Ternary weight networks

Fengfu Li and Bo Zhang
Institute of Applied Math., AMSS, CAS
Beijing, China
lifengfu12@mails.ucas.ac.cn;
b.zhang@amt.ac.cn

Bin Liu
Moshanghua Tech Co., Ltd.
Beijing, China
liubin@dress-plus.com

Abstract

We introduce ternary weight networks (TWNs) - neural networks with weights constrained to +1, 0 and -1. The Euclidian distance between full (float or double) precision weights and the ternary weights along with a scaling factor is minimized. Besides, a threshold-based ternary function is optimized to get an approximated solution which can be fast and easily computed. TWNs have stronger expressive abilities than recently proposed binary precision counterparts and are more effective than the latter. Meanwhile, TWNs achieve up to $16\times$ or $32\times$ model compression rate and need fewer multiplications compared with the full precision counterparts. Benchmarks on MNIST, CIFAR-10, and large scale ImageNet datasets show that the performance of TWNs is only slightly worse than the full precision counterparts but outperforms the analogous binary precision counterparts a lot.

1 Introduction

Deep neural networks (DNN) have made significant improvements in lots of computer vision tasks such as object recognition [4, 10, 17, 18] and object detection [14, 16]. This motivates interests to deploy the state-of-the-art DNN models to real world applications like smart phones or embedded devices. However, these models often need considerable storage and computational power [15], and can easily overburden the limited storage, battery power, and computer capabilities of the small embedded devices. As a result, it remains a challenge for the deployment.

1.1 Binary weight networks and model compression

To address the storage and computational issues [2, 3], methods that seek to binarize weights or activations in DNN models have been proposed. BinaryConnect [1] uses a single sign function to binarize the weights. Binary Weight Networks [15] adopts the same binarization function but adds an extra scaling factor. The extensions of the previous methods are BinaryNet [5] and XNOR-Net [15] where both weights and activations are binary-valued. These models eliminate most of the multiplications in the forward and backward propagations, and thus own the potential of gaining significant benefits with specialized deep learning (DL) hardware by replacing many multiply-accumulate operations by simple accumulation [13]. Besides, binary weight networks achieve up to $32\times$ or $64\times$ model compression rate.

Despite the binary techniques, some other compression methods focus on identifying models with few parameters while preserving accuracy by compressing existing state-of-the-art DNN models in a lossy way. SqueezeNet [7] is such a model that has 50x fewer parameters than AlexNet [10] but maintains AlexNet-level accuracy on ImageNet. Deep Compression [3] is another most recently proposed method that uses pruning, trained quantization and huffman coding for compressing neural networks. It reduced the storage requirement of AlexNet and VGG-16 [17] by $35\times$ and $49\times$, respectively, without loss of accuracy.

2 Ternary weight networks

We address the limited storage and limited computational resources issues by introducing ternary weight networks (TWNs), which constrain the weights to be ternary-valued: +1, 0 and -1. TWNs seek to make a balance between the full precision weight networks (FPWNs) counterparts and the binary precision weight networks (BPWNs) counterparts. The detailed features are listed as follows.

Expressive ability In most recent network architectures such as VGG [17], GoogLeNet [18] and residual networks [4], a most commonly used convolutional filter is of size 3×3. With binary precision, there is only $2^{3×3} = 512$ templates. However, a ternary filter with the same size owns $3^{3×3} = 19683$ templates, which gains 38x more stronger expressive abilities than the binary counterpart.

Model compression In TWNs, 2-bit storage requirement is needed for a unit of weight. Thus, TWNs achieve up to $16\times$ or $32\times$ model compression rate compared with the float (32-bit) or double (64-bit) precision counterparts. Take VGG-19 [17] as an example, float version of the model needs $\sim$500M storage requirement, which can be reduced to $\sim$32M with ternary precision. Thus, although the compression rate of TWNs is $2\times$ less than that of BPWNs, it is fair enough for compressing most of the existing state-of-the-art DNN models.

Computational requirement Compared with the BPWNs, TWNs own an extra 0 state. However, the 0 terms need not be accumulated for any multiple operations. Thus, the multiply-accumulate operations in TWNs keep unchanged compared with binary precision counterparts. As a result, it is also hardware-friendly for training large-scale networks with specialized DL hardware.

In the following parts, we will give detailed descriptions about the ternary weight networks problem and an approximated but efficient solution. After that, a simple training algorithm with error back-propagation is introduced and the run time usage is described at last.

2.1 Problem formulation

To make the ternary weight networks perform well, we seek to minimize the Euclidian distance between the full precision weights $\mathbf{W}$ and the ternary-valued weights $\mathbf{W}^t$ along with a nonnegative scaling factor α [15]. The optimization problem is formulated as follows,

$$\begin{cases} \alpha^*, \mathbf{W}^{t*} = & \arg\min_{\alpha, \mathbf{W}^t} J(\alpha, \mathbf{W}^t) = ||\mathbf{W} - \alpha\mathbf{W}^t||_2^2 \\ \text{s.t.} & \alpha \geq 0, \ \mathbf{W}_i^t \in \{-1, 0, 1\}, \ i = 1, 2, \ldots, n. \end{cases} \quad (1)$$

Here n is the size of the filter. With the approximation $\mathbf{W} \approx \alpha\mathbf{W}^t$, a basic block of forward propagation in ternary weight networks is as follows,

$$\begin{cases} \mathbf{Z} = & \mathbf{X} * \mathbf{W} \approx \mathbf{X} * (\alpha\mathbf{W}^t) = (\alpha\mathbf{X}) \oplus \mathbf{W}^t \\ \mathbf{X}^{\text{next}} = & g(\mathbf{Z}) \end{cases} \quad (2)$$

Here $\mathbf{X}$ is the input of the block; $*$ is a convolutional operation or an inner product; g is a nonlinear activation function; $\oplus$ indicates an inner product or a convolution operation without any multiplication; $\mathbf{X}^{\text{next}}$ is the output of the block, and can serve as an input of the next block.

2.2 Approximated solution with threshold-based ternary function

One way to solve the optimization problem (1) is to expand the cost function $J(\alpha, \mathbf{W}^t)$ and take the derivative w.r.t. α and $\mathbf{W}_i^t$s respectively. However, this would get interdependent α^* and $\mathbf{W}_i^{t*}$. Thus, there is no deterministic solution in this way [6]. To overcome this, we try to find an approximated optimal solution with a threshold-based ternary function,

$$\mathbf{W}_i^t = f_t(\mathbf{W}_i|\Delta) = \begin{cases} +1, & \text{if } \mathbf{W}_i > \Delta \\ 0, & \text{if } |\mathbf{W}_i| \leq \Delta \\ -1, & \text{if } \mathbf{W}_i < -\Delta \end{cases} \quad (3)$$

Here Δ is an positive threshold parameter. With (3), the original problem can be transformed to

$$\alpha^*, \Delta^* = \arg\min_{\alpha \geq 0, \Delta > 0} \left(|\mathbf{I}_\Delta|\alpha^2 - 2(\sum_{i \in \mathbf{I}_\Delta} |\mathbf{W}_i|)\alpha + c_\Delta \right) \quad (4)$$

where $\mathbf{I}_\Delta = \{i \, | \, |\mathrm{W}_i| > \Delta\}$ and $|\mathbf{I}_\Delta|$ denotes the number of elements in $\mathbf{I}_\Delta$; $c_\Delta = \sum_{i \in \mathbf{I}_\Delta^c} \mathrm{W}_i^2$ is a α-independent constant. Thus, for any given Δ, the optimal α can be computed as follows,

$$\alpha_\Delta^* = \frac{1}{|\mathbf{I}_\Delta|} \sum_{i \in \mathbf{I}_\Delta} |\mathrm{W}_i|. \tag{5}$$

By substituting α_Δ^* into (4), we get a Δ-dependent equation, which can be simplified as follows,

$$\Delta^* = \arg\max_{\Delta > 0} \frac{1}{|\mathbf{I}_\Delta|} \big(\sum_{i \in \mathbf{I}_\Delta} |\mathrm{W}_i| \big)^2 \tag{6}$$

Problem (6) has no straightforward solutions. Though discrete optimization can be made to solve the problem (due to states of W_is are finite), it can be time consuming. Instead, we make a single assumption that W_is are generated from uniform or normal distribution. In case of W_is are uniformly distributed in $[-a, a]$ and Δ lies in $(0, a]$, the approximated Δ^* is $\frac{1}{3}a$, which equals to $\frac{2}{3}\mathrm{E}(|\mathbf{W}|)$. When W_is are generated from normal distributions $N(0, \sigma^2)$, the approximated Δ^* is 0.6σ which equals to $0.75 \cdot \mathrm{E}(|\mathbf{W}|)$. Thus, we can use a rule of thumb that $\Delta^* \approx 0.7 \cdot \mathrm{E}(|\mathbf{W}|) \approx \frac{0.7}{n} \sum_{i=1}^{n} |\mathrm{W}_i|$ for fast and easy computation.

2.3 Training with stochastic gradient descent method

We use stochastic gradient descent (SGD) method to train TWNs. As in Courbariaux et al. [1] and Rastegari et al. [15], ternary-valued weights are used during the forward and backward propagations but not during the parameters update. In addition, two useful tricks, Batch Normalization (BN) [8] and learning rate scaling, are adopted. We also use momentum for acceleration.

2.4 Model compression and run time usage

In the forward propagation, the scaling factor α could be transformed to the inputs according to (2). Thus, we only need to keep the ternary-valued weights and the scaling factors for deployment. This would results up to $16\times$ or $32\times$ model compression rate for run time usage compared with the float or double precision counterparts, respectively.

3 Experiments

In this section, we benchmark TWNs with BPWNs and FPWNs on the MNIST, CIFAR-10 and ImageNet (2012) dataset [1]. For fair comparison, we set the following terms to be identical: network architecture, regularization method (L2 weight decay), learning rate scaling procedure (multi-step) and optimization method (SGD with momentum). BPWNs use sign function to binarize the weights and FPWNs use float-valued weights. See Table. 1 for detailed configurations.

Table 1: Network architecture and parameters setting for different datasets.

	MNIST	CIFAR-10	ImageNet
network architecture	LeNet-5	VGG-7	ResNet-18(B)
weight decay	1e-4	1e-4	1e-4
mini-batch size of BN	50	100	64 $(\times 4)$[2]
initial learning rate	0.01	0.1	0.1
learning rate decay[3] epochs	15, 25	80, 120	30, 40, 50
momentum	0.9	0.9	0.9

MNIST The LeNet-5 [11] architecture we used is "32-C5 + MP2 + 64-C5 + MP2 + 512 FC + SVM". It starts with a convolutional block that owns 32 filters with size 5×5. A max-pooling layer is

---

[1] The implementation codes will be available at https://github.com/fengfu-chris/caffe-twns.
[2] We use 4 GPUs to speed up the training in Caffe [9].
[3] Learning rate is divided by 10 at these epochs.

Table 2: Validation accuracies (%). Results on ImageNet are with ResNet-18 / ResNet-18B.

	MNIST	CIFAR-10	ImageNet (top-1)	ImageNet (top-5)
TWNs	**99.35**	**92.56**	**61.8 / 65.3**	**84.2 / 86.2**
BPWNs	99.05	90.18	57.5 / 61.6	81.2 / 83.9
FPWNs	99.41	92.88	65.4 / 67.6	86.76 / 88.0
BinaryConnect	98.82	91.73	-	-
Binarized Neural Networks	88.6	89.85	-	-
Binary Weight Networks	-	-	60.8	83.0
XNOR-Net	-	-	51.2	73.2

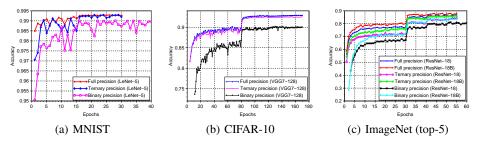(a) MNIST   (b) CIFAR-10   (c) ImageNet (top-5)

Figure 1: Validation accuracy curves.

followed with stride 2. The "FC" is a fully connect block with 512 nodes. The top layer is a SVM classifier with 10 labels. Finally, hinge loss is minimized with SGD.

CIFAR-10 We define a VGG inspired architecture, denoted as VGG-7, by "2×(128-C3) + MP2 + 2×(256-C3) + MP2 + 2×(512-C3) + MP2 + 1024-FC + Softmax". Compared with the architecture adopted in Courbariaux et al. [1], we ignore the last fully connection layer. We follow the data augmentation in He et al. [4] and Lee et al. [12] for training: 4 pixels are padded on each side, and a 32×32 crop is randomly sampled from the padded image or its horizontal flip. At testing time, we only evaluate the single view of the original 32×32 image.

ImageNet We adopt the recent proposed ResNet-18 architecture [4]. Besides, to address the issue of model size, we also benchmark another enlarged counterpart whose number of filters in each block is 1.5× of the original one. This enlarged model is named as ResNet-18B. In each training iteration, images are randomly cropped with 224×224 size. We do not use any resize tricks [15] or any color augmentation, yet.

Table. 2 summarizes the overall benchmark results with the previous settings. On the small scale datasets (MNIST and CIFAR-10), TWNs achieve state-of-the-art performance as FPWNs, while beat BPWNs a lot. On the large scale ImageNet dataset, BPWNs and TWNs both get poorer performance than FPWNs. However, the accuracy gap between TWNs and FPWNs is smaller than the gap between BPWNs and TWNs. Thus, TWNs beat BPWNs again. In addition, as the model size enlarges, the performance gap between TWNs (or BPWNs) and FPWNs has been reduced. This indicates low precision networks gain more merits from larger models than the full precision counterparts.

Fig. 1 shows the validation accuracy curves on these datasets. As shown in the figure, BPWNs converge slowly and vibrate more seriously than TWNs and FPWNs. However, TWNs converge almost as fast and stably as FPWNs.

4   Conclusion

We have proposed a ternary weight networks optimization problem and given an approximated solution with a simple but accurate ternary function. The proposed TWNs find a balance between the high accuracy of TWNs and the high model compression rate as well as potentially low computational requirements of BPWNs. Benchmarks demonstrate the superior performance of the proposed method.

References

[1] 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*, pages 3123–3131, 2015.

[2] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *arXiv preprint arXiv:1603.08270*, 2016.

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

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

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

[6] K. Hwang and W. Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, 2014.

[7] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 562–570, 2015.

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

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. Ssd: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

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

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.