasi-static fashion.
- We evaluate the performance of CBP applied to AlexNet, ResNet-18, ResNet-50, and GoogLeNet (pre-trained using backpropagation with full-precision weights) with four different constraints (binary, ternary, one-bit shift, and two-bit shift weight constraints) on ImageNet as proof-of-concept examples. The results highlight the classification accuracy outperforming the previous state-of-the-art results.

2 Related work

The simplest approach to weight quantization is the quantization of pre-trained weights. Gong et al. [20] proposed several methods for weight quantization and demonstrated that binarizing weights using a sign function degraded the top-1 accuracy on ImageNet by less than 10%. Mellempudi et al. [21] proposed a fine-grained quantization algorithm that calculates the optimal thresholds for the ternarization of pre-trained weights. The expectation backpropagation algorithm [22] implements a variational Bayesian approach to weight quantization. It uses binary weights and activations during the inference stage. Zhou et al. [23] proposed the incremental network quantization method (INQ) that iteratively re-trains a group of weights to compensate for the performance loss caused by the rest of weights which are quantized using pre-set quantization thresholds.

Several methods of weight quantization utilize auxiliary real-valued weights in conjunction with quantized weights during training. The straight-through-estimator (STE) comprises the conduction of forward and backpropagation using quantized weights but relies on the auxiliary real-valued weights for the update of weights [3]. BinaryConnect [16] utilizes weights binarized by a sign function for forward and backpropagation, and the real-valued weights are updated via backpropagation with binary weights. The binary-weight-network (BWN) [17] identifies the binary weights closest to the real-valued weights using a scaling factor, and it exhibits a higher classification accuracy than BinaryConnect on ImageNet. The binarized neural nets [24] and XNOR-Nets [17] are extensions of BinaryConnect and BWN, respectively, which utilize binary activations alongside binary weights.

Lin et al. [18] proposed TernaryConnect and Ternary-weight-network (TWN), which are similar to BinaryConnect and BWN but use weight-ternarization methods instead. Trained ternary quantization

(TTQ) [25] also uses ternary weights that are quantized using trainable thresholds for quantization. LQ-Nets proposed by Zhang et al. [26] utilize activation- and weight-quantizers considering the actual distributions of full-precision activation and weight, respectively. DeepShift Network [27] includes the LinearShift and ConvShift operators that replace the multiplications of fully-connected layers and convolution layers, respectively. Qin et al. [28] introduced IR-Nets that feature the use of error decay estimators to approximate sign functions for weight and activation binarization to differentiable forms. Gong et al. [29] and Yang et al. [30] employed functions similar to step functions but differentiable. Pouransari et al. [31] proposed the least squares quantization method that searches proper scale factors for binary quantization. Elthakeb et al. [32] attempted to quantize weights and activations by applying sinusoidal regularization. The regularization involves two hyperparameters that determine weight-quantization and bitwidth-regularization strengths.

To take into account the constraint on weight precision, Leng et al. [33] used an augmented Lagrangian function as an objective function, which includes a constraint-agreement sub-function. The method was successfully applied to various DNNs on ImageNet; yet, the method failed to reach the accuracy level for full-precision models even when 3-bit precision was used.

The CBP algorithm proposed in this study also utilizes LMM; but CBP essentially differs from [33] given that (i) the basic differential multiplier method (BDMM), rather than ADMM, is used to apply various constraints on weight precision, (ii) particularly designed constraint functions with gradually vanishing unconstrained-weight windows are used, and (iii) substantial improvement on the classification accuracy on ImageNet is achieved.

3 Optimization method

3.1 Pseudo-Lagrange multiplier method

LMM calculates the maximum or minimum value of a differentiable function f under a differentiable constraint $cs = 0$. [34] Let us assume that a function $f(x, y)$ attains its minimum (or maximum) value m satisfying a given constraint at (x_m, y_m), i.e., $f(x_m, y_m) = m$. Further, $cs(x_m, y_m) = 0$ as the constraint is satisfied at this point. In this case, the point of intersection between the graphs of the two functions, $f(x, y) = m$ and $cs(x, y) = 0$, is (x_m, y_m). Because the two functions have a common tangent at the point of intersection, the following equation holds:

$$\nabla_{x,y} f = -\lambda \nabla_{x,y} cs, \tag{1}$$

at (x_m, y_m), where λ is a Lagrange multiplier [35].

The Lagrangian function is defined as $\mathcal{L}(x, y, \lambda) = f(x, y) + \lambda cs(x, y)$. Consider a local point (x, y) at which the gradient of $\mathcal{L}(x, y, \lambda)$ is zero. Therefore, $\nabla_{x,y,\lambda} \mathcal{L}(x, y, \lambda) = \nabla_{x,y}[f(x, y) + \lambda cs(x, y)] + \nabla_\lambda[\lambda cs(x, y)]$. Thus, $\nabla_{x,y,\lambda} \mathcal{L}(x, y, \lambda) = \mathbf{0}$ is equivalent to the following equations.

$$\nabla_{x,y,\lambda} \mathcal{L}(x, y, \lambda) = \mathbf{0} \quad \Leftrightarrow \quad \begin{cases} \nabla_{x,y}[f(x, y) + \lambda cs(x, y)] &= \mathbf{0}, \\ \nabla_\lambda[\lambda cs(x, y)] &= 0 \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} \nabla_{x,y} f(x, y) &= -\lambda \nabla_{x,y} cs(x, y) \\ cs(x, y) &= 0 \end{cases} \tag{2}$$

This satisfies the condition in Eq. (1) as well as the constraint $cs(x, y) = 0$. Therefore, the local point (x, y) corresponds to the minimum (or maximum) point of the function f under the constraint.

We define pseudo-LMM to address similar minimization tasks but with continuous constraint functions that are nondifferentiable at $(x_m, y_m) = \arg\min_{x,y} cs(x, y)$. Thus, Eq. (1) cannot be satisfied at the optimal point. Nevertheless, pseudo-LMM enables us to *minimize* the function $f(\boldsymbol{x})$ subject to the constraint condition $cs(\boldsymbol{x}) = 0$ using the Lagrangian function $\mathcal{L}(\boldsymbol{x}, \lambda)$.

Definition 3.1 (Pseudo-LMM). Pseudo-LMM is a method to attain the optimal variables $\boldsymbol{x}_m$ that minimize function $f(\boldsymbol{x})$ subject to the constraint condition $cs(\boldsymbol{x}) = 0$, where the function $cs(\boldsymbol{x})$ is nondifferentiable at $\boldsymbol{x}_m$ but reaches the minimum at $\boldsymbol{x}_m$, i.e., $cs(\boldsymbol{x}_m) = 0$.

Theorem 3.1. Minimizing the Lagrangian function $\mathcal{L}(\boldsymbol{x}, \lambda)$, which is given by $\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda cs(\boldsymbol{x})$, is equivalent to minimizing the function $f(\boldsymbol{x})$ subject to $cs(\boldsymbol{x}) = 0$

Proof. The Lagrangian function $\mathcal{L}(\boldsymbol{x}, \lambda)$ is always differentiable with respect to the Lagrangian multiplier λ, so that the equation $cs(\boldsymbol{x}) = 0$ holds at the optimal point $\boldsymbol{x}_m$. Thus, we have

$$\underset{\boldsymbol{x}, \lambda}{\text{minimize }} \mathcal{L}(\boldsymbol{x}, \lambda) \Leftrightarrow \begin{cases} \underset{\boldsymbol{x}}{\text{minimize}} & \mathcal{L}(\boldsymbol{x}; \lambda) \\ \text{subject to} & cs(\boldsymbol{x}) = 0. \end{cases} \tag{3}$$

When the constraint is satisfied, i.e., $cs(\boldsymbol{x}) = 0$, the Lagrangian function $\mathcal{L}(\boldsymbol{x}; \lambda)$ equals the function $f(\boldsymbol{x})$, so that the task to minimize $\mathcal{L}(\boldsymbol{x}, \lambda)$ with respect to $\boldsymbol{x}$ and λ corresponds to the task to minimize $f(\boldsymbol{x})$ subject to $cs(\boldsymbol{x}) = 0$. $\qquad\square$

Given Theorem 3.1, pseudo-LMM can attain the optimal point by minimizing the Lagrangian function $\mathcal{L}$ in spite of the nondifferentiability of the constraint function $cs(\boldsymbol{x}) = 0$ at the optimal point $\boldsymbol{x}_m$. Note that not all functions have zero gradients at their minimum points; for instance, the function $y = |x|$ attains its minimum at $x = 0$ but the gradient at the minimum point is not defined. However, all convex functions have zero gradients at their minimum points. Thus, LMM to minimize the function $f(\boldsymbol{x})$ subject to the constraint convex function $cs(\boldsymbol{x})$ with minimum point $\boldsymbol{x}_m$ is a subset of pseudo-LMM. In this case, the constraint function has zero gradient at the minimum point, so that Eq. (2) becomes

$$\nabla_{\boldsymbol{x}, \lambda} \mathcal{L}(\boldsymbol{x}, \lambda) = \boldsymbol{0} \quad \Leftrightarrow \quad \begin{cases} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) &= 0 \\ \text{subject to } cs(\boldsymbol{x}) &= 0. \end{cases} \tag{4}$$

We will consider *continuous* constraint functions with a few nondifferentiable points in their variable domains, including their minimum points. Other than such nondifferentiable points, we will use the gradient descent method to search for the minimum points within the framework of pseudo-LMM. Hereafter, when the gradient of the Lagrangian $\mathcal{L}$ function is remarked, its variable domain excludes such nondifferentiable points.

The optimal solution to Eq. (4) can be found using the basic differential multiplier method (BDMM) [36] that calculates the point at which $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ attains its minimum value by driving $\boldsymbol{x}$ toward the constraint subspace $\bar{x}$ ($\boldsymbol{cs}(\bar{x}) = 0$). The BDMM updates $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ according to the following relations.

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta_x \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}), \tag{5}$$

and

$$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \eta_\lambda \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}). \tag{6}$$

BDMM is cheaper than Newton's method in terms of computational cost. Additionally, Eq. (5) is identical to the solution used in the gradient descent method during backpropagation, except for the use of a Lagrangian function instead of a loss function. This indicates the compatibility of pseudo-LMM with the optimization framework based on backpropagation.

3.2 Constrained backpropagation using the pseudo-Lagrange multiplier method

We utilize pseudo-LMM to train DNNs with particular sets of weight-constraints. We define a Lagrangian function $\mathcal{L}$ in the context of feedforward DNN using the following relation.

$$\begin{aligned} \mathcal{L}\left(\boldsymbol{y}^{(k)}, \hat{\boldsymbol{y}}^{(k)}; \boldsymbol{W}, \boldsymbol{\lambda}\right) &= C\left(\boldsymbol{y}^{(k)}, \hat{\boldsymbol{y}}^{(k)}; \boldsymbol{W}\right) + \boldsymbol{\lambda}^{\mathrm{T}} \boldsymbol{cs}(\boldsymbol{W}), \\ \boldsymbol{cs} &= [cs_1(w_1), \dots, cs_{n_w}(w_{n_w})]^{\mathrm{T}} \\ \boldsymbol{\lambda} &= [\lambda_1, \dots, \lambda_{n_w}]^{\mathrm{T}}. \end{aligned} \tag{7}$$

where $\boldsymbol{y}^{(k)}$ and $\hat{\boldsymbol{y}}^{(k)}$ denote the actual output vector for the kth input data and correct label, respectively. The set $\boldsymbol{W}$ denotes a set of weight matrices, including n_w weights in aggregate, and the function C denotes a loss function. Each weight w_i ($1 \le i \le n_w$) is given one constraint function $cs_i(w_i)$ and one multiplier λ_i.

We chose sawtooth-shaped constraint functions. We quantize the real-valued weights into n_q values in the set $\boldsymbol{Q} = \{q_i\}_{i=1}^{n_q}$, where $q_i < q_{i+1}$ for all i. We also employ a set of the medians of neighboring
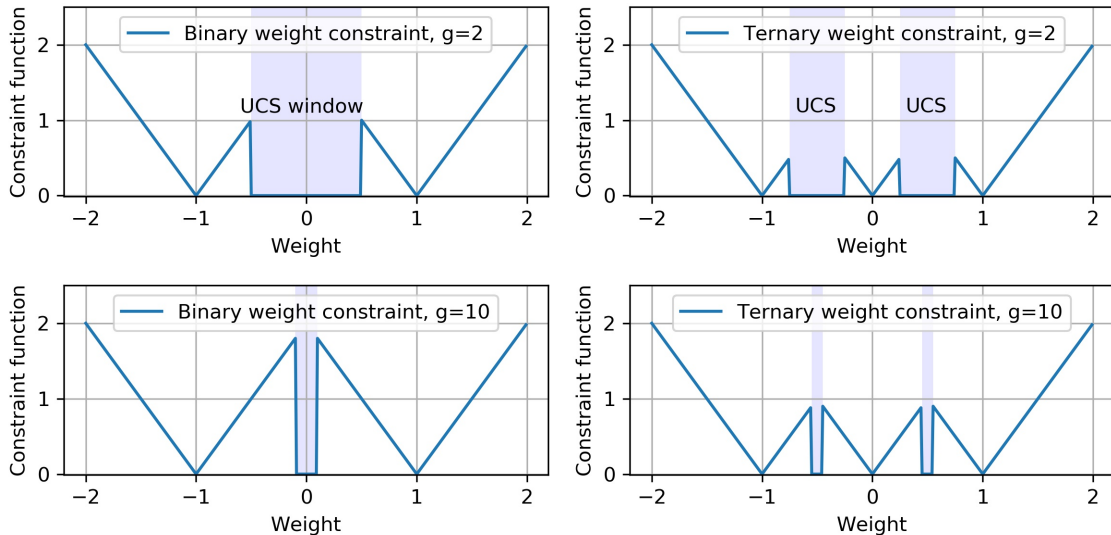
4

Figure 1: Binary- and ternary-weight constraint functions for $g = 2$ and 10. Blue-filled regions indicate unconstrained-weight windows.

values in the set $\boldsymbol{Q}$: $\boldsymbol{M} = \{m_i\}_{i=1}^{n_q-1}$, where $m_i = (q_i + q_{i+1})/2$. Using $\boldsymbol{Q}$ and $\boldsymbol{M}$, we define a partial constraint function y_i for $i = 0$, $1 \leq i < n_q$, and $i = n_q$,

$$
\begin{aligned}
y_0(w) &= \begin{cases} -2(w - q_1) & \text{if} \quad w < q_1, \\ 0 & \text{otherwise}, \end{cases} \\
y_i(w) &= \begin{cases} -2|w - m_i| + q_{i+1} - q_i & \text{if} \quad q_i \leq w < q_{i+1}, \\ 0 & \text{otherwise}, \end{cases} \\
y_{n_q}(w) &= \begin{cases} 2(w - q_{n_q}) & \text{if} \quad w \geq q_{n_q}, \\ 0 & \text{otherwise}, \end{cases}
\end{aligned}
\tag{8}
$$

respectively. The constraint function cs is the summation of the partial constraint functions $Y(w) = \sum_{i=0}^{n_q} y_i(w)$, gated by the unconstrained-weight window $ucs(w)$ parameterized by a variable g.

$$
cs(w; \boldsymbol{Q}, \boldsymbol{M}, g) = ucs(w) Y(w),
\tag{9}
$$

where

$$
ucs(w) = 1 - \sum_{i=0}^{n_q-1} H\left(\frac{1}{2g}(q_{i+1} - q_i) - |w - m_i + \epsilon|\right),
\tag{10}
$$

where $\epsilon \to 0^+$, and H denotes the Heaviside step function. The function $ucs(w)$ realizes the unconstrained-weight window as a function of $g \ (\geq 1)$. When $g = 1$, the function outputs zero for $q_1 \leq w < q_{n_q}$, merely confining w to the range $q_1 \leq w < q_{n_q}$ without weight quantization, whereas, when $g \to \infty$, the window vanishes, allowing the constraint function to quantize the weight in the entire weight range. Examples of function $ucs(w)$ are shown in Fig. 1.

The unconstrained-weight window variable g is initially set to one and updated such that it keeps increasing during training, i.e., the window gradually vanishes. The window gradually vanishing allows sequential weight quantization such that the further the initial weights from their nearest q_i, the later their weights are subject to quantization, which is otherwise subject to simultaneous (abrupt) quantization. It is likely that the further the initial weights from their nearest q_i, the larger the increase in loss function C when they are quantized. Thus, the sequential quantization from the weights close to their q_i likely avoids an abrupt increase in the loss. Further, while the closer weights are being quantized, the further weights (not subject to quantization yet) are being updated to reduce the loss given the partially quantized weights. This effect will be discussed in Sections 3.3 and 5.

For every training batch, the weights are updated following a method similar to conventional back-propagation. Nevertheless, the use of the Lagrangian function in Eq. (8), rather than a loss function

only, as an objective function constitutes a critical difference. The Lagrange multipliers $\boldsymbol{\lambda}$ are subsequently updated using the gradient ascent method in Eq. (6). Updating cross-coupled variables, such as $\boldsymbol{W}$ and $\boldsymbol{\lambda}$, often experiences difficulties in convergence toward the optimal values because of oscillation around the optimal values. A feasible solution involves quasi-static update. To this end, we significantly reduce the update frequency of the Lagrange multipliers $\boldsymbol{\lambda}$ compared with weights $\boldsymbol{W}$.

Weight update: Weights $\boldsymbol{W}$ are updated once every iteration as for the conventional backpropagation but using the Lagrangian function $\mathcal{L}$.

Lagrange multiplier update: Lagrange multipliers $\boldsymbol{\lambda}$ are conditionally updated once every training epoch. The update is allowed if the summation of all $\mathcal{L}$ in a given epoch $(\mathcal{L}_{sum})$ is not smaller than $\mathcal{L}_{sum}$ for the previous epoch $(\mathcal{L}_{sum}^{pre})$ or the multipliers $\boldsymbol{\lambda}$ have not been updated in the past p_{max} epochs. This achieves the convergence of $\boldsymbol{W}$ for a given $\boldsymbol{\lambda}$ in a quasi-static manner.

Unconstrained weight window update: Unconstrained-weight window variable $\boldsymbol{g}$ is updated on the same condition as for the Lagrange multipliers $\boldsymbol{\lambda}$. Unlike weights $\boldsymbol{W}$ and multipliers $\boldsymbol{\lambda}$, the variable $\boldsymbol{g}$ (initialized to one) constantly increases when updated such that $\Delta g = 1$ when $g < 10$, $\Delta g = 10$ for $10 \leq g < 100$, and $\Delta g = 100$ otherwise.

The detailed learning algorithm is shown in the pseudocode in Appendix A.2.

3.3 Learning kinetics

Learning with BDMM using the Lagrangian function is better understood in the continuous time domain. We first address the kinetics of learning without the unconstrained-weight window $ucs(w)$. The change in the Lagrangian function $\mathcal{L}$ at a given learning step in the discrete time domain is equivalent to the derivative of $\mathcal{L}$ with time at time t in the continuous time domain, which is given by

$$\frac{d\mathcal{L}}{dt} = -\tau_W^{-1} \sum_{i=0}^{n_w} \left(\frac{\partial C}{\partial w_i} + \lambda_i \frac{\partial cs_i}{\partial w_i} \right)^2 + \tau_\lambda^{-1} \sum_{i=0}^{n_w} cs_i^2. \tag{11}$$

The Lagrange multiplier λ_i at time t is given by

$$\lambda_i(t) = \lambda_i(0) + \tau_\lambda^{-1} \int_0^t cs_i dt. \tag{12}$$

Eqs. (11) and (12) are derived in Appendix A.1. The constraint functions cs_i approach zero as the weights approach their corresponding quantized values q_i, and thus the Lagrange multipliers in Eq. (12) asymptotically converge to their limits.

At equilibrium, the Lagrange function is no longer time-dependent, i.e., $d\mathcal{L}/dt = 0$. This requires the Lagrange multipliers reaching their limits, which in turn requires the weights reaching their corresponding quantized values $\boldsymbol{W}^{*cs}$, leading to $\boldsymbol{cs} = \boldsymbol{0}$. For convenience, we define the integration of cs_i in Eq. (12) as $\Delta cs_i (\geq 0)$.

$$\Delta cs_i = \int_0^t cs_i dt, \quad \text{if } cs_i \to 0 \text{ as } t \to \infty. \tag{13}$$

Thus, the equilibrium Lagrange multiplier λ_i^{*cs} can be expressed as

$$\lambda_i^{*cs} = \lambda_i(0) + \tau_\lambda^{-1} \Delta cs_i. \tag{14}$$

Therefore, it is evident from Eq. (11) that the equilibrium leads to

$$\forall i, \frac{\partial C}{\partial w_i} = -\left(\lambda_i(0) + \tau_\lambda^{-1} \Delta cs_i \right) \frac{\partial cs_i}{\partial w_i}. \tag{15}$$

We consider sawtooth constraint functions with slopes $\pm s$, i.e., $\partial cs_i / \partial w_i = \pm s$, where $s > 0$. Eq. (8) is the case of $s = 2$. Generally, the Lagrange multiplier is initialized to zero, i.e., $\lambda_i(0) = 0$. Therefore, the gradient of loss function C at the equilibrium point $\boldsymbol{W}^{*cs}$ is given by

$$\forall i, \frac{\partial C}{\partial w_i} = \pm \tau_\lambda^{-1} s \Delta cs_i = \pm \lambda_i^{*cs} s. \tag{16}$$

6

Consider that the loss function C has the equilibrium point $\boldsymbol{W}^*\left(=\arg\min_{\boldsymbol{W}} C\right)$. Eq. (16) elucidates the increase of loss by attaining the equilibrium point $\boldsymbol{W}^{*cs}$. In this regard, λ_i^{*cs} corresponds to the cost of weight quantization. Assuming the convexity of the loss function C in a domain D including $\boldsymbol{W}^*$ and $\boldsymbol{W}^{*cs}$, $C\left(\boldsymbol{W}^{*cs}\right)$ keeps increasing as λ_i^{*cs} increases. If the initial pre-trained weights equal their corresponding quantized weights, i.e., $w_i^* = w_i^{*cs}$, then $\Delta cs_i = 0$, and thus $\lambda_i^{*cs} = 0$ according to Eq. (14). Eq. (16) consequently yields $\partial C / \partial w_i = 0$, indicating zero cost of quantization.

Considering the gradually vanishing unconstrained-weight window $ucs\left(w\right)$ yields the derivative of $\mathcal{L}$ with time at time t in the continuous time domain as follows.

$$\frac{d\mathcal{L}}{dt} = -\tau_W^{-1} \sum_{i=0}^{n_w} \left(\frac{\partial C}{\partial w_i} + \lambda_i ucs_i \frac{\partial Y_i}{\partial w_i} \right)^2 + \tau_\lambda^{-1} \sum_{i=0}^{n_w} \left(ucs_i Y_i \right)^2 . \qquad (17)$$

The derivation of Eq. (17) is given in Appendix A.3. Distinguishing the weights in the unconstrained-weight window D_{ucs} from the others at a given time t, Eq. (17) can be written by

$$\frac{d\mathcal{L}}{dt} = -\tau_W^{-1} \sum_{w_i \in D_{ucs}} \left(\frac{\partial C}{\partial w_i} \right)^2 - \sum_{w_i \notin D_{ucs}} \left[\tau_W^{-1} \left(\frac{\partial C}{\partial w_i} + \lambda_i \frac{\partial Y_i}{\partial w_i} \right)^2 - \tau_\lambda^{-1} Y_i^2 \right] . \qquad (18)$$

The latter term on the right-hand side of Eq. (18) indicates that the weights outside the window D_{ucs} are being quantized at the cost of increase of loss. However, as indicated by the former term, the weights in the window D_{ucs} are being optimized only to decrease the loss function with partially quantized weights. Compare this gradual quantization with abrupt quantization without the gradually vanishing unconstrained-weight window, where all weights are subject to simultaneous (abrupt) quantization. The gradual quantization allows the weights in the window to further reduce the loss function regarding the weights that have already been quantized or are being quantized, and thus the eventual cost of quantization is likely smaller than the simultaneous quantization case.

4 Experiments

To evaluate the performance of our algorithm, we trained three models (AlexNet, and ResNet-18 and 50) on the ImageNet dataset [37] with four different weight constraints (binary, ternary, and one-bit, and two-bit shift weight constraints). ImageNet consists of approximately 1.2 million training images and 50 thousands validation images. All training images were pre-processed such that they were randomly cropped and resized to 224×224 with mean subtraction and variance division. Additionally, random horizontal flipping and color jittering were applied. For validation, the images were resized to 256×256 and their centers in 224×224 were cropped. We evaluated the top-1 and top-5 classification accuracies on the validation set.

We considered binary, ternary, one-bit shift and two-bits shift weight constraints to validate the CBP algorithm as a general weight-quantization framework. For all cases, we introduced layer-wise scaling factors a such that $a^{(l)}$ (for the lth layer) is given by $a^{(l)} = \|\boldsymbol{W}^{(l)}\|_1 / n^{(l)}$, where $\boldsymbol{W}^{(l)}$ and $n^{(l)}$ denote the weight matrix of the lth layer and the number of elements of $\boldsymbol{W}^{(l)}$, respectively. As for [17] and [19], the weight matrices of the first and last layers were not quantized. The quantized weights employed for each constraint case is elaborated as follows.

Binary-weight constraint: A set of quantized weights $\boldsymbol{Q}$ is $\{-a, a\}$.

The other weight constraints: A set of quantized weights $\boldsymbol{Q}$ is $\left\{0, \pm 2^{-d}a\right\}_{d=0}^{D}$, where $D = 0, 1$ and 2 for the ternary, one-bit shift, and two-bit shift weight constraints. Each ternary weight needs 2-bit memory while each of one-bit and two-bit shift weight needs 3-bit memory.

We adopted the STE [38] to train the models such that the forward pass is based on quantized weights w_q,

$$w_q = q_1 + \sum_{i=1}^{n_q-1} \left(q_{i+1} - q_i \right) \left(sign\left(w - m_i \right) + 1 \right) / 2,$$

whereas the backward pass uses the real-valued weights w that are subject to quantization, $\partial \mathcal{L} / \partial w = \partial \mathcal{L} / \partial w_q$.

For all cases, the DNN was pre-trained using conventional backpropagation with full-precision weights and activations, which was followed by post-learning using CBP. We used the stochastic gradient descent with momentum to minimize the Lagrangian function $\mathcal{L}$ with respect to $\boldsymbol{W}$ and Adam [39] to maximize $\mathcal{L}$ with respect to $\boldsymbol{\lambda}$. The initial multiplier-learning rate η_λ and p_{max} were set to 10^{-4} and 20, respectively. The weight-learning rate η_W decreased to 10^{-1} times the initial rate when g reached 20 for all cases except GoogLeNet with the binary-weight constraint (the weight-learning rate decayed when $g = 200$). The hyperparameters used are shown in Appendix A.5, which were found using manual searches.

By asymptotically minimizing the Lagrangian function $\mathcal{L}$, the constraint function $\boldsymbol{cs}(\boldsymbol{W})$ approaches $\boldsymbol{0}$. The degree of constraint-failure per weight was evaluated based on the constraint-failure score (CFS), which is defined as

$$CFS = \frac{1}{n_w} \sum_{i=1}^{n_w} Y_i\left(w_i; \boldsymbol{Q}, \boldsymbol{M}\right), \tag{19}$$

where n_w denotes the total number of weights. The CBP algorithm was implemented in Python on a workstation (CPU: Intel Xeon Silver 4110 2.10GHz, GPU: Titan RTX).

It should be noted that we used CBP as a post-training method, so that the random seed effect is involved only when organizing the mini-batches. The accuracy deviation is consequently marginal.

Table 1: Top-1/Top-5 accuracy of AlexNet, ResNet-18, ResNet-50, and GoogLeNet on ImageNet

Algorithm	Binary	Ternary	One-bit shift	Two-bit shift	Full-precision
AlexNet					
BWN [17]	56.8%/79.4%	-	-	-	
ADMM [33]	57.0%/79.7%	58.2%/80.6%	59.2%/81.8%	60.0%/82.2%	
LQ-Nets [26]	-	60.5%/82.7%	-	-	60.0%/82.4%
TTQ [25]	-	57.5%/79.7%	-	-	
CBP	**58.0%/80.6%**	**58.8%/81.2%**	**60.8%/82.6%**	**60.9%/82.8%**	
ResNet-18					
BWN [17]	60.8%/83.0%	-	-	-	
TWN [19]	-	61.8%/84.2%	-	-	
INQ [23]	-	66.0%/87.1%	-	68.1%/88.4%	
ADMM [33]	64.8%/86.2%	67.0%/87.5%	67.5%/87.9%	68.1%/88.3%	69.6%/89.2%
QN [30]	66.5%/87.3%	69.1%/88.9%	69.9%/89.3%	70.4%/89.6%	
IR-Nets [28]	66.5%/86.8%	-	-	-	
LQ-Nets [26]	-	68.0%/88.0%	-	69.3%/88.3%	
TTQ [25]	-	66.6%/87.2%	-	-	
DSQ [29]	63.71%/-	-	-	-	
LS [31]	66.1%/86.5	-	-	-	
CBP	**66.6%/87.1%**	**69.1%/89.0%**	**69.6%/89.3%**	**69.6%/89.3%**	
ResNet-50					
BWN [17]	63.9%/85.1%	-	-	-	
TWN [19]	-	65.6%/86.5%	-	-	76.0%/93.0%
ADMM [33]	68.7%/88.6%	72.5%/90.7%	73.9%/91.5%	74.0%/91.6%	
QN [30]	72.8%/91.3%	75.2%/92.6%	75.5%/92.8%	76.2%/93.2%	
CBP	**74.4%/92.1%**	**75.1%/92.5%**	**76.0%/92.9%**	**76.0%/92.9%**	
GoogLeNet					
BWN [17]	59.0%/82.4%	-	-	-	
TWN [19]	-	61.2%/86.5%	-	-	71.0%/90.8%
ADMM [33]	60.3%/83.2%	63.1%/85.4%	65.9%/87.3%	66.3%/87.5%	
CBP	**64.0%/86.0%**	**66.0%/87.3%**	**69.8%/89.7%**	**70.5%/90.1%**	

4.1 AlexNet

AlexNet is a simple convolutional networks which consists of five convolutional layers and three fully-connected layers [2]. We used AlexNet with batch normalization [40] as in [17, 19, 33]. The initial weight-learning rate η_W was set to 10^{-3} for the binary- and ternary-weight constraints and

10^{-4} for the other constraints. The batch size was set to 256. We used a weight decay rate (L2-regularization) of 5×10^{-4}. The CBP algorithm exhibited state-of-the-art results as listed in Table. 1. The detailed behaviors of networks with binary- and ternary-weight constraints are addressed in Appendix B. The behaviors highlight asymptotic increases in the top-1 and top-5 recognition accuracy with asymptotic decrease in CFS. Consequently, the weight distribution bifurcates asymptotically, fulfilling the constraints imposed on the weights.

4.2 ResNet-18 and ResNet-50

We also evaluated our algorithm on ResNet-18 and ResNet-50 [15] which were pre-trained using conventional backpropagation. For ResNet-18, the initial weight-learning rate η_W was set to 10^{-3} for all constraint cases. The batch size was 256. For ReNet-50, the initial weight-learning rate η_W was set to 10^{-3} for binary- and ternary-weight constraints and 10^{-4} for the other cases. The batch size was set to 128. The weight decay rate (L2-regularization) was set to 10^{-4} for both ResNet-18 and ResNet-50. The results are summarized in Table 1, highlighting state-of-the-art performance compared with previous results. Notably, CBP with the one- and two-bit shift weight constraints almost reaches the performance of the full-precision networks. Particularly, for ResNet-50, CBP with the binary-weight constraint significantly outperforms other methods. The detailed behaviors of weight quantizations for ResNet-18 and ResNet-50 are addressed in Appendix B.

4.3 GoogLeNet

GoogLeNet consists of 22 layers organized with the inception modules [41]. We evaluated our algorithm on GoogLeNet which was pre-trained using conventional backpropagation. The weight-learning rate η_W was initially set to 10^{-3} for all constraint cases. The batch size, p_{max}, and weight decay rate were set to 256, 10 and 10^{-4}, respectively. For the binary-weight constraint case, the weight-learning rate η_W decreased to 10^{-1} times the initial rate when g reached 200. The results are summarized in Table 1. Notably, CBP significantly outperforms the previous results for all constraint cases. The detailed behaviors of weight quantizations for GoogLeNet are addressed in Appendix B.

5 Discussion

To evaluate the effect of the constraint function on training performance, we considered three different cases of post-training a DNN using CBP (i) with and (ii) without the unconstrained-weight window, and (iii) without the constraint function at all, i.e., conventional backpropagation with STE only. Because all DNNs in this work include STE, the comparison between these three cases highlights the effect of the constraint function in addition to STE. For all cases, pre-training using conventional backpropagation preceded the three different post-training schemes. Table 2 addresses the comparison, highlighting accuracy and CFS improvement in Case (i) over Case (iii). This indicates that CBP allows the DNN to learn features while the weights are being quantized by the constraint function with the gradually vanishing unconstrained-weight window. On the contrary, CBP without the unconstrained-weight window (Case (ii)) rather degraded the accuracy compared with Case (iii), whereas the improvement on CFS was significant. This may be because the constraint function without unconstrained-weight window strongly forced the weights to be quantized without learning the features.

Table 2: Top-1 accuracy of ResNet-18 trained in various conditions

Post-training algorithm	Accuracy	CFS
CBP with update of g	66.6%/87.1%	1.19×10^{-3}
CBP without unconstrained-weight window	60.2%/82.7%	1.05×10^{-5}
Backpropagation+STE	64.6%/85.9%	3.58×10^{-2}

We used manual searches for the hyperparameters, weight-learning rate η_W, multiplier-learning rate η_λ, multiplier update scheduling variable p_{max}, and unconstrained-weight window variable Δg. We used identical parameters η_λ, p_{max}, and Δg for all four models, each with the four distinct constraints, i.e., 12 cases in total. CBP is unlikely susceptible to the hyperparameters for different

models. Therefore, the hyperparameters used in this work may serve as the decent initial values for other models.

CBP needs floating-point operations (FLOPs) for the Lagrange multiplier update in addition to FLOPs for the weight update, which causes additional computational complexity. Given that a Lagrange multiplier is assigned to each weight, the additional complexity scales with the number of weights. The total computational complexity of CBP exceeds the conventional backpropagation by approximately 2% for ResNet-18 and ResNet-50 whereas by approximately 25% for AlexNet. The complexity estimation is elaborated in Appendix A.6.

We used CBP as a post-training method. That is, the networks considered were pre-trained using conventional backpropagation. Applying CBP to untrained networks hardly reached the accuracies of classification listed in Table 1. When efficiency in training is of the most important concern, CBP may not be the best choice. However, when efficiency in memory usage is of the most important concern, CBP may be the optimal choice with regard to its excellent learning capability with maximum 3-bit weight precision, which almost reaches the classification accuracy of the full-precision networks. The use of one-bit or two-bit shift weights can avoid multiplication operations that consume a considerable amount of power, so that it can significantly improve computational efficiency. Additionally, CBP is not a method tailored to particular models. Therefore, our work may have a broader impact on various application domains where memory capacity is limited and/or computational efficiency is of significant concern.

6   Conclusion

In this study, we proposed the CBP algorithm that trains DNNs by simultaneously considering both loss and constraint functions. It enables the implementation of any well-defined set of constraints on weights in a common training framework, unlike previous algorithms for weight quantization, which were tailored to particular constraints. The evaluation of CBP on ImageNet with with different constraint functions (binary, ternary, one-bit shift and two-bit shift weight constraints) demonstrated its high capability, highlighting its state-of-the-art accuracy of classification.

References

[1] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097–1105.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] T. N. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 8614–8618.

[5] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] K. Lee, D. Yoo, W. Jeong, and S. Han, "SIMPLE-NN: an efficient package for training and executing neural-network interatomic potentials," *Computer Physics Communications*, vol. 242, pp. 95–103, 2019.

[8] G. Kim, V. Kornijcuk, D. Kim, I. Kim, C. S. Hwang, and D. S. Jeong, "Artificial neural network for response inference of a nonvolatile resistance-switch array," *Micromachines*, vol. 10, no. 4, p. 219, 2019.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 2672–2680.

[10] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2016.

[11] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," in *5th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2017.

[12] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 2172–2180.

[13] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 214–223.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2015.

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

[16] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 3123–3131.

[17] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, 2016, pp. 525–542.

[18] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," in *4th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2016.

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

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

[21] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary neural networks with fine-grained quantization," 2017, *arXiv*:1705.01462.

[22] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 963–971.

[23] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," in *5th International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2017.

[24] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv*:1602.02830.

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

[26] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[27] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "Deepshift: Towards multiplication-less neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2021, pp. 2359–2368.

[28] H. Qin, R. Gong, X. Liu, M. Shen, Z. Wei, F. Yu, and J. Song, "Forward and backward information retention for accurate binary neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2250–2259.

[29] R. Gong, X. Liu, S. Jiang, T. Li, P. Hu, J. Lin, F. Yu, and J. Yan, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[30] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-s. Hua, "Quantization networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[31] H. Pouransari, Z. Tu, and O. Tuzel, "Least squares binary quantization of neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[32] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, T. Elgindi, C.-A. Deledalle, and H. Esmaeilzadeh, "Waveq: Gradient-based deep quantization of neural networks through sinusoidal adaptive regularization," *arXiv preprint arXiv:2003.00146*, 2020.

[33] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with admm," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[34] D. Bertsekas and W. Rheinboldt, *Constrained Optimization and Lagrange Multiplier Methods*, ser. Computer science and applied mathematics. Elsevier Science, 2014.

[35] D. Luenberger and Y. Ye, *Linear and Nonlinear Programming*. Springer, 2015.

[36] J. C. Platt and A. H. Barr, "Constrained differential optimization," in *Proceedings of the 1987 International Conference on Neural Information Processing Systems*, 1987, p. 612–621.

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[38] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR, Conference Track Proceedings*, 2015.

[40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

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