Margin-Aware Binarized Weight Networks for Image Classification

Ting-Bing Xu[1,2(✉)], Peipei Yang[1], Xu-Yao Zhang[1], and Cheng-Lin Liu[1,2]

[1] National Laboratory of Pattern Recognition,
Institute of Automation of Chinese Academy of Sciences, Beijing, China
{tingbing.xu,ppyang,xyz,liucl}@nlpr.ia.ac.cn
[2] University of Chinese Academy of Sciences (UCAS), Beijing, China

Abstract. Deep neural networks (DNNs) have achieved remarkable successes in many vision tasks. However, due to the dependence on large memory and high-performance GPUs, it is extremely hard to deploy DNNs on low-power devices. For compressing and accelerating deep neural networks, many techniques have been proposed recently. Particularly, binarized weight networks, which store one weight using only one bit and replace complex floating operations with simple calculations, are attractive from the perspective of hardware implementation. In this paper, we propose a simple strategy to learn better binarized weight networks. Motivated by the phenomenon that the stochastic binarization approach usually converges with real-valued weights close to two boundaries $\{-1, +1\}$ and gives better performance than deterministic binarization, we construct a margin-aware binarization strategy by adding a weight constraint into the objective function of deterministic scheme to minimize the margins between real-valued weights and boundaries. This constraint can be easily realized by a Binary-L2 regularization without suffering from the complex random number generation. Experimental results on MNIST and CIFAR-10 datasets show that the proposed method yields better performance than recent network binarization schemes and the full precision network counterpart.

Keywords: Deep network compression · Binarized weight networks
Binary-L2 regularization

1 Introduction

Deep Neural Networks (DNNs) have been successfully applied to a wide range of vision tasks, such as object recognition [1–4], object detection [5–7] and semantic segmentation [8,9]. Despite their huge power and success, DNNs require large memory overhead and high-performance hardware for implementation, which makes it difficult to run DNNs on some resource-limited devices (e.g., Smartphone, Smart wearable devices and Embedded devices) or dedicated Deep Learning hardware directly. For instance, the famous AlexNet [1] requires about 61×10^6 parameters (232 MB[1]) with 725 million FLoating-point OPerations

[1] MB: MegaBytes storage requirement.

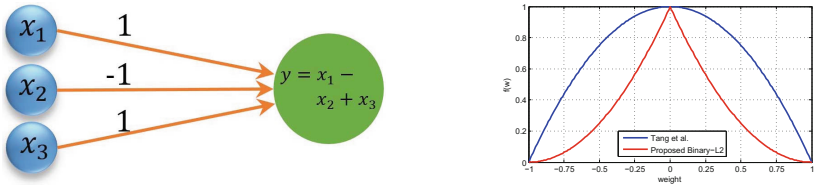
Y. Zhao et al. (Eds.): ICIG 2017, Part I, LNCS 10666, pp. 590–601, 2017.
https://doi.org/10.1007/978-3-319-71607-7_52

(FLOPs), and VGG-16 [2] involves 138×10^6 parameters (527 MB) with 15.3 billion FLOPs. Therefore, reducing the parameters of DNNs and removing most arithmetic operations are crucial problems for many real-time deployments.

To break the above bottleneck of DNNs' application on resource-limited devices, various deep model compression methods have been proposed recently. The first strategy proposes new architectures to reduce the parameters, especially in the fully-connected (*fc*) layer, such as GoogLenet [3], ResNet [4], Network in Network [10], SqueezeNet [11]. They replace *fc* layers with global average pooling layer to reduce parameters dramatically, and design diversified compact blocks (e.g., Inception module [3], "bottleneck" building block of ResNet [4], mlpconv layer [10] and Fire module [11]) to reduce parameters of convolutional layers further. Finally, the deeper and light networks are constructed for higher performance. However, such kind of light networks do not take care of the number of FLOPs (speed) adequately. For instance, the ResNet-18 [4] network still needs about 1.8 billion FLOPs to classify an image of size 224×224.

The second strategy utilizes compression techniques to remove redundant parameters [12] of the classical DNNs for less storage and faster inference, including tensor decomposition with low-rank approximation [13–15], sparse constraints [16], hash function [17], threshold based reduction with retraining [18], integration of hash trick and Huffman coding [19], etc. Though these approaches achieve significant compression ratio for *fc* layers, they are not well suitable for the new DNN models because the new DNNs structures (e.g., ResNet [4], SqueezeNet [11]) have discarded *fc* layers. While the training process is also complicated since the repetitional fine-tuned operations and difficultly recover the original accuracy produced by information loss of pre-trained models.

To obtain better compression and acceleration, the third strategy which uses lower bits to store the weight is proposed. For example, Vanhoucke et al. [20] directly replaced the 32-bit "float" with 8-bit "char" to shrink total memory footprint by 4×. Later, binarized weight networks were proposed, where the weights were approximated with binary values $\{-1, +1\}$ (in Fig. 1(a)) resulting in 32× memory saving and simple addition or subtraction operations. Soudry et al. [21] were the first to show that the binarized DNNs can obtain good accuracy by the Expectation BackPropagation (EBP) algorithm which updates the posterior distributions over real-valued weights. Courbariaux et al. [22,23] designed the BinaryConnect algorithm and BinarizedNet method by extending the probabilistic



(a) 1-bit stores 1-weight (b) weight function

Fig. 1. Binarized weight networks and weight constraint terms.

idea behind EBP algorithm, where they directly utilized binarized parameters to train Binarized Neural Networks using standard back propagation. For better performance, Rastegari et al. [24] proposed Binary-Weight-Networks and XNOR-Networks by combining optimized scale factors with binarized weights or activations. However, the overall compression rate of XNOR-Network was severely degraded because both the first and last layers were of full precision to guarantee reasonable accuracy. To address this problem, Tang et al. [25] adopted a scalar layer and new regularization term to mitigate severe accuracy drop when the last layer was also binarized, but the compression rate is still limited since the first layer is still of full precision.

In this paper, we focus on binarized weight networks which are attractive from the perspective of hardware, and study to achieve both better performance and simplicity of implementation. In the experiments of BinaryConnect [22], the stochastic binarization scheme is of better performance than deterministic binarization, but harder to implement as it needs the hardware to generate random bits. To analyse the difference between the two schemes, we carefully examine the weight distribution after the algorithm converges and find an interesting phenomenon. For the stochastic approach, all the weights gather close to the boundaries $\{-1, +1\}$ by a very small margin, while for the deterministic approach, there are considerable amount of weights scattered over the interval and some even stuck around 0, leaving a large margin to the boundaries.

This phenomenon motivates our proposed margin-aware binarization strategy which is constructed by introducing a weight constraint into the objective function of deterministic binarization to minimize the loss between real-valued weights and corresponding binarized weights (boundaries). The constraint formulated as a *Binary-L2* term gives the weights awareness of the margins to boundaries, and the proposed method can effectively improve the performance of deterministic binarization approach, which can be easily realized without random number generation. Extensive experiments demonstrate that our Binarized Weight Networks algorithm yields better performance than recent network binarization approaches and the corresponding full precision networks.

It's worth noting that although the regularization used in our method is similar to the one in [25] to some extent, the contributions of the two are obviously different in the following perspectives: (1) in Fig. 1(b), the weight gradient in our Binary-L2 term becomes stronger as it approaches 0 by a large margin to the boundaries, which shrinks the margin effectively and drives the weights towards two boundaries $\{-1, +1\}$ friendly. On the contrary, the gradient of regularization term in [25] tends to vanish around 0, and its large gradient at ± 1 is likely to stick the weights at their current binary values. (2) While the motivation in [25] is to apply a regularization on the weights, we propose the Binary-L2 based on the analysis of the discovered phenomenon, which provides a perspective on explaining why this term is effective to improve the performance. A detailed discussion on the potential mechanism behind this phenomenon is given in this paper.

The reminder of this paper is organized as follows. Section 2 presents details about our Binarized-Weight-Networks. Section 3 shows experimental results on several common datasets and the analysis. Concluding remarks will be drawn in Sect. 4.

2 Binarized Weight Networks

In the past, people thought that low-precision networks would suffer from obvious performance drop, especially under the extreme scenario: Binarized DNNs were believed to be highly destructive to the network accuracy [26]. However, recent binarized based networks [21,22,24] have exhibited promising results when the weights or activations of DNNs are binarized to ± 1. BinaryConnect [22] algorithm constrains the weights to be only two possible values $\{-1, +1\}$ and use the binarized weights in the forward and backward propagations. To exploit the Stochastic Gradient Descent (SGD) technique to optimize the network, there is a real-valued weight corresponding to each binarized weight. In every step, the real-valued weights are updated according to the gradients of the binarized weights. Courbariaux et al. [22] explain that the constrained function of binarization can be regarded as a new regularization (a binary sampling process) for real-valued weights which is able to improve generalization capability, like Dropout [27] and DropConnect [28] operations.

Motivation: In BinaryConnect [22], there are two types of binarization operation: deterministic (det.) and stochastic (stoch.) binarization. The deterministic binarization operation constrains each real-valued weight to binary values ± 1 using the following sign function:

$$w_i^b = \text{sign}(w_i) = \begin{cases} +1 & \text{if } w_i \geq 0, \\ -1 & \text{if } w_i < 0, \end{cases} \tag{1}$$

where w_i^b is the binarized weight and w_i is corresponding real-valued weight. This binarization scheme is easy to be implemented on various devices and effective in practice [22,24]. On the other hand, the stochastic binarization requires generating random bits that is much more computationally costly and more difficult to be implemented on low-power hardware devices. However, the stochastic binarization method performs better than deterministic scheme in the experiments of BinaryConnect [22]. To explore the reason, we observe the distribution of the real-valued weights after training stage. An interesting phenomenon is observed in deterministic scheme that although a majority of weights converge to the two boundaries $\{-1, +1\}$, there are also quite a few weights scattering along the interval, leaving a large margin to the boundaries. On the contrary, in the stochastic scheme, all weights converge to two boundaries $\{-1, +1\}$ friendly with a very small margin. According to this phenomenon, we guess there may be some relationship between the margin and the performance. Thus we attempt to design a simple approach to eliminate such a margin in deterministic binarization scheme for pursuing better performance of Binarized-Weight-Networks (in Fig. 2).

Method: Through investigations, we find an analogous problem in visual similarity search field. For these algorithms, how to encode high-dimensional image data into short and effective binary codes based on hashing approaches is a crucial problem. Liong et al. [29] insert three constrained conditions into the

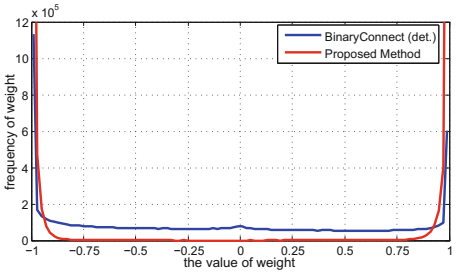


Fig. 2. Distribution of real-valued weights after training time.

objective function of DNNs to make the different bits of the learned binary vector $\mathbf{h}^b$ possess independent and discriminative properties, one of which aims at minimizing the loss between real-valued feature descriptor and the learned binary vector by the following function:

$$\min \frac{1}{2}\|\mathbf{h} - \text{sign}(\mathbf{h})\|_2^2 = \min \frac{1}{2}\|\mathbf{h} - \mathbf{h}^b\|_2^2, \tag{2}$$

where $\mathbf{h}$ denotes float values of the last *fc* layer and $\mathbf{h}^b$ is binarized values of $\mathbf{h}$. It is similar to our weight constraint term.

As discussed in our motivation, we are trying to get a better performance by reducing the margins between real-valued weights and the boundaries $\{-1, +1\}$, which is a similar task as the one above. Thus, we introduce a weight constraint deterministic binarization scheme. The constraint term formulated as a Binary-L2 regularization (standard norm) can effectively minimize the loss between the learned weights of DNNs and the corresponding binarized weights, which is defined as:

$$\begin{aligned} \min \frac{1}{2}\|\mathbf{W} - \text{sign}(\mathbf{W})\|_F^2 &= \min \frac{1}{2}\|\mathbf{W} - \mathbf{W}^b\|_F^2 \\ &= \min \frac{1}{2}\||\mathbf{W}| - \mathbf{1}\|_{l2}^2. \end{aligned} \tag{3}$$

Then the optimization objective function is as follows:

$$\begin{aligned} &\min_{\mathbf{W}} J(\mathbf{W}) = L(\mathbf{W}^b, \mathbf{X}, \mathbf{y}) + \frac{\lambda}{2}\||\mathbf{W}| - \mathbf{1}\|_{l2}^2, \\ &s.t. \quad \mathbf{W}^b = \text{sign}(\mathbf{W}), \end{aligned} \tag{4}$$

where L is common loss function (such as SoftmaxLoss or HingeLoss), the second term is our Binary-L2 regularization and λ controls the balance between the two terms. $\mathbf{X}$ and $\mathbf{y}$ represents the training samples and corresponding label vector respectively. $\mathbf{W}$ is the weight tensor of the entire DNNs and $\mathbf{1}$ is the counterpart identity tensor. The first loss function (L) is minimized for high classification accuracy and the Binary-L2 term gives weights awareness to converge to the two edges $\{-1, +1\}$ friendly during parameter update for discovering better accuracy of Binarized-Weight-Networks.

Algorithm 1. SGD Training for the Binarized-Weight-Networks

Require: A minibatch of training samples ($\mathbf{X}$, $\mathbf{y}$), network layer number M, iterative number t, learning rate η, scaled coefficient γ, Eq. 4 is the optimization objective function, previous real-valued parameters $\mathbf{W}^{t-1}$ (weights) and $\mathbf{b}^{t-1}$ (biases).

Ensure: updated parameter $\mathbf{W}^t$ and $\mathbf{b}^t$.

1. BinaryFoward propagation:

for $m = 1$ to M layers **do**

$\mathbf{W}_m^b = \text{sign}(\mathbf{W}_m^{t-1})$

Compute the weighted sum $\mathbf{Z}_m$ with $\mathbf{X}_{m-1}$, $\mathbf{W}_m^b$ and $\mathbf{b}_m^{t-1}$

Apply batch-normalization [31] and ReLU [32] to $\mathbf{Z}_m$ to obtain $\mathbf{X}_m$

end for

Compute the loss L using $\mathbf{X}_{\text{M}}$ and $\mathbf{y}$

2. BinaryBackward propagation:

Initialize output layer's activations gradient $\frac{\partial L}{\partial \mathbf{X}_{\text{M}}}$

for $m =$ M to 2 layers **do**

Compute $\frac{\partial L}{\partial \mathbf{X}_{m-1}}$ using $\frac{\partial L}{\partial \mathbf{X}_m}$ and $\mathbf{W}_m^b$

end for

3. Parameter update:

for $m = 1$ to M layers **do**

Compute $\frac{\partial L}{\partial \mathbf{W}_m^b}$ and $\frac{\partial L}{\partial \mathbf{b}_m^{t-1}}$ knowing $\frac{\partial L}{\partial \mathbf{X}_m}$ and $\mathbf{X}_{m-1}$

$\mathbf{W}_m^t \leftarrow \text{clip}\big(\mathbf{W}_m^{t-1} - \eta \cdot \gamma_m \left(\frac{\partial L}{\partial \mathbf{W}_m^b} + \frac{\partial R}{\partial \mathbf{W}_m^{t-1}}\right)\big)$ // R is the new constraint

$\mathbf{b}_m^t \leftarrow \mathbf{b}_m^{t-1} - \eta \frac{\partial L}{\partial \mathbf{b}_m^{t-1}}$

end for

Training Process: The entire Binarized-Weight-Networks can be trained in an end-to-end manner by back propagation (BP) algorithm and SGD optimization or Adam learning rule [30]. Same as BinaryConnect, there is a corresponding real-valued weight for each binary weight for parameter update. In every step, the previous real-valued weights are updated directly by the gradients, which are computed with respect to the binarized weights. Then the binarized weights are obtained using sign function Eq. 1. After training stage, the float weights are discarded for memory saving since the forward propagation only need binarized weights. The whole procedure is illustrated in Algorithm 1, where R denotes Binary-L2 term (the second term in Eq. 4).

Implementation Detail: Since existing deep learning platforms do not support binarized layers and Binary-L2 regularization, we need to implement binarized convolutional layer, binarized fully-connected layer and special Binary-L2 regularization based on Caffe [33] toolbox. For each real-valued weight, the Binary-L2 regularization term is continuously differentiable and its derivative is define as:

$$\frac{\partial R}{\partial w_i} = \lambda(|w_i| - 1)\text{sign}(w_i). \tag{5}$$

In addition, we also adopted several useful techniques as used in BinaryConnect algorithm [22] to achieve better Binarized-Weight-Networks. These techniques include Batch Normalization (BN), scaling the weights learning rates (γ in Algorithm 1) and clipping operation. The real-valued weights are easily updated in the range $[-1, 1]$ by these operations. The weight learning rates are scaled to enlarge the gradient magnitude with the coefficient γ from "Xavier" [34] algorithm, thus we set $\lambda\gamma = 0.0001$–0.0005 as the ratio of regularization that is similar to L2 weight decay in DNNs. Clipping operation constrains weights to $[-1, 1]$ interval for preventing weights far from the boundary $\{-1, +1\}$ overly.

3 Experiments

We conducted experiments on two widely used datasets (natural image classification task) to evaluate the effectiveness of our proposed binarization scheme with routine convolutional neural networks (CNN). In our experiments, the basic block structure of our binarized CNN contains convolutional or fully-connected layer with binarized weights, BN [31] layer and ReLU [32] activation layer. We use weights initialization of uniform distribution $[-1, 1]$, Binary-L2 regularization and Adam optimization for all Binarized-Weight-Networks, while the "Xavier" [34] initialization and Adam optimization are adopted for the corresponding full precision weight networks. The experimental settings and results are described in following.

3.1 MNIST

The MNIST [35] dataset contains handwritten digit images (0–9) including 60k training images and 10k testing images. The images are of normalized size 28×28. An architecture similar to LeNet5 [35] is used in our experiment, which is also adopted in [36] and shown in Fig. 3 (left).

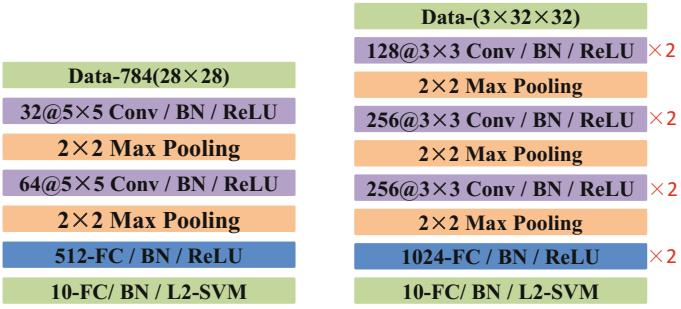


Fig. 3. Left: MNIST dataset and Right: CIFAR-10 dataset.

In this picture, the "32@5×5 Conv" denotes that a convolutional layer possess 32 convolutional kernels with size 5×5, "2×2 Max Pooling" presents a max-pooling layer with kernel size 2 and stride 2, "512-FC" is a fully connected layer

with 512 output neurons, and L2-SVM indicates the square hinge loss is used as loss function (L in Eq. 4) because L2-SVM has achieved better results than Softmax on several classification dataset [37,38]. For fairness of the comparison, the same settings are adopted based on MNIST dataset as used in BinaryConnect [22] algorithm. We use the last 10000 samples of training set for validation without retraining on the validation set and do not use any data-augmentation or unsupervised pretraining. Except for our Binary-L2 regularization, no other regularizations such as Dropout or L2 regularization are used. For detailed settings (e.g., batch size, learning rate), please refer to our Caffe [33] configuration files[2]. The whole Binarized-Weight-Networks are trained from scratch and the test accuracy is reported in Table 1.

Table 1 shows that the proposed binarized method improves the accuracy by more than 1.5% compared to the recent ternary or binarized approaches, and is slightly better than corresponding full precision network. Meanwhile, our simple 4-layers binarized weight network also achieve competitive performance compared with some complex full precision networks.

Table 1. Test set accuracy for MNIST of various methods.

Method	Test accuracy (%)
Our binarized-weight-network	**99.52**
Original full precision network	**99.48**
Binary or ternary CNN models	
Ternary weight network [36]	99.35
Binary precision weight network [36]	99.05
Binary MLP models	
BinaryConnect (det.) [22]	98.71
Binarized Neural Network [23]	99.04
Full Precision MLP or CNN Models	
Maxout Networks (MLP) [39]	99.06
Deep L2-SVM (MLP) [38]	99.13
Network In Network (CNN) + Dropout [10]	99.53
Conv.maxout (CNN) + Dropout [39]	99.55
Deeply-supervised-net (CNN) + Dropout [37]	99.61

3.2 CIFAR-10

The CIFAR-10 [40] dataset consists of 32×32 colour images in 10 classes including 50000 training images and 10000 test images. Generally speaking, the dataset is preprocessed by the global contrast normalization and ZCA whitening as was used in [10,22,37,39]. We adopt the same network architecture (VGG-like in

[2] https://github.com/xugithub1/caffe-Binarized-Weight-Networks.

Fig. 3 right) as other binarization methods [22,23,36], where the 9-layers architecture is more simple than standard VGG-Net [2] and "×2" denotes the same two layers. To compare with previous state-of-the-art works, we also evaluate our method on this dataset with data augmentation as used in [4,37]. In the training phase, zero-padding 4 pixels on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. At test time, we only use the single view of original 32×32 test image.

A comparison of our method with previous binarized or full precision approaches is shown in Table 2, and Fig. 4 exhibits test set error rate curves of several schemes. Experimental results[3] show that our binarized weight network not only obtain the new state-of-the-art accuracy compared with other binarized or ternary weight networks, but also surpass most of full precision networks including ResNet-56 structure. It indicates that the extreme low-precision networks do not destroy original network performance. On the contrary, network binarization and Binary-L2 techniques act as regularizer for improving generalization capability of original networks to some extent.

Table 2. Test set accuracy for CIFAR-10 of various methods.

Method	Test accuracy (%)
No data augmentation	
Our binarized-weight-network	**90.90**
Original full precision network	**89.66**
BinaryConnect (det.) [22]	90.10
Binary-weight-network [24]	90.12
Binarized neural network [23]	89.85
Maxout networks [39]	88.32
Network in network [10]	89.59
Deeply-supervised-net [37]	90.31
With data augmentation	
Our binarized-weight-network	**93.30**
Original full precision network	**92.89**
BinaryConnect (det.)	**91.82**
Ternary weight network [36]	92.56
Binary precision weight network [36]	90.18
Conv.maxout + dropout [39]	90.62
Network in network + dropout [10]	91.19
Deeply-supervised-net + dropout [37]	92.03
ResNet-56 [4]	93.03

[3] Boldface is our own experimental results.

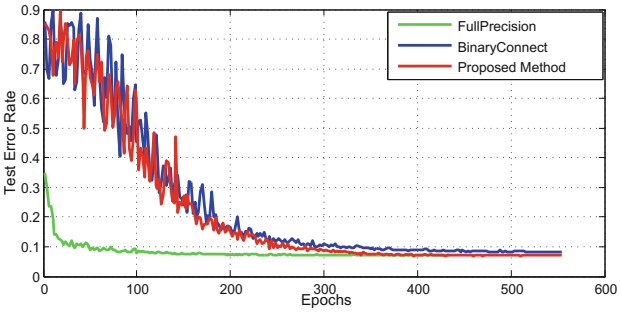


Fig. 4. Test set accuracy curves on CIFAR-10 dataset.

3.3 Compression Rates

The real compression rates of our binarized weight network are described in Table 3. We pack 32 binary-valued weight units into a single fixed32 ProtoBuffer unit [33], and other parameters such as biases and special parameters of Batch Normalization layers are still stored in float values. Finally, the proposed binarization scheme compress the float-valued similar LeNet5 (2.23 MB) and VGG-9Layer (53.5 MB) to binarized weight models whose storage requirement are 84.7 KB and 1.74 MB, respectively. The compression rates are 26.96× and 30.75×, respectively.

Table 3. The compression rates (Comp. rate) of Binarized-Weight-Networks on MNIST and CIFAR-10 datasets. 'N/Y' denotes without/with data augmentation.

Dataset	Model (full/binary)	Accuracy ('N/Y'/%)	Storage	Comp. rate
MNIST	Full	99.48	2.23 MB	1.00×
	Binary	99.52	84.7 KB	26.96×
CIFAR-10	Full	89.66/92.89	53.5 MB	1.00×
	Binary	90.90/93.30	1.74 MB	30.75×

4 Conclusion

In this paper, we present a simple yet effective algorithm for learning binarized weight deep neural network by introducing margin-aware weight constraint. The algorithm results in memory saving 32× (theoretically) compared with single-float precision models and make it possible to deploy deep neural networks on low-power devices or dedicated Deep Learning hardware. While training by back propagation algorithm and gradient decent optimization, the introduced margin-aware weight constraint brings obvious improvement of the performance with only negligible computational cost. Experimental results show that the proposed

algorithm outperforms existing binarization schemes, and achieves better performance than corresponding full precision networks. The proposed binarization scheme is also robust to different network architectures on several classical image classification datasets.

Acknowledgment. This work has been supported by the National Natural Science Foundation of China (Grant No. 61633021).

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.: ImageNet classification with deep convolutional neural networks. In: NIPS (2012)
2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Rabinovich, A.: Going deeper with convolutions. In: CVPR (2015)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
5. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: NIPS (2015)
6. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., Berg, A.: SSD: single shot MultiBox detector. In: ECCV (2016)
7. Kim, K.H., Cheon, Y., Hong, S., Roh, B., Park, M.: PVANET: deep but lightweight neural networks for real-time object detection. arXiv:1608.08021 (2016)
8. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR (2015)
9. Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.: Semantic image segmentation with deep convolutional nets and fully connected CRFs. In: ICLR (2015)
10. Lin, M., Chen, Q., Yan, S.: Network in network. In: ICLR (2014)
11. Iandola, F., Moskewicz, M., Ashraf, K., Han, S., Dally, W., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv:1602.07360 (2016)
12. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N.: Predicting parameters in deep learning. In: NIPS (2013)
13. Denton, E., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: NIPS (2014)
14. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: BMVC (2014)
15. Zhang, X., Zou, J., Ming, X., He, K., Sun, J.: Efficient and accurate approximations of nonlinear convolutional networks. In: CVPR (2015)
16. Zhou, H., Alvarez, J.M., Porikli, F.: Less is more: towards compact CNNs. In: ECCV (2016)
17. Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing neural networks with the hashing trick. In: ICML (2015)
18. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: NIPS (2015)
19. Han, S., Mao, H., Dally, W.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: ICLR (2016)

20. Vanhoucke, V., Senior, A., Mao, M.: Improving the speed of neural networks on CPUs. In: Proceedings of Deep Learning and Unsupervised Feature Learning NIPS Workshop (2011)
21. Soudry, D., Hubara, I., Meir, R.: Expectation Backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: NIPS (2014)
22. Courbariaux, M., Bengio, Y.: BinaryConnect: training deep neural networks with binary weights during propagations. In: NIPS (2015)
23. Courbariaux, M., Hubara, I., Soudry, D., ElYaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or –1. arXiv:1602.02830 (2016)
24. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: ECCV (2016)
25. Tang, W., Hua, G., Wang, L.: How to train a compact binary neural network with high accuracy? In: AAAI (2017)
26. Courbariaux, M., Bengio, Y., David, J.P.: Training deep neural networks with low precision multiplications. arXiv:1412.7024 (2014)
27. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
28. Wan, L., Zeiler, M., Zhang, S., LeCun, Y., Fergus, R.: Regularization of neural networks using DropConnect. In: ICML (2013)
29. Liong, V.E., Lu, J., Wang, G., Moulin, P., Zhou, J.: Deep hashing for compact binary codes learning. In: CVPR (2015)
30. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv:1412.6980 (2014)
31. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
32. Nair, V., Hinton, G.: Rectified linear units improve restricted Boltzmann machines. In: ICML (2010)
33. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the ACM International Conference on Multimedia (2014)
34. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS (2010)
35. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE (1998)
36. Li, F., Liu, B.: Ternary weight networks. arXiv:1605.04711 (2016)
37. Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: AISTATS (2015)
38. Tang, Y.: Deep learning using linear support vector machines. In: Workshop on Challenges in Representation Learning, ICML (2013)
39. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A.C., Bengio, Y.: Maxout networks. In: ICML (2013)
40. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, University of Toronto (2009)