2017 International Conference on Tools with Artificial Intelligence

Asymmetric Ternary Networks

Jie Ding
University of Science and Technology of China
Department of Computer Science and Technology
SuZhou, China
dj1993@mail.ustc.edu.cn

JunMin Wu
University of Science and Technology of China
Department of Computer Science and Technology
SuZhou, China
jmwu@ustc.edu.cn

Huan Wu
University of Science and Technology of China
Department of Computer Science and Technology
SuZhou, China
wuhuan00@mail.ustc.edu.cn

***Abstract*—Deep Neural Networks (DNNs) are widely used in a variety of machine learning tasks currently, especially in speech recognition and image classification. However, the huge demand for memory and computational power makes DNNs cannot be deployed on embedded devices efficiently. In this paper, we propose asymmetric ternary networks (ATNs) – neural networks with weights constrained to ternary values ($-\alpha_1, 0, +\alpha_2$), which can reduce the DNN models size by about 16× compared with 32-bits full precision models. Scaling factors $\{\alpha_1, \alpha_2\}$ are used to reduce the quantization loss between ternary weights and full precision weights. We compare ATNs with recently proposed ternary weight networks (TWNs) and full precision networks on CIFAR-10 and ImageNet datasets. The results show that our ATN models outperform full precision models of VGG13, VGG16 by 0.11%, 0.33% respectively on CIFAR-10. On ImageNet, our model outperforms TWN AlexNet model by 2.25% of Top-1 accuracy and has only 0.63% accuracy degradation over the full-precision counterpart.**

Keywords—Deep Neural Networks; Asymmetric Ternary Networks; Model Compression; Embedded efficient Neural Networks

I. INTRODUCTION

Deep Neural Networks (DNNs) have substantially made improvements in a wide of range of machine learning tasks, including but not limited to the image classification [1, 2] and object detection [3, 4]. However, a typical DNN always has millions of parameters which make it difficult to be deployed on embedded devices with limited storage and computational power. As a result, it remains a great challenge to deploy deep DNNs on embedded devices.

Many approaches have been proposed to solve this problem. The most common method is to compress a full-trained model directly. [5] introduced vector quantization techniques to compress the DNNs' weights, by replacing the weights in full connected layers with respective floating-point centers obtained from k-means clustering. HashedNets [6] reduced the model size by using a hash function to push the weights into corresponding buckets and force them to share the same value. However, they both concentrated on full connected layers only. Another effective method is using lower precision weights, which can not only reduce the size of networks, but also speed up the execution. [7] proposed that using SIMD instructions with 8-bits fixed point implementation can improve the performance of computing during inference, yielding 3× speed-up over floating-point baseline. [8] trained deep neural networks with low precision multipliers and high precision accumulators. [9] introduced an approach to eliminate the need of float-point multiplication by converting multiplication into binary shift.

Nowadays, 1-bit and 2-bits networks have aroused the interest of the researchers. BinaryConnect [10] uses a single sign function to binarize the weights to {+1, -1}, which obtained 32× compression rate compared with full precision counterparts. Binary Weight Networks [12] adopts the same binarization function but add an extra scaling factor to improve the model capacity of networks. The extensions of the previous methods are BinaryNet [11] and XNOR-Net [12] where both weights and activations are binary-valued, which outperformed previous binary methods by large margins on ImageNet dataset. Furthermore, [13] introduced ternary weight networks (TWNs) with the weights constrained to $\{-\alpha, 0, +\alpha\}$ to find a balance between high model compression rate and high accuracy, which achieved better performance compared with previous quantization methods due to the increased weight precision. However, the same scaling factors for positive and negative weights have limited the expression ability of the ternary weight networks.

In this paper, we propose Asymmetric Ternary Networks (ATNs) to explore higher model capacity with the ternary weights. We constrain weights to three values $\{-W_l^n, 0, +W_l^p\}$ for each layer, which can also be encoded with two bits. Compared with TWNs quantization method, our ATNs eliminate the limitation between positive and negative weights to achieve higher model capacity and better performance. Furthermore, because the scaling factors $\{W_l^n, W_l^p\}$ are fixed during inference, multiplication

2375-0197/17/31.00 ©2017 IEEE
DOI 10.1109/ICTAI.2017.00021

operations can be replaced with simple addition and subtraction operations by computing the scaling factors on activate functions in advance on specialized hardware, which can greatly speed up the network propagations.

II. Asymmetric Ternary Networks

In this section, we give a more detailed view of asymmetric ternary networks (ATNs), considering how to obtain ternary values from full precision weights and train deep neural networks with ternary weights.

A. Ternary Values

Our asymmetric ternary networks (ATNs) constrain the full precision weights W_l to ternary weights W_l^t with values belong to $\{-W_l^n, 0, +W_l^p\}$. We use threshold-based ternary function to quantize the full precision weights, which is shown in (1).

$$W_{l\,i}^t = \begin{cases} +W_l^p & W_{l\,i} > \Delta_l^p \\ 0 & -\Delta_l^n < W_{l\,i} < \Delta_l^p \\ -W_l^n & W_{l\,i} < -\Delta_l^n \end{cases} \quad (1)$$

Here $\{\Delta_l^p, \Delta_l^n\}$ are the thresholds used to quantize the positive weights and negative weights respectively, and $\{W_l^p, W_l^n\}$ are the scaling factors used to reduce the loss between ternary weights and real-valued weights.

In our ternary function, we propose four independent factors $\{W_l^p, W_l^n, \Delta_l^p, \Delta_l^n\}$ to quantize the full precision weights. In order to make the asymmetric ternary networks (ATNs) perform well, we seek to minimize the Euclidian distance between the full precision weights W_l and ternary weights W_l^t [14]. The optimization problem is formulated as (2).

$$\begin{cases} W_l^{p*},\ W_l^{n*} = \arg\min\ J(W_l^p, W_l^n) = \arg min\, ||W_l - W_l^t||_2^2 \\ \text{s.t.}\ W_l^p, W_l^n > 0,\ W_{l\,i}^t \in \{-W_l^n, 0, +W_l^p\}, \mathrm{i} = 1, 2, \ldots, \mathrm{n}. \end{cases} \quad (2)$$

Substitute the ternary function (1) into the formula (2), the expression can be transformed to:

$$\begin{aligned} \mathrm{J}(W_l^p, W_l^n) &= \left\|W_l - W_l^t\right\|_2^2 \\ &= \textstyle\sum_{i|W_{l\,i}>\Delta_l^p} \left||W_{l\,i}| - W_l^p\right|^2 + \sum_{i|\,W_{l\,i}<-\Delta_l^n} \left||W_{l\,i}| - W_l^n\right|^2 \\ &\quad + \textstyle\sum_{i|-\Delta_l^n<W_{l\,i}<+\Delta_l^p} |W_{l\,i}|^2 \\ &= \left|I_{\Delta_l^p}\right| * (W_l^p)^2 - 2 * \left(\textstyle\sum_{i\in I_{\Delta_l^p}} |W_{l\,i}|\right) * (W_l^p) \\ &\quad + \left|I_{\Delta_l^n}\right| * (W_l^n)^2 - 2 * \left(\textstyle\sum_{i\in I_{\Delta_l^n}} |W_{l\,i}|\right) * (W_l^n) + \mathrm{C} \end{aligned} \quad (3)$$

Where $I_{\Delta_l^p} = \{i \mid W_{l\,i} > \Delta_l^p\}$, $I_{\Delta_l^n} = \{i \mid W_{l\,i} < -\Delta_l^n\}$ and $\left|I_{\Delta_l^*}\right|$ denotes the number of elements in $I_{\Delta_l^*}$. Δ_l^p and Δ_l^n are independent together. $\mathrm{C} = \sum_{i=1}^{n} |W_{l\,i}|^2$ is a $\{W_l^p, W_l^n\}$-independent constant.

Therefore, our scaling factors $\{W_l^p, W_l^n\}$ can be simplified to:

$$\begin{aligned} W_l^{p*},\ W_l^{n*} &= \arg\min\ J(W_l^p, W_l^n) \\ &= \arg min(\left|I_{\Delta_l^p}\right| * (W_l^p)^2 - 2 * \left(\textstyle\sum_{i\in I_{\Delta_l^p}} |W_{l\,i}|\right) * (W_l^p)) \\ &\quad + \arg min(\left|I_{\Delta_l^n}\right| * (W_l^n)^2 - 2 * \left(\textstyle\sum_{i\in I_{\Delta_l^n}} |W_{l\,i}|\right) * (W_l^n)) \end{aligned} \quad (4)$$

For any given thresholds $\{\Delta_l^p, \Delta_l^n\}$, the optimal scaling factors $\{W_l^p, W_l^n\}$ can be computed as follows:

$$\begin{aligned} W_l^{p*} &= \frac{1}{\left|I_{\Delta_l^p}\right|} * \textstyle\sum_{i\in I_{\Delta_l^p}} |W_{l\,i}| \\ W_l^{n*} &= \frac{1}{\left|I_{\Delta_l^n}\right|} * \textstyle\sum_{i\in I_{\Delta_l^n}} |W_{l\,i}| \end{aligned} \quad (5)$$

By substituting W_l^{p*}, W_l^{n*} into (4), we get $\{\Delta_l^p, \Delta_l^n\}$-dependent equation, which can be simplified as:

$$\begin{aligned} \Delta_l^{p*} &= \arg max\ \frac{1}{\left|I_{\Delta_l^p}\right|} * \left(\textstyle\sum_{i\in\Delta_l^p} |W_{l\,i}|\right)^2 \\ \Delta_l^{n*} &= \arg max\ \frac{1}{\left|I_{\Delta_l^n}\right|} * \left(\textstyle\sum_{i\in\Delta_l^n} |W_{l\,i}|\right)^2 \end{aligned} \quad (6)$$

Here $\{\Delta_l^p, \Delta_l^n\}$ are both positive values. Because problem (6) has no straightforward solutions as [14], we use approximate values for fast computations. We make a single assumption that $W_l's$ values are generated from uniform distribution or normal distribution, by using the same trick as described in [13], but with different values for positive and negative weights, we obtain the thresholds as follows.

$$\begin{aligned} \Delta_l^{p*} &\approx 0.7 * \frac{1}{|I^p|} * \textstyle\sum_{i\in I^p} |W_{l\,i}| \\ \Delta_l^{n*} &\approx 0.7 * \frac{1}{|I^n|} * \textstyle\sum_{i\in I^n} |W_{l\,i}| \end{aligned} \quad (7)$$

Where $I^p = \{i\ |W_{l\,i} \geq 0 \mid \mathrm{i} = 1,2 \ldots \mathrm{n}\}$, $I^n = \{i|W_{l\,i} < 0 \mid i = 1,2, \ldots n\}$. Finally, by substituting (5) (7) into (1), we can get ternary weights easily from the full precision weights.

B. Train Asymmetric Ternary Networks

In this section, we give a detailed view of the training algorithm for Asymmetric Ternary Networks with Stochastic Gradient Descent (SGD), which is shown in Algorithm 1.

Our training method is similar to normal training process except for the ternary weights are used in forward and backward propagations (step1 and step 2), which is same as [10]. And reserved full precision weights are used to update parameters to obtain the tiny changes in each iteration (step 3).

During training, some useful tricks are used to speed up training process and improve the inference accuracy. Batch Normalization [15] not only accelerates training by reducing internal covariate shift, but also reduces the impact of weight scales. Also, momentum and learning rate scaling [16] are effective methods to optimize network training.

C. Inference

During inference, only ternary weights are needed, by storing the weights with 2-bits values, we can reduce the model size by about 16×. Furthermore, due to the scaling factors $\{W_l^p, W_l^n\}$ are fixed after training, pre-computing the scaling factors on activations is an effective way to speed up the forward propagation on custom hardware, for lots of multiplications can be replaced with simple addition or subtraction operations.

Algorithm 1: SGD training for ATNs

Input: Learned full-precision weights $\boldsymbol{W_l}$ and $\boldsymbol{b_l}$ for layer l
layer l output: $\boldsymbol{a_l}$, **C** is the loss function of networks
Ternary ($\boldsymbol{W_l}$) means to quantize weights to ternary values. **L** is the number of layers

Begin

1. Forward propagation:

for l ← 1 to **L-1 do**

$$\boldsymbol{W_l^t} \leftarrow \text{Ternary}(\boldsymbol{W_l})$$

$$\boldsymbol{a_{l+1}} \leftarrow f(\boldsymbol{W_l^t} * \boldsymbol{a_l} + \boldsymbol{b_l})$$

// * means inner product or convolutional operation

end

2. Backward propagation:

for l ← **L-1** to 1 **do**

$$\frac{\partial C}{\partial a_l} \leftarrow \left((\boldsymbol{W_l^t})^{\boldsymbol{T}} * \frac{\partial C}{\partial a_{l+1}} \right) \circ f'$$

// ○ means element-wise product

$$\frac{\partial C}{\partial W_l} \leftarrow \frac{\partial C}{\partial a_{l+1}} * (\boldsymbol{a_l})^{\boldsymbol{T}}$$

$$\frac{\partial C}{\partial b_l} \leftarrow \frac{\partial C}{\partial a_{l+1}}$$

end

3. Parameter update:

for l ← 1 to **L-1 do**

$$\boldsymbol{W_l} \leftarrow \boldsymbol{W_l} - \eta * \frac{\partial C}{\partial W_l}$$

$$\boldsymbol{b_l} \leftarrow \boldsymbol{b_l} - \eta * \frac{\partial C}{\partial b_l}$$

end

End

III. EXPERIMENTS

In this section, we compare the performance of ATNs with ternary weight networks (TWNs) and full precision networks on CIFAR-10 and ImageNet benchmark datasets. For fair comparison, we set the same hyper parameters to train the networks, such as networks architecture, regularization method (L2 weight decay), learning rate scaling and optimized method (SGD) with momentum to accelerate convergence. In addition, experiments on CIFAR-10 are repeated 4 times to obtain the average results, reducing the effect of random initialization and data augmentation. We implement our experiments on Caffe [17] framework.

A. Cifar-10

CIFAR-10 is a benchmark image classification dataset, consisting of a training set with 50 thousand 32×32 color images and a test set with 10 thousand images. We train VGG16 [18] network structure on CIFAR-10 with some data-augmentation operations. We pad 2 pixels in each side of images and randomly crop 32×32 size from padded images during training, while original 32×32 images are used during testing.

In order to accelerate the convergence of the networks, we use a full-trained full precision VGG16 model to initialize our ternary model. SGD method is used to update parameters with momentum equals 0.9. Mini-batch size is set to 100. Learning rate is initialized to 0.01 and scaled by 0.1 at epoch 50, 100, 150 and 200. A L2-normalized weight decay of 0.0005 is used as regularization.

We compare our model with the full precision model and a ternary weights model (TWNs). First we train a full precision VGG16 model on CIFAR-10 as a baseline (blue line in Fig. 1), and then fine-tune this baseline with ternary weight networks (TWNs) quantization method (red line) and asymmetric ternary networks (ATNs) quantization method (green line). The result (Fig. 1) shows that our ATNs model outperforms TWNs model and full precision model by 0.41%, 0.33% respectively.

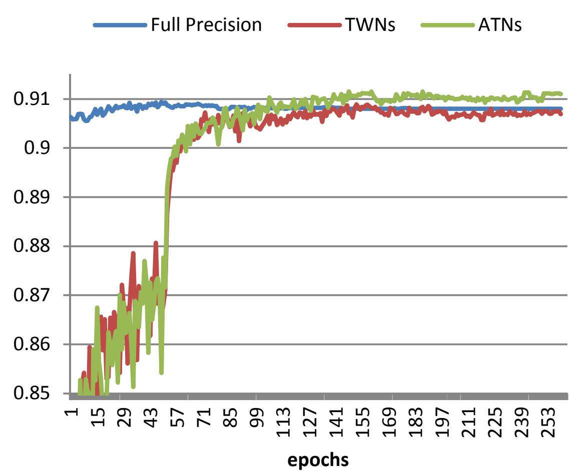

Figure 1. Test Accuracy of VGG16 networks on CIFAR-10

Furthermore, we expand our experiments to VGG13 and VGG10 networks which are obtained by removing the last 3 convolution layers, last 6 convolution layers from VGG16 respectively. All ternary models are fine-tuned from full precision models, the results are listed in Table I.

Our results (Table I) show that ATNs models improve the accuracy of VGG13 and VGG16 by 0.11% and 0.33% compared with the full precision networks. The deeper the model, the larger the improvement. We conjecture that the ternary weights have sufficient expressiveness in the depth networks and prevent over-fitting by the sparse weights like dropout [19].

TABLE I. TEST ACCURACY OF VGG ON CIFAR-10

Model	Full Precision	TWNs	ATNs	Improvements
VGG10	**90.55**	90.20	**90.35**	**-0.20**/0.15
VGG13	**90.66**	90.49	**90.77**	**0.11**/0.28
VGG16	**90.80**	90.72	**91.13**	**0.33**/0.41

B. ImageNet

ImageNet [20] is an image classification dataset with over 1.28 million training images and 50 thousand validation images. We use AlexNet [21] structure in our experiment with the full precision weights for the first convolution layer and the last full connected layer, other layer parameters are all quantized to ternary values. We also use the same data augmentation operations as described in III-A and add some image rotation operations. During training, images are resized to 256×256 and randomly cropped to 227×227 before input. Stochastic gradient descent (SGD) method is used to update parameters with momentum equals 0.9. Mini-batch size is set to 256. Learning rate is initialized to 0.0001 and scaled by 0.1 at epochs 50 and 60. A L2-normalized weight decay of 0.0005 is used as a regularization.

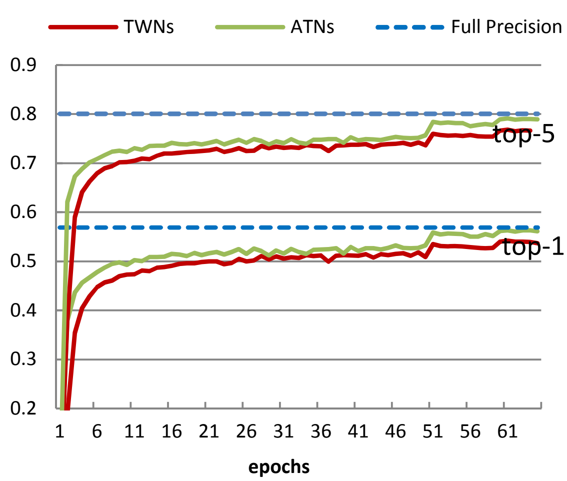

Figure 2. Validation accuracy of AlexNet on ImageNet

In order to accelerate the convergence of the ternary network, we download a full-trained AlexNet model from caffe model zoo as a full precision baseline, and fine tune the ternary models with this baseline. Our training curves are shown in Fig. 2, the result shows that our ATNs model (green line) reaches top-1 validation accuracy of 56.27% which has only 0.63% accuracy degradation over full precision counterpart, while TWNs model (red line) gets 54.02%. The complete results are listed in Table II.

TABLE II. VALIDATION ACCURACY OF ALEXNET ON IMAGENET

Accuracy	Full Precision	TWNs	ATNs
Top-1	56.9	54.02	56.27
Top-5	80.1	76.46	78.91

IV. CONCLUSION

In this paper, we propose an asymmetric ternary networks (ATNs) with weights constrained to $\{+W_l^p, 0, -W_l^n\}$ for each layer which can reduce the model size by about 16× compared with full precision weights. Also, we give a simple but accurate threshold function to quantize the full precision weights to ternary values. The proposed ATNs eliminate the limitation between positive and negative weights to achieve higher model capacity and better networks performance. Experiments show that our ATN models reach or even surpass the accuracy of full precision models on CIFAR-10 datasets and exceed the accuracy of ternary weights networks (TWNs) by 2.25% on ImageNet dataset. Future works will extend those results to other models and datasets, and explore deeper relationships between ternary values and networks output.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition., 2016.

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

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[4] R. Girshick, "Fast R-CNN," Proceedings of the IEEE International Conference on Computer Vision, 2015.

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

[6] W. Chen, W. James, T. Stephen, W. Kilian, and Y. Chen, "Compressing neural networks with the hashing trick," arXiv preprint arXiv:1504.04788, 2015.

[7] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," Proceedings of the Deep Learning and Unsupervised Feature Learning NIPS Workshop, Vol. 1, 2011.

[8] M. Courbariaux, Y. Bengio, and J. P. David, "Training deep neural networks with low precision multiplications," arXiv preprint arXiv:1412.7024, 2015.

[9] Z. Lin, M. Courbariaux, R. Memisevic, Y. Bengio, "Neural Networks with Few Multiplications," arXiv preprint arXiv:1510.03009, 2016.

[10] M. Courbariaux, Y. Bengio, and J. P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations,"

Proceedings of the Advances in Neural Information Processing Systems, 2015.

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

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

[13] F. Li, B. Zhang, and B. Liu, “Ternary weight networks,” arXiv preprint arXiv:1605.04711, 2016.

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

[15] S. Ioffe, and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” International Conference on Machine Learning, 2015.

[16] Y. Bengio, and Y. LeCun, “Scaling learning algorithms towards AI,” Large-scale kernel machines, 34(5), 2007.

[17] Y. Jia, “Caffe: An open source convolutional architecture for fast feature embedding,” Proceedings of the 22nd ACM international conference on Multimedia, 2014.

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

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” Journal of machine learning research, 15(1), 2014.

[20] J. Deng, W. Dong, J. Socher, L. Li, K. Li, and F. Li, “Imagenet: A large-scale hierarchical image database,” Proceedings of the Computer Vision and Pattern Recognition, 2009.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks,” Proceedings of the Advances in Neural Information Processing Systems, 2012