OvSW: Overcoming Silent Weights for Accurate Binary Neural Networks

Jingyang Xiang[1], Zuohui Chen[2], Siqi Li[1], Qing Wu[3], and Yong Liu[1,4](✉)

[1] APRIL Lab, Zhejiang University, Hangzhou, China
[2] Zhejiang University of Technology, Hangzhou, China
[3] College of Computer Science, Hangzhou Dianzi University, Hangzhou, China
[4] Huzhou Institute, Zhejiang University, Hangzhou, China
yongliu@iipc.zju.edu.cn

Abstract. Binary Neural Networks (BNNs) have been proven to be highly effective for deploying deep neural networks on mobile and embedded platforms. Most existing works focus on minimizing quantization errors, improving representation ability, or designing gradient approximations to alleviate gradient mismatch in BNNs, while leaving the weight sign flipping, a critical factor for achieving powerful BNNs, untouched. In this paper, we investigate the efficiency of weight sign updates in BNNs. We observe that, for vanilla BNNs, over 50% of the weights remain their signs unchanged during training, and these weights are not only distributed at the tails of the weight distribution but also universally present in the vicinity of zero. We refer to these weights as "silent weights", which slow down convergence and lead to a significant accuracy degradation. Theoretically, we reveal this is due to the independence of the BNNs gradient from the latent weight distribution. To address the issue, we propose Overcome Silent Weights (OvSW). OvSW first employs Adaptive Gradient Scaling (AGS) to establish a relationship between the gradient and the latent weight distribution, thereby improving the overall efficiency of weight sign updates. Additionally, we design Silence Awareness Decaying (SAD) to automatically identify "silent weights" by tracking weight flipping state, and apply an additional penalty to "silent weights" to facilitate their flipping. By efficiently updating weight signs, our method achieves faster convergence and state-of-the-art performance on CIFAR10 and ImageNet1K dataset with various architectures. For example, OvSW obtains 61.6% and 65.5% top-1 accuracy on the ImageNet1K using binarized ResNet18 and ResNet34 architecture respectively. Codes are available at https://github.com/JingyangXiang/OvSW.

Keywords: Binary Neural Networks · Silent Weights · Adaptive Gradient Scaling · Silence Awareness Decaying

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/978-3-031-73414-4_1.

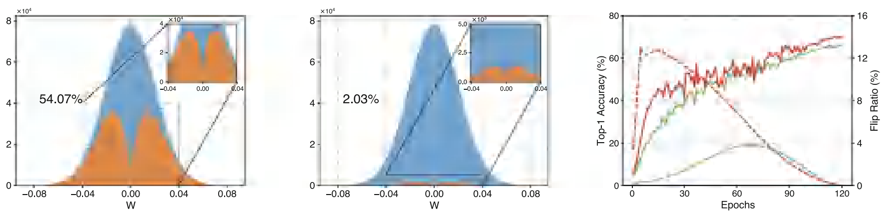
A. Leonardis et al. (Eds.): ECCV 2024, LNCS 15091, pp. 1–18, 2025.
https://doi.org/10.1007/978-3-031-73414-4_1

1 Introduction

Deep neural networks (DNNs) have shown tremendous success in various computer vision tasks, including image classfication [15,23], object detection [10,14,43], and semantic segmentation [12,35]. However, the remarkable performance is always attributed to deeper and wider architectures [15,46], which comes with expensive memory and computational overhead and makes it challenging to deploy DNNs on resource-constrained edge platforms.

The community has been delving into model compression, which aims to reduce inference overhead for DNNs while preserving their performance. Typical techniques include, but are not limited to, efficient architecture design [7,58], neural architecture search [8,47], network pruning [28,36], knowledge distillation [18,27], and network quantization [4,11,61]. Among them, network quantization is widely suggested as a promising solution. It effectively enhances memory efficiency and execution speed on embedding platforms by reducing the weight and activation bits and replacing expensive floating-point arithmetic with relatively inexpensive fixed-point arithmetic. Extensive studies [11,20,54,57] have demonstrated its effectiveness and yielded light and efficient DNNs.

In this paper, we focus on studying binary neural networks (BNNs), which are an aggressive form of quantized neural networks (QNNs). BNNs binarize both weights and activations to discrete values ($\{+1, -1\}$), which can be represented by 1-bit and reduce memory usage by 32×. On the other hand, by employing efficient bitwise operations instead of floating-point ones, BNNs also significantly reduce computation complexity, providing 58× speedup as reported by XNOR-Net [42]. These characteristics make them well-suited for deployment on low-power embedding platforms, including FPGA, ASICs, and IoT devices [19].



(a) Vanilla **(b)** OvSW **(c)** Top-1 Acc & Flip Ratio

Fig. 1. (a) and (b): Histogram of the initialized weight distribution (blue) and the weights that never update signs throughout training (orange). 54.07% and 2.03% represent the ratio of the corresponding orange area to the blue. (c) Top-1 Accuracy (solid lines) and Flip Ratio (dashed lines) are for Vanilla (green) and OvSW (red). (a), (b) and Flip Ratio in (c) is for layer4.conv2.weight. (Color figure online)

However, the application of BNNs is still limited due to constraining both weights and activations to 1-bit significantly reducing accuracy compared to full-precision models. To address this problem, many methods have been developed to reduce quantization error and enhance the representation capability, thereby closing the performance gap between BNNs and their real-valued counterparts.

Although these methods have effectively narrowed the performance gap, they still overlook the essence of BNN optimization is to update the signs of latent weights [17], instead of updating their values. In other words, if the signs of the latent weights don't change, the BNNs are hardly ever updated. To demonstrate the weight sign flipping of vanilla BNNs, we train binarized ResNet18 on CIFAR100 with 120 epochs and plot the flipping information for layer4.conv2.weight. As shown in Fig. 1a, more than 50% of the weights never change their signs throughout training process, which slows down convergence and leads to a significant accuracy degradation for BNNs. We denote the part of weights that hardly change their signs during training as "silent weights".

It's worth noting that, as a study mostly related to ours, ReCU [53] proposes a rectified clamp unit to revive the "dead weights" for updating, which refers to a group of weights that are distributed at the tails of the weight distribution and barely update their signs during training. They claim that the magnitudes of latent weights do not contribute to the forward propagation. If two weights have the same sign, they have the same effect on the forward propagation. Intuitively, the signs of the weights around the zero are easily changed, while weights distributed at the tails of the weight distribution tend to have difficulties in changing their signs. Although ReCU facilitates the updating for "dead weights", it ignores the fact, the gradient distribution across layers and weights have great distinction. Weights with larger magnitudes are intuitively more difficult to flip their signs, but small weights may also suffer from silence once their gradients are relatively small compared to their magnitudes. As shown in Fig. 1a, for a vanilla BNN, these weights are not only distributed at the tails of the weight distribution but also universally present in the vicinity of zero. As a result, it is still sub-optimal and remains to be improved.

In this paper, we reveal that the root cause of the large number of "silent weights" is attributed to the independence of the BNN gradients from the latent weight distribution via a systematical and theoretical analysis. To this end, two simple yet effective techniques, including Adaptive Gradient Scaling (AGS) and Silence Awareness Decaying (SAD), are introduced to Overcome Silent Weights (OvSW). Specifically, AGS adaptively scales the gradients by establishing a relationship between the gradients and the latent weight distribution, thereby improving the overall efficiency of weight sign updates. Meanwhile, SAD automatically identifies "silent weights" by tracking weight sign flipping state and applies an additional penalty to them, further promoting the updating efficiency of their signs. Benefiting from AGS and SAD, OvSW enables efficient flipping of weight signs, significantly accelerates the convergence and promotes the performance for BNNs as shown in Fig. 1b and Fig. 1c. In comparison to the expensive computational cost of forward and backward propagation, OvSW introduces negligible extra overhead to the training process, maintaining its efficiency. Furthermore, since OvSW aims to facilitate the weight sign flipping for

BNNs, which is orthogonal to the previous work, it has good compatibility and can be used as plug-and-play modules to enhance other BNNs' performance.

The contributions of our work are highlighted as follows:

- We are the first to find that a large number of weights in BNNs fail to update their signs throughout the training. These "silent weights" are not only distributed at the tails of the weight distribution but also universally present in the vicinity of zero. Theoretically, we reveal this is attributed to the independence of the gradients from the distribution of the latent weights.
- We propose to overcome "silent weights" with two novel techniques: (1) an adaptive gradient scaling method to scale the gradients according to the distribution of the latent weights, which improves overall efficiency in updating signs for weights; (2) a silence awareness decaying strategy to identify "silent weights" by tracking weight sign flipping state and introduce an additional penalty to them, further facilitating the flipping of their signs.
- Extensive experiments for various BNNs including ResNet18/20 [15] and VGGsmall [46] on CIFAR10 [22] and ResNet18/34 [15] on ImageNet1K [44] demonstrate the effectiveness of our method. For example, OvSW achieves 61.6% and 65.5% top-1 accuracy on the ImageNet for binarized ResNet18 and ResNet34, improving ReCU [53] by 0.6% and 0.4% respectively. Besides, for ResNet18, OvSW achieves 2.00% and 2.83% improvement on CIFAR100 [22] when combined with AdaBin (enhancing representation ability) and RBNN (training-aware gradient approximation) respectively, demonstrating its good compatibility with the other methods.

2 Related Work

Great efforts have been put into reducing the performance gap between BNNs and their real-valued counterparts. The pioneering BNN work dates back to Courbariaux *et al.* [5], which binarizes weights and activations to +1 or -1 by sign function. However, this aggressive approach limits the representation ability of BNNs to the binary space, resulting in a significant accuracy degradation. To reduce the quantization error and improve their accuracy, XNOR-Net [42] introduces a scaling factor obtained through the ℓ_1-norm of weights or activations. Furthermore, XNOR-Net++ [3] merges two scale factors from the weights and activations into a trainable parameter and optimizes them via backpropagation. ABC-Net [31] employs a linear combination of multiple binary weight sets to approximate the real-valued weights and alleviate the information loss. Bi-Real Net [34] connects the real-valued activations to the consecutive block via an identity shortcut, which significantly enhances network representation ability while incurring negligible computational overhead. AdaBin [48] introduces adaptive binary sets to fit different distributions, enhancing binarized representation ability. UaBNN [59] introduces an uncertainty-aware BNN to reduce these weights' uncertainties. INSTA-BNN [24] controls the quantization threshold in an input instance-aware manner, taking higher-order statistics into account.

Apart from enhancing the representation ability of BNNs, gradient estimation is also one of the critical research directions, since gradients in the sign function are almost zero everywhere. Straight through estimator (STE) [1] is the most widely used function to enable the gradient to backpropagate. However, the gradient error is huge for STE and will accumulate during backpropagation, leading to instability in training and severe accuracy degradation. To alleviate this, Bi-Real Net [34] utilizes a piecewise polynomial function as the approximation function. IR-Net [40] proposes an error decay estimator and RBNN [30] employs a training-aware approximation function to replace the sign function. EWGS [25] takes discretization error between input and output into account and introduces element-wise gradient scaling to adaptively scale up or down each gradient element. ReSTE [49] revises the original STE and balance the estimating error and the gradient stability well. All of these methods effectively reduce the gradient error and achieve consistent improvement in both training stability and accuracy compared to the vanilla STE.

3 Background

We briefly review the optimization process of BNNs in this section. Given a DNN, we denote $\mathcal{W}_j \in \mathsf{R}^{C^j_{\texttt{out}} \times C^j_{\texttt{in}} \times K^j_{\texttt{h}} \times K^j_{\texttt{w}}}$ as the real-valued weight in the j-th layer, $C^j_{\texttt{out}}$, $C^j_{\texttt{in}}$, $K^j_{\texttt{h}}$ and $K^j_{\texttt{w}}$ are the number of output channels, input channels, kernel height, and kernel width, respectively. Let the real-valued activation be $\mathcal{A}_j$, then the convolution process can be formulated as

$$\mathcal{A}_{j+1} = \mathcal{W}_j * \mathcal{A}_j, \tag{1}$$

where $*$ represents the standard convolution operation.

BNNs aim to binarize weights $\mathcal{W}_j$ and activations $\mathcal{A}_j$ to discrete values ($\{+1, -1\}$) through sign function:

$$\hat{x} = \operatorname{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0, \\ -1, & \text{otherwise.} \end{cases} \tag{2}$$

To reduce the quantization error in BNNs, XNOR-Net [42] introduces two scale factors to approximate the binarized weights $\hat{\mathcal{W}}^b_j$ and activation $\hat{\mathcal{A}}^b_j$. Furthermore, XNOR-Net++ [3] proposes fusing the activation and weight scaling factors into one and optimizing it via backpropagation. This approach significantly outperforms XNOR-Net within the same computational budget and has been widely adopted by recent works [33,34,40,53]. Following them, we denote the scaling factor as α_j. Then the binary convolution operation can be formulated as

$$\mathcal{A}_{j+1} = \left(\hat{\mathcal{A}}_j \circledast \hat{\mathcal{W}}_j\right) \odot \alpha_j, \tag{3}$$

where $\circledast$ is the efficient XNOR and Bitcount operation and $\odot$ is the hadamard product. In the implementation of BNNs, $\mathcal{A}_{j+1}$ will be processed through several layers, *e.g.*, Batch Normalization (BN) layer, non-linear activation layer, and max-pooling layer. In this section, we omit these operations for simplicity.

To train a BNN, the forward propagation includes Eq. 2 and Eq. 3, where their real-value counterparts $\mathcal{W}_j$ and $\mathcal{A}_j$ are used for calculating gradients and updating during the backpropagation. However, the sign is not differentiable, thereby gradient estimation is important in BNNs. Following the previous studies, we use straight-through estimator (STE) [1] to approximate the gradient of the loss w.r.t the weight $w \in \mathcal{W}$:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial \hat{w}} \cdot \frac{\partial \hat{w}}{\partial w} \approx \frac{\partial \mathcal{L}}{\partial \hat{w}}, \tag{4}$$

where $\mathcal{L}$ denotes the loss function. For the gradient w.r.t the activation $a \in \mathcal{A}$, we adopt the piece-wise polynomial gradient estimation function [34] as follows:

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial \mathcal{L}}{\partial \hat{a}} \cdot \frac{\partial \hat{a}}{\partial a} \approx \frac{\partial \mathcal{L}}{\partial \hat{a}} \cdot \frac{\partial F(a)}{\partial a}, \tag{5}$$

where

$$\frac{\partial F(a)}{\partial a} = \begin{cases} 2 + 2a, \text{ if } -1 \leq a < 0, \\ 2 - 2a, \text{ if } \quad 0 \leq a < 1, \\ 0, \qquad\quad \text{otherwise.} \end{cases} \tag{6}$$

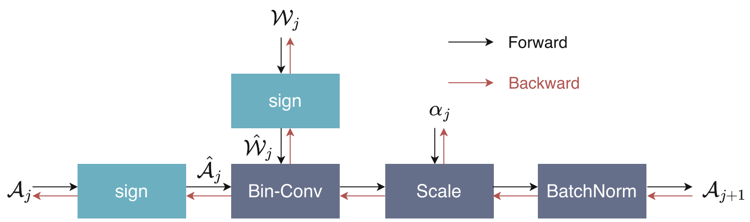


Fig. 2. Forward and backward computation graph for binary convolutional operation with quantization aware training.

4 Methodology

In this section, we show that the distribution of gradients and weights for BNNs is independent by systematical and theoretical analysis. Then, we propose Adaptive Gradient Scaling (AGS) to scale the gradient and introduce Silence Awareness Decaying (SAD) to detect "silent weights", moving them towards zero. Both of them can enhance the efficiency of weight sign flipping.

4.1 The Independence of the Gradient and Weight Distribution

Figure 2 demonstrates forward and backward computation graph for binary convolutional (Bin-Conv) operation. As seen, Bin-Conv is often followed by a BN layer. We can conclude that once two BNNs $\mathcal{N}$ and $\mathcal{N}^{'}$ satisfy:

$$\hat{\mathcal{W}}_j = \hat{\mathcal{W}}_j^{'}, \alpha_j = \alpha_j^{'}, \mathrm{BN}_j = \mathrm{BN}_j^{'}, \forall j, \tag{7}$$

for the same input $\hat{\mathcal{A}}_j$, the output $\mathcal{A}_{j+1}$ and $\mathcal{A}'_{j+1}$ can be formulated as:

$$\mathcal{A}_{j+1} = \mathrm{BN}\left(\left(\hat{\mathcal{A}}_j \circledast \hat{\mathcal{W}}_j\right) \odot \alpha_j\right) = \mathrm{BN}'\left(\left(\hat{\mathcal{A}}_j \circledast \hat{\mathcal{W}}'_j\right) \odot \alpha'_j\right) = \mathcal{A}'_{j+1}. \tag{8}$$

Through mathematical induction, we can know that the $\mathcal{N}$ and $\mathcal{N}'$ will have the same output and loss, ***i.e.*** $\mathcal{L} = \mathcal{L}'$. Therefore, the backpropagation for them is:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}_j} = \frac{\partial \mathcal{L}}{\partial \mathcal{A}_{j+1}} \frac{\partial \mathcal{A}_{j+1}}{\partial \hat{\mathcal{W}}_j} \frac{\partial \hat{\mathcal{W}}_j}{\partial \mathcal{W}_j} = \frac{\partial \mathcal{L}'}{\partial \mathcal{A}'_{j+1}} \frac{\partial \mathcal{A}'_{j+1}}{\partial \hat{\mathcal{W}}'_j} \frac{\hat{\mathcal{W}}'_j}{\partial \mathcal{W}'_j} = \frac{\partial \mathcal{L}'}{\partial \mathcal{W}'_j}. \tag{9}$$

Without loss of generality, we can obtain that once two networks satisfy the conditions in Eq. 7, the gradients obtained from the backpropagation are the same (a more generalized scenario are extrapolated in Appendix B). The gradient has nothing to do with the magnitude of weight and weight with larger magnitude does not necessarily corresponding to larger gradient. Therefore, although intuitively, weight with large magnitude tends to remain its sign unchanged, it is also potential for weight with small magnitude to suffer from silence if its gradient is relatively small compared to its magnitude. Furthermore, if we assume the weights further satisfy $\mathcal{W}'_j = \gamma \mathcal{W}_j$, where $\gamma > 1$ and networks are optimized via vanilla SGD, the optimization of t-step can be formulated as:

$$\begin{aligned} \mathcal{W}'_j(t+1) &= \mathcal{W}'_j(t) - \beta(t)\frac{\partial \mathcal{L}'(t)}{\partial \mathcal{W}'_j(t)} = \gamma \mathcal{W}_j(t) - \beta(t)\frac{\partial \mathcal{L}(t)}{\partial \mathcal{W}_j(t)} \\ &= (\gamma - 1)\mathcal{W}_j(t) + \mathcal{W}_j(t) - \beta(t)\frac{\partial \mathcal{L}(t)}{\partial \mathcal{W}_j(t)} = (\gamma - 1)\mathcal{W}_j(t) + \mathcal{W}_j(t+1), \end{aligned} \tag{10}$$

where $\beta(t)$ is the learning rate at t-step. Equation 10 can be viewed as a variant of exponential moving averages and the effect of $\mathcal{W}_j(t)$ on $\mathcal{W}'_j(t+1)$ will gradually increases as γ increases. Once k is large enough:

$$\lim_{\gamma \to \infty} \mathcal{W}'_j(t+1) = \lim_{\gamma \to \infty} \left[(\gamma - 1)\mathcal{W}_j(t) + \mathcal{W}_j(t+1)\right] = (\gamma - 1)\mathcal{W}_j(t). \tag{11}$$

The signs of $\mathcal{W}'_j(t+1)$ will be the same as $\mathcal{W}_j(t)$.

However, BNNs binarize $\mathcal{W}'_j$ to $\{+1, -1\}$ at each training step. Once the signs of $\mathcal{W}'_j$ do not change, the BNNs hardly ever update. It not only slows down convergence, but also leads to significant accuracy degradation. Please see Appendix C for a detailed ablation analysis for γ.

4.2 Adaptive Gradient Scaling

To improve the efficiency of the update, an intuitive approach is to scale $\beta(t)$ to $\gamma\beta(t)$. In this case, $\mathcal{W}'_j(t+1)$ can be formulated as:

$$\begin{aligned} \mathcal{W}'_j(t+1) &= \mathcal{W}'_j(t) - \gamma\beta(t)\frac{\partial \mathcal{L}(t)}{\partial \mathcal{W}'_j(t)} \\ &= \gamma \mathcal{W}_j(t) - \gamma\beta(t)\frac{\partial \mathcal{L}(t)}{\partial \mathcal{W}_j(t)} = \gamma \mathcal{W}_j(t+1). \end{aligned} \tag{12}$$

However, selecting the appropriate scaling factor is tricky. Meanwhile, an inappropriate scaling factor may make weights with small magnitudes correspond to large gradients, leading them to oscillate frequently in $\{+1, -1\}$ and introducing instability to BNNs training.

To overcome this issue, we introduce "Adaptive Gradient Scaling" (AGS). Let $\mathcal{G}_j \in \mathsf{R}^{C^j_{\mathsf{out}} \times C^j_{\mathsf{in}} \times K^j_{\mathsf{h}} \times K^j_{\mathsf{w}}}$ denote the gradient with respect to $\mathcal{W}_j$, *i.e.*, $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_j}$, and $\| \cdot \|_F$ denote the Forbenius norm, *i.e.*, the k-th filter norm in $\mathcal{W}_j$ can be formulated as:

$$\|\mathcal{W}^k_j\|_F = \sqrt{\sum_{l=1}^{C^j_{\mathsf{in}}} \sum_{m=1}^{K^j_{\mathsf{h}}} \sum_{n=1}^{K^j_{\mathsf{w}}} \mathcal{W}^{k,l,m,n}_j}. \tag{13}$$

The AGS algorithm is motivated by the observation that the ratio of the norm of $\mathcal{G}^k_j$ to $\mathcal{W}^k_j$ ($\frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F}$) provides a simple measure of how much a single gradient descent step will change the original weight $\mathcal{W}_j$. For instance, if we train BNNs via vanilla SGD without momentum, then $\frac{\|\triangle \mathcal{W}^k_j\|_F}{\|\mathcal{W}^k_j\|_F} = \beta \frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F}$, where the parameter update for the $\mathcal{W}^k_j$ is given by $\triangle \mathcal{W}^k_j = -\beta \mathcal{G}^k_j$. We can conclude that the small value of $\frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F}$ during training is the root cause why a large number of weights fail to flip signs and adaptively scaling $\frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F}$ plays a crucial role in promoting the update efficiency for BNNs. Specifically, in AGS algorithm, $\mathcal{G}^{k,l,m,n}_j$ is scaled as:

$$\overline{\mathcal{G}}^{k,l,m,n}_j = \begin{cases} \lambda \frac{\|\mathcal{W}^k_j\|_F}{\|\mathcal{G}^k_j\|_F} \mathcal{G}^{k,l,m,n}_j & \text{if } \frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F} < \lambda, \\ \mathcal{G}^{k,l,m,n}_j & \text{otherwise.} \end{cases} \tag{14}$$

where λ is a scalar scaling threshold to limits the lower bound of $\frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F}$. Ablation analysis for λ can be found in Sect. 5.3.

AGS and "Adaptive Gradient Clipping" (AGC) [2] are closely related but fundamentally different, as the former restricts the lower bound of $\frac{\|\mathcal{G}^k_j\|_F}{\|\mathcal{W}^k_j\|_F}$ to enhance the flipping efficiency of BNN weight signs, while the latter restricts upper bound and is designed to improve training stability. AGS also can be viewd as a adaptive varient to Layer-wise Adaptive Rate Scaling (LARS) [56], which sets the norm of update parameter to a fixed ratio of the parameter norm, and completely ignores the gradient magnitude to real-valued networks. Although LARS is also able to improve the accuracy for BNNs, we find that LARS and AGS is much different in the update of gradient momentum and doing so degrades performance compared to AGS. More details for comparing AGS with LARS and ablation analysis for LARS can be found in Appendix D.

4.3 Silence Awareness Decaying

We propose another approach (Silence Awareness Decaying, SAD) orthogonal to AGS to detect and prevent "silent weights". Specifically, we track the flipping

Algorithm 1. Overview of the OvSW method.

Input: A minibatch of inputs and their labels, real-valued weights $\mathcal{W}(t)$, scaling factor $\alpha(t)$, λ for AGS, τ for SAD, $(\mathcal{S}(t),\ m,\ \sigma)$ for flipping state detection.
Output: Updated $\mathcal{W}(t+1)$, $\alpha(t+1)$ and $\mathcal{S}(t+1)$.

while Forward propagation **do**
  $\hat{\mathcal{A}}_j \leftarrow \text{sign}(\mathcal{A}_j)$.
  $\hat{\mathcal{W}}_j \leftarrow \text{sign}(\mathcal{W}_j)$.
  Computing features via Eq. 2 and Eq. 3.
  Computing loss $\mathcal{L}$.
end while
while Backward propagation **do**
  Computing $\frac{\partial \mathcal{L}}{\partial \mathcal{W}_j}$, *i.e.* $\mathcal{G}_j$, and $\frac{\partial \mathcal{L}}{\partial \mathcal{A}_j}$ via Eq. 4 and Eq. 5&Eq. 6
  Scaling $\mathcal{G}_j$ to $\overline{\mathcal{G}}_j$ via Eq. 14.
  Adding penalties to $\overline{\mathcal{G}}_j$ via Eq. 16.
  Update $\mathcal{W}_j(t+1)$ and $\alpha_j(t+1)$ via SGD optimizer.
  Update $\mathcal{S}_j(t+1)$ via Eq. 15.
end while

state of $\mathcal{W}_j$ over time using an exponential moving average (EMA) strategy, which is formulated as:

$$\mathcal{S}_j(t) = m \cdot \mathcal{S}_j(t-1) + (1-m) \cdot \frac{|\text{sign}\left(\mathcal{W}_j\left(t\right)\right) - \text{sign}\left(\mathcal{W}_j\left(t-1\right)\right)|_{abs}}{2}, \quad (15)$$

where m is the momentum and $\mathcal{S}_j$ is the auxiliary variable. Nagel *et al.* [38] employ this technique to identify and dampen oscillations prematurely while we employ it to identify "silent weights" and dynamically introduce additional weight penalties to move them towards zero. In our algorithm, we think if $\mathcal{S}_j^{k,l,m,n}$ is less than a pre-defined threshold σ, its corresponding weight $\mathcal{W}_j^{k,l,m,n}$ is considered as a "silent weight" and will be applied with an additional penalty. The silence awareness decaying process is formulated as:

$$\overline{\overline{\mathcal{G}}}_j^{k,l,m,n}(t) = \begin{cases} \overline{\mathcal{G}}_j^{k,l,m,n}(t) + \gamma \mathcal{W}_j^{k,l,m,n}(t), & \text{if } \mathcal{S}_j^{k,l,m,n}(t) < \sigma, \\ \overline{\mathcal{G}}_j^{k,l,m,n}(t), & \text{otherwise,} \end{cases} \quad (16)$$

where γ is the proportion of penalty term.

It is worth noting that while both AGS and SAD improve the efficiency of weight sign flipping, they solve the problem in fundamentally different ways. AGS facilitates sign flipping for the whole weights by adaptively scaling the gradient, while SAD detects "silent weights" by tracking their flipping state and applies additional penalties. In Sect. 5.3, we show that both methods achieve significant performance improvement and they are complementary to each other. As an algorithm guideline, the pseudo-code of OvSW is provided in Algorithm 1.

5 Experiment

In this section, we conduct extensive image classification experiments for OvSW and compare it to state-of-the-art (SOTA) methods on CIFAR10 and ImageNet1K with various architectures. Then, we discuss the hyperparameter settings for OvSW, including λ for AGS and σ for SAD and convergence speed on CIFAR100. We also conduct ablation study to demonstrate the compatibility and visualize the loss landscape for OvSW. Finally, we deploy OvSW to a real-world mobile device to exhibit its efficiency. To train OvSW, we use one NVIDIA RTX 3090 on the CIFAR10 and CIFAR100 and four on the ImageNet1K. All experiments are implemented on PyTorch [39].

Table 1. Performance comparison with SOTAs on CIFAR10.We report the Top-1 Accuracy performance on ResNet18, ResNet20, and VGGsmall. W/A denotes the bit-width of weights/activations.

Networks	Methods	Bit-width (W/A)	Top-1 Acc.(%)
ResNet18	Full-precision	32/32	94.8
	IR-Net [40]	1/1	91.5
	RBNN [30]	1/1	92.2
	CMIM [45]	1/1	92.2
	SiMaN [29]	1/1	92.5
	ReCU [53]	1/1	92.8
	OvSW (Ours)	1/1	**93.2**
ResNet20	Full-precision	32/32	92.1
	SLB [55]	1/1	85.5
	FDA-BNN [52]	1/1	86.2
	IR-Net [40]	1/1	86.5
	CMIM [45]	1/1	87.3
	SiMaN [29]	1/1	87.4
	ReCU [53]	1/1	87.4
	OvSW (Ours)	1/1	**87.7**
VGGsmall	Full-precision	32/32	94.1
	DoReFa [60]	1/1	90.2
	RAD [6]	1/1	90.5
	RBNN [30]	1/1	91.3
	DSQ [11]	1/1	91.7
	Proxy-BNN [16]	1/1	91.8
	SLB [55]	1/1	92.0
	ReCU [53]	1/1	92.2
	FDA-BNN [52]	1/1	92.5
	SiMaN [29]	1/1	92.5
	OvSW (Ours)	1/1	**92.8**

Table 2. Performance comparison with SOTAs on ImageNet1K.We report the Top-1 and Top-5 Accuracy performance on ResNet18 and ResNet34. W/A denotes the bit-width of weights/activations. * means using the two-step training setting as ReActNet.

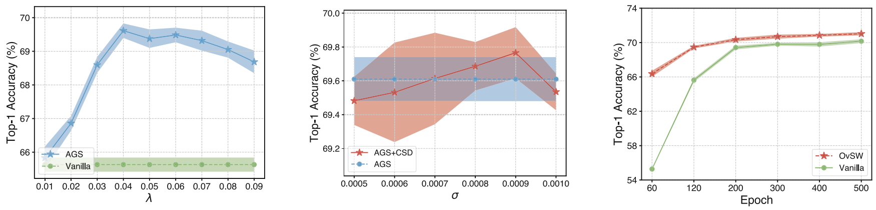
Model	Method	Bit-width (W/A)	Top-1 Acc.(%)	Top-5 Acc.(%)
ResNet18	Full-precision	32/32	69.6	89.2
	XNOR [42]	1/1	51.2	73.2
	BiReal [34]	1/1	56.4	79.5
	IR-Net [40]	1/1	58.1	80.0
	RBNN [30]	1/1	59.9	81.9
	SiMaN [29]	1/1	60.1	82.3
	FDA-BNN [52]	1/1	60.2	82.3
	ReCU [53]	1/1	61.0	82.6
	OvSW (Ours)	1/1	**61.6**	**83.1**
	ReActNet [33]	1/1	65.9	86.1
	ReCU [53]	1/1	66.4	86.5
	OvSW* (Ours)	1/1	**66.6**	**86.7**
ResNet34	Full-precision	32/32	73.3	91.3
	XNOR++ [3]	1/1	57.1	79.9
	BiReal [34]	1/1	62.2	83.9
	IR-Net [40]	1/1	62.9	84.1
	RBNN [30]	1/1	63.1	84.4
	SiMaN [29]	1/1	63.9	84.8
	CMIM [45]	1/1	65.0	85.7
	ReCU [53]	1/1	65.1	85.8
	OvSW (Ours)	1/1	**65.5**	**86.1**

5.1 Results on CIFAR10

We trained OvSW for CIFAR10 with 600 epochs, where the batch size is set to 256 and the initial learning rate to 0.1, decaying with CosineAnealing. We adopt SGD optimizer with a momentum of 0.9 and weight decay of 5e-4 and employ the same data augmentation in ReCU [53]. λ and σ are set to 0.04 and 9e-4 respectively. We compare OvSW with IR-Net [40], RBNN [30], CMIM [45], SiMaN [29], ReCU [53], SLB [55], FDA-BNN [52], DoReFa [60], RAD [6], DSQ [11], and Proxy-BNN [16]. As shown in Table 1, OvSW achieves the best performance among all methods. For ResNet18, OvSW obtains 93.2% top-1 accuracy, which outperforms SiMaN and ReCU by 0.7% and 0.4% respectively, reducing the accuracy gap between BNNs and full-precision model to 1.6%. In addition, it yields 87.7% and 92.8% top-1 accuracy on ResNet20 and VGGsmall, which succeeds ReCU and SiMaN by 0.3% and 0.3% respectively.

5.2 Results on ImageNet1K

On ImageNet1K, OvSW is trained from scratch.We train OvSW with 200 epochs, where the batch size is set to 512 and the initial learning rate is set to 0.1, decaying with CosineAnnealing. We adopt SGD optimizer with a momentum of 0.9 and weight decay of 1e-4 and the data augmentation is the same as ReCU [53]. λ and σ are set to 0.02 and 2e-5 respectively. We demonstrate the ImageNet1K performance of ResNet18/34 and compare OvSW with SOTA methods, including one-stage training methods XNOR [42], BiReal [34], IR-Net [40], RBNN [30], SiMaN [29], FDA-BNN [52], ReCU [53], CMIM [45], and two-stage training method [37] adopted by ReActNet [33]. As shown in Table 2, OvSW also achieves the best performance. For ResNet18, OvSW achieves 61.6% top-1 and 83.1% top-5 accuracy compared to ReCU's 61.0% top-1 and 82.6% top-5 accuracy, which demonstrates the efficiency of overcoming the "silent weights". For ResNet34, OvSW achieves 65.5% top-1 accuracy, which is also better than ReCU. We further compare OvSW with the two-stage training method ReActNet. OvSW obtains 66.6% top-1 accuracy, succeeding ReActNet and ReCU by 0.7% and 0.2% respectively.



(a) AGS vs Vanilla **(b)** AGS+SAD (OvSW) vs AGS **(c)** Convergence

Fig. 3. Top-1 accuracy (mean±std) of binarized ResNet18 w.r.t different values of λ (a), σ (b) and epoch (c) on CIFAR100.

5.3 Ablation Analysis

We investigate the effectiveness of hyper-parameters, including λ and σ, convergence speed, different components, and compatibility through ablation analysis. All the following results are based on binarized ResNet18 for CIFAR100.

λ for AGS and σ for SAD. We first compare AGS with vanilla BNNs and analyze the λ for AGS. In Fig. 3a, we present the results for nine different settings of λ, which vary from 0.01 to 0.09. As seen, the appropriate λ can significantly improve the performance of BNNs, and $\lambda = 0.04$ achieves the best performance. If λ is too small, the sign of the weights can flip inefficiently; while λ is too large, the sign of the weights can flip dramatically, introducing instability to the training. Based on this result, we further introduce SAD to AGS (λ = 0.04) to analyze σ. Figure 3b demonstrates that the performance of the model can be further improved by identifying "silent weights" and applying additional penalties to them via SAD ($\sigma = 0.0009$).

Convergence for OvSW. To verify that OvSW facilitates the flipping of weight signs and thus improves the efficiency of convergence, we fix the λ and σ to 0.04 and 0.0009 respectively, and record the top-1 accuracy over the training epoch from 60 to 500. As shown in Fig. 3c, OvSW effectively accelerates training convergence and achieves better performance. For example, OvSW achieves 66.37±0.35% top-1 accuracy with only 60 traning epochs, while vanilla BNNs only reaches 55.28±0.07% and 65.23±0.21% top-1 accuracy with 60 and 120 training epochs respectively. It indicates that OvSW can achieve significant performance gains in scenarios with limited training resources, such as training the network on edge devices. Meanwhile, although increasing epochs improve the final performance for both OvSW and vanilla BNNs, OvSW consistently outperforms the vanilla BNNs.

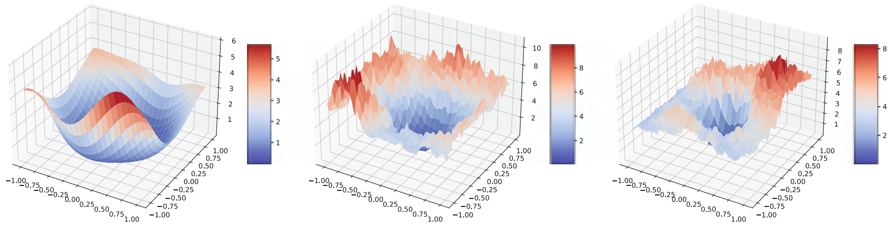
Table 3. Left: Ablation study of different components in OvSW. Right: Applying OvSW as a plug-and-play module to other methods.

AGS	SAD	mean±std (%)
✗	✗	65.23±0.21
✓	✗	69.61±0.22
✗	✓	69.45±0.18
✓	✓	69.77±0.15

Method	mean±std (%)
AdaBin	70.56±0.19
AdaBin+OvSW	72.56±0.11
RBNN	67.15±0.23
RBNN+OvSW	69.98±0.13

Components. AGS and SAD promote sign flipping for overall weights and "silent weights" respectively. To prove that they are orthogonal to each other, we conduct components ablation study for different modules and show the results in Table 3 (left). As seen, AGS and SAD achieve 69.61±0.22% and 69.45±0.18% top-1 accuracy respectively and both of them succeed vanilla BNNs. It shows that they are effective in improving the efficiency of weight signs flipping. By combining them, OvSW (AGS+SAD) further achieves the best top-1 accuracy with 69.77±0.15%.

Compatibility. We show the good compatibility of OvSW by inserting it as a plug-and-play module into the current state-of-the-art methods, including AdaBin [48] and RBNN [30]. The former introduces an adaptive binary set to enhance the feature representation of BNNs, while the latter proposes a training-aware approximation function to reduce the gradient estimation error during training. Table 3 (right) demonstrates OvSW achieves 72.56±0.11% top-1 accuracy on AdaBin and 69.98±0.13% on RBNN, succeeding their original performance. This result indicates that OvSW has good compatibility and can effectively enhance the performance of existing methods.



(a) Full-precision (b) XNOR-Net++ (c) OvSW

Fig. 4. 3D visualization of the loss surfaces of ResNet18 on CIFAR100, which is used to enable comparisons of sharpness/flatness of different methods.

5.4 Loss Landscape Visualization

BNNs restrict the weights and activations to discrete values, which naturally limits the representational capacity of the model and further result in disparate optimization landscapes compared to real-valued ones [32]. As illustrated in Fig. 4, we follow the method in [26] to plot the actual optimization landscape about our OvSW and compare it with the same architecture to real-valued and XNOR-Net++. As seen, our OvSW has a significantly smoother loss-landscape and minor loss elevation compared to XNOR-Net++, which confirms the effectiveness of OvSW in the BNN optimization.

5.5 Deployment Efficiency

We implement 1-bit models on the M1 Pro, which features 8 high-performance Firestorm cores and 2 efficient Icestorm cores in a hybrid design. The Firestorm cores can be clocked up to 3.2 GHz, while the Icestorm cores can reach 2.1 GHz. Our OvSW shows significant efficiency gains when deployed on real-world mobile devices, as evidenced by practical speed evaluations. To make our inference framework BOLT [9] compatible with OvSW, we leverage the ARM NEON SIMD instruction SSHL. We compare OvSW with 32-bit and 16-bit backbones. As shown in Table 4, OvSW inference speed is substantially faster with the highly efficient BOLT framework in a single thread. For instance, OvSW achieves an acceleration rate of about 3.47× on ResNet18 compared to its 32-bit counterpart. For the ResNet34 backbone, OvSW achieves a 3.42× acceleration rate with the BOLT framework on hardware, which is significant for computer vision applications on real-world edge devices. At the same time, OvSW can save memory by a factor of 16.64× and 21.16×, which demonstrates its potential for applications with limited memory resources.

Table 4. Comparing OvSW (1-bit) with 32-bit and 16-bit backbones on M1 Pro.

Network	W/A	Size (MB)	Memory Saving	Latency (ms)	Acceleration
ResNet18	32/32	46.76	-	27.67	-
	16/16	23.38	2×	16.65	1.66×
	1/1	2.81	16.64×	7.97	3.47×
ResNet34	32/32	87.19	-	48.82	-
	16/16	43.60	2×	29.01	1.68×
	1/1	4.12	21.16×	14.28	3.42×

6 Conclusion

BNN is a crucial method to compress deep learning models and reduce inference overhead. In this paper, systematically and theoretically, we prove the distribution of gradients is independent of latent weights, which is the root cause of inefficient updating and performance degradation of BNNs. To this end, Adaptive Gradient Scaling (AGS) and Silence Awareness Decaying (SAD), are proposed to Overcome Silent Weights (OvSW) and achieve SOTA performance on BNNs. Specifically, AGS adaptively scales the gradient based on the distribution of weights, improving the efficiency of sign flipping for the overall weights. SAD can effectively measure the states of weight flipping, detect the "silent weights" and introduce additional penalty to them to facilitate their flipping. Extensive experiments demonstrate that OvSW can achieve notable performance gains over the SOTA methods on various datasets and networks. In addition, OvSW has better convergence efficiency and excellent compatibility, which can be combined with existing methods to further enhance the performance of BNNs.

References

1. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
2. Brock, A., De, S., Smith, S.L., Simonyan, K.: High-performance large-scale image recognition without normalization. In: International Conference Machine Learning, pp. 1059–1071. PMLR (2021)
3. Bulat, A., Tzimiropoulos, G.: Xnor-net++: Improved binary neural networks. arXiv preprint arXiv:1909.13863 (2019)
4. Chen, J., Liu, L., Liu, Y., Zeng, X.: A learning framework for n-bit quantized neural networks toward fpgas. IEEE Trans. Neural Netw. Learn. Syst. **32**(3), 1067–1081 (2020)
5. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
6. Ding, R., Chin, T.W., Liu, Z., Marculescu, D.: Regularizing activation distribution for training binarized deep networks. In: IEEE Conference Computer Vision Pattern Recognition, pp. 11408–11417 (2019)

7. Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., Sun, J.: Repvgg: making vgg-style convnets great again. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 13733–13742 (2021)
8. on, Duan, Y., et al.: Transnas-bench-101: improving transferability and generalizability of cross-task neural architecture search. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 5251–5260 (2021)
9. Feng, J.: Bolt. https://github.com/huawei-noah/bolt (2021)
10. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 580–587 (2014)
11. Gong, R., et al.: Differentiable soft quantization: bridging full-precision and low-bit neural networks. In: International Conference on Computer Vision, pp. 4852–4861 (2019)
12. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: International Conference Computer Vision, pp. 2961–2969 (2017)
13. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: International Conference Computer Vision, pp. 1026–1034 (2015)
14. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Trans. Pattern Anal. Mach. Intell. **37**(9), 1904–1916 (2015)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 770–778 (2016)
16. He, X., et al.: ProxyBNN: learning binarized neural networks via proxy matrices. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12348, pp. 223–241. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58580-8_14
17. Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.T., Nusselder, R.: Latent weights do not exist: rethinking binarized neural network optimization. In: Advance Neural Information Processing System, vol.32 (2019)
18. Hinton, G., Vinyals, O., Dean, J., et al.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 **2**(7) (2015)
19. Horowitz, M.: 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-state Circuits Conference Digest of Technical Papers (ISSCC), pp. 10–14. IEEE (2014)
20. Jacob, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: IEEE Conference Computer Vision Pattern Recognition, pp. 2704–2713 (2018)
21. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
22. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. Neural Inform. Process. Syst. **25** (2012)
24. Lee, C., Kim, H., Park, E., Kim, J.J.: Insta-bnn: binary neural network with instance-aware threshold. In: International Conference Computer Vision, pp. 17325–17334 (2023)
25. Lee, J., Kim, D., Ham, B.: Network quantization with element-wise gradient scaling. In: IEEE Conference Computer Vision Pattern Recognition, pp. 6448–6457 (2021)

26. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. Adv. Neural Inform. Process. Syst. (2018)
27. Li, S., Lin, M., Wang, Y., Fei, C., Shao, L., Ji, R.: Learning efficient gans for image translation via differentiable masks and co-attention distillation. IEEE Trans, Multimedia (2022)
28. f Lin, M., et al.: Hrank: filter pruning using high-rank feature map. In: IEEE Conference Computer Vision Pattern Recognition, pp. 1529–1538 (2020)
29. Lin, M., Ji, R., Xu, Z., Zhang, B., Chao, F., Lin, C.W., Shao, L.: Siman: Sign-to-magnitude network binarization. IEEE Trans. Pattern Anal. Mach, Intell (2022)
30. Lin, M., et al.: Rotated binary neural network. Adv. Neural Inform. Process. Syst. **33**, 7474–7485 (2020)
31. Lin, X., Zhao, C., Pan, W.: Towards accurate binary convolutional neural network. Adv. Neural Inform. Process. Syst. **30** (2017)
32. Liu, Z., Shen, Z., Li, S., Helwegen, K., Huang, D., Cheng, K.T.: How do adam and training strategies help bnns optimization. In: International Conference Machine Learning, pp. 6936–6946. PMLR (2021)
33. Liu, Z., Shen, Z., Savvides, M., Cheng, K.-T.: ReActNet: towards precise binary neural network with generalized activation functions. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) ECCV 2020. LNCS, vol. 12359, pp. 143–159. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58568-6_9
34. Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., Cheng, K.T.: Bi-real net: enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In: European Conference Computer Vision, pp. 722–737 (2018)
35. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 3431–3440 (2015)
36. Luo, J.H., Wu, J.: Autopruner: an end-to-end trainable filter pruning method for efficient deep model inference. Pattern Recogn. **107**, 107461 (2020)
37. Martinez, B., Yang, J., Bulat, A., Tzimiropoulos, G.: Training binary neural networks with real-to-binary convolutions. arXiv preprint arXiv:2003.11535 (2020)
38. Nagel, M., Fournarakis, M., Bondarenko, Y., Blankevoort, T.: Overcoming oscillations in quantization-aware training. In: International Conference Machince Learning, pp. 16318–16330. PMLR (2022)
39. Paszke, A., et al.: PyTorch: an Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., Garnett, R. (eds.) Adv. Neural Inform. Process. Syst, pp. 8024–8035. Curran Associates, Inc. (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
40. Qin, H., et al.: Forward and backward information retention for accurate binary neural networks. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 2250–2259 (2020)
41. Rajbhandari, S., Rasley, J., Ruwase, O., He, Y.: Zero: memory optimizations toward training trillion parameter models. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–16. IEEE (2020)
42. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: imagenet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer Vision – ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV,

pp. 525–542. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
43. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1137–1149 (2016)
44. Russakovsky, O., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)
45. Shang, Y., Xu, D., Zong, Z., Yan, Y.: Network binarization via contrastive learning. arXiv preprint arXiv:2207.02970 (2022)
46. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
47. Su, X., et al.: Prioritized architecture sampling with monto-carlo tree search. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 10968–10977 (2021)
48. Tu, Z., Chen, X., Ren, P., Wang, Y.: Adabin: improving binary neural networks with adaptive binary sets. In: European Conference Computer Vision, pp. 379–395. Springer (2022). https://doi.org/10.1007/978-3-031-20083-0_23
49. Wu, X.M., Zheng, D., Liu, Z., Zheng, W.S.: Estimator meets equilibrium perspective: a rectified straight through estimator for binary neural networks training. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 17055–17064 (2023)
50. Xu, S., et al.: Resilient binary neural network. AAAI. **37**, 10620–10628 (2023)
51. Xu, S., et al.: Recurrent bilinear optimization for binary neural networks. Eur. Conf. Comput. Vis. 19–35. Springer (2022). https://doi.org/10.1007/978-3-031-20053-3_2
52. Xu, Y., Han, K., Xu, C., Tang, Y., Xu, C., Wang, Y.: Learning frequency domain approximation for binary neural networks. Adv. Neural Inform. Process. Syst. **34**, 25553–25565 (2021)
53. Xu, Z., et al.: Recu: reviving the dead weights in binary neural networks. In: International Conference Computer Vision, pp. 5198–5208 (2021)
54. Yang, J., et al.: Quantization networks. In: IEEE Conference Computing Vision Pattern Recognition, pp. 7308–7316 (2019)
55. Yang, Z., et al.: Searching for low-bit weights in quantized neural networks. Adv. Neural Inform. Process. Syst. **33**, 4091–4102 (2020)
56. You, Y., Gitman, I., Ginsburg, B.: Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888 (2017)
57. Zhang, D., Yang, J., Ye, D., Hua, G.: Lq-nets: learned quantization for highly accurate and compact deep neural networks. In: European Conference on Computer Vision, pp. 365–382 (2018)
58. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: IEEE Conference on Computer Vision Pattern Recognition, pp. 6848–6856 (2018)
59. Zhao, J., Yang, L., Zhang, B., Guo, G., Doermann, D.S.: Uncertainty-aware binary neural networks. In: IJCAI. pp. 3441–3447 (2021)
60. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)
61. Zhu, F., et al.: Towards unified int8 training for convolutional neural network. In: IEEE Conference on Computer Vision Pattern Recognition,. pp. 1969–1979 (2020)