Training Ternary Neural Networks with Exact Proximal Operator

Penghang Yin *, Shuai Zhang, Jack Xin, and Yingyong Qi †

Abstract

In this paper, we propose a stochastic proximal gradient method to train ternary weight neural networks (TNN). The proposed method features weight ternarization via an exact formula of proximal operator. Our experiments show that our trained TNN are able to preserve the state-of-the-art performance on MNIST and CIFAR-10 benchmark datesets.

1 Introduction

Deep neural networks (DNN) [9, 8] have gained increasing popularity in machine learning due to their generally stronger performances over the other techniques. How to deploy DNN with tons of parameters on small devices of limited storage and computing power, however, still poses a challenge to the real world application. Much work has been done to address the storage and computational issues; e.g., see [4] and references therein.

Recently one promising idea has emerged regarding carefully performing binarization on the float or double precision weights of DNN assisted with only a few extra scaling factors [1, 2, 12], that is, the weights of BNN are restricted to $\{-\alpha, \alpha\}$ for some $\alpha > 0$. Binary weight neural networks (BNN) not only enables substantial memory savings, but also eliminates most of the multiplications during inference thanks to the distribute law. To trade off between accuracy and model size, ternary weight neural networks [10, 16] leverage an extra state 0 in the weights, in which both the model capacity and the weight sparsity is greatly enhanced. However, more efforts are needed to do the training well.

One key step in the training of a ternary net lie in the ternarization on float or double precision weights, i.e., how to properly scale the larger weights (in terms of magnitude) and meanwhile zero out the smaller weights, which is trickier than training a binary net. The recent work [10, 16] suggest to use a heuristic thresholding value to handle this issue. In this paper, we derive an exact formula for computing the ternarization that fits the full-precision weights the best; see section 2 for the technical details, where a stochastic proximal gradient algorithm is also proposed. Experiment results are provided in section 3.

2 Training ternary neural networks

To ternarize some real-valued weight W^r, we seek to minimize the Euclidean distance between W^r and some set $\mathcal{T}$ characterizing the ternarization on W^r. The ternarization is simply abstracted as the following optimization problem,

$$\mathbf{prox}_{\mathcal{T}}(W^r) := \arg\min_{W} \ \|W - W^r\|^2 \quad \text{subject to} \quad W \in \mathcal{T}. \tag{1}$$

This a problem of finding the proximal operator $\mathbf{prox}_{\mathcal{T}}$ [11] of the indicator function $\mathbf{1}_{\mathcal{T}}$ defined as

$$\mathbf{1}_{\mathcal{T}}(W) = \begin{cases} 0 & \text{if } W \in \mathcal{T}, \\ +\infty & \text{otherwise.} \end{cases}$$

*Department of Mathematics, University of California, Los Angeles, CA, USA. Email: yph@ucla.edu.

†Department of Mathematics, University of California, Irvine, CA, USA. Email: (szhang3, jack.xin, yqi)@uci.edu.

This work was partially supported by NSF grants DMS-1522383 and IIS-1632935.

2.1 Ternarization with single-scalar

As an instance of (1), Li et al. [10] proposed to solve the following problem to perform ternarization:

$$(\alpha^*, T^*) = \arg\min_{\alpha,T} \|\alpha T - W^r\|^2 \quad \text{subject to} \quad \alpha > 0,\ T \in \{-1, 0, 1\}^n \tag{2}$$

with $(\alpha^*, T^*) \in \mathcal{T} = \mathbb{R}_+ \times \{-1, 0, 1\}^n$ and $\mathbf{prox}_{\mathcal{T}}(W^r) = \alpha^* T^*$, where n is the size of W^r. In [12], Rastegari et al. derived the exact expression of $\mathbf{prox}_{\mathcal{T}}$ for binarizing the weight with $\mathcal{T} = \mathbb{R}_+ \times \{-1, 1\}^n$. In [10], the solution to (2) was approximated under some statistical assumptions made on the distribution of weights.

In what follows, we derive the formula for the ternarization $\mathbf{prox}_{\mathcal{T}}(W^r)$ without imposing any assumption on W^r. Let us first fix the sparsity of T to be k, then we have $\|T\|^2 = k$ and $|\langle T, W^r\rangle| \le \|W^r_{[k]}\|_1$, where $W^r_{[k]}$ extracts the k largest entries in magnitude of W^r and enforces 0 elsewhere. Therefore,

$$\begin{aligned}
\|\alpha T - W^r\|^2 &= \alpha^2\|T\|^2 - 2\alpha\langle T, W^r\rangle + \|W^r\|^2 = k\alpha^2 - 2\langle T, W^r\rangle\alpha + \|W^r\|^2 \\
&= k\left(\alpha - \frac{\langle T, W^r\rangle}{k}\right)^2 - \frac{\langle T, W^r\rangle^2}{k} + \|W^r\|^2 \ge -\frac{\langle T, W^r\rangle^2}{k} + \|W^r\|^2 \\
&\ge -\frac{\|W^r_{[k]}\|_1^2}{k} + \|W^r\|^2.
\end{aligned} \tag{3}$$

Note that the above lower bound for $\|\alpha T - W^r\|^2$ is tight as will be seen later. Since $\|W^r\|^2$ is a constant, the optimal sparsity k^* is settled automatically so as to maximize the term $\frac{\|W^r_{[k]}\|_1^2}{k}$ in (3), i.e.,

$$k^* = \arg\max_k \frac{\|W^r_{[k]}\|_1^2}{k}.$$

Finally, to achieve the minimum in (3) w.r.t. k^*, we let

$$T^* = \text{sign}(W^r_{[k^*]})$$

so that $\langle T^*, W^r\rangle = \|W^r_{[k^*]}\|_1$, and take

$$\alpha^* = \frac{\langle T^*, W^r\rangle}{k^*} = \frac{\|W^r_{[k^*]}\|_1}{k^*}.$$

Solving the problem (2) mainly involves sorting magnitudes of the elements in W^r and computing accumulative sum of the sorted sequence, which requires the complexity of $O(n\log(n))$.

2.2 Ternarization with dual-scalar

In the recent paper [16], to enable stronger model capacity and performance, Zhu et al. introduced two real-valued scaling factors associated with the negative and positive weights in W^r, respectively, which hereby we denote by α_- and α_+. In this case, $\mathcal{T}$ in (1) can be viewed as $\mathbb{R}_+^2 \times \{-1, 0, 1\}^n$. Their solution to (1) is again inexact. To find the $\mathbf{prox}_{\mathcal{T}}(W^r)$ for such $\mathcal{T}$, we split W as $W^r = W^r_- + W^r_+$ into negative and positive parts, and correspondingly decompose (1) into two independent subproblems:

$$\begin{cases}
(\alpha^*_-, T^*_-) = \arg\min_{\alpha_-, T_-} \|\alpha_- T_- - W^r_-\|^2 \quad \text{subject to} \quad \alpha_- > 0,\ T_- \in \{-1, 0\}^n, \\
(\alpha^*_+, T^*_+) = \arg\min_{\alpha_+, T_+} \|\alpha_+ T_+ - W^r_+\|^2 \quad \text{subject to} \quad \alpha_+ > 0,\ T_+ \in \{0, 1\}^n,
\end{cases} \tag{4}$$

both of which can be tackled in the same manner as done for (2), and then the ternarized weight is given by $\mathbf{prox}_{\mathcal{T}}(W^r) = \alpha^*_- T^*_- + \alpha^*_+ T^*_+$.

2.3 A stochastic proximal gradient method

To preserve the performance of the original full-precision DNN, however, sufficient number of scaling factors are necessary for the ternary weight. So we need to break down the weight W into blocks and assign a scaling factor to each one. In order to leverage distribute law of multiplications efficiently during forward propagation, each block should be a 4-D weight tensor of a convolutional layer, a 3-D tensor of a channel, or just a filter matrix. Apparently, the more detailed the partitioning of W, the generally higher the accuracy of TNN. But the downside is that we will have less memory savings and potentially need to do more multiplications in the computation. Our proposed algorithm for training TNN is straightforward, as summarized in Algorithm 1.

Algorithm 1 Training one minibatch with proximal operator

Input: Current ternary weight $W^{(t)}$, cost function $C^{(t)}(W)$ restricted to the current minibatch, and learning rate $\eta^{(t)}$.
Output: Updated ternary weight $W^{(t+1)}$.
$W^r = \text{UpdateParameters}(W^{(t)}, \nabla C^{(t)}(W^{(t)}), \eta^{(t)})$. // Update the temporary real weight W^r with rules like SGD or ADAM [7].
for i = 1, 2,..., m **do**
  $W_i^{(t+1)} = \mathbf{prox}_{\mathcal{T}}(W_i^r)$ // Ternarize the i-th block of W^r by solving (2) or (4).
end for

We remark that the proposed algorithm differs from that used in [12] or [10], in that we update W^r using current ternary weight $W^{(t)}$ instead of the real weight from the previous minibatch training. When SGD is applied for updating $W^r = W^{(t)} - \eta^{(t)}\nabla C^{(t)}(W^{(t)})$, for example, then Algorithm 1 can be interpreted as stochastic proximal gradient descent [13, 3] for minimizing the cost function $C(W)$ subject to the ternary constraint upon the blocks of W, i.e.,

$$\min_{W} C(W) \quad \text{subject to} \quad W_i \in \mathcal{T},\ i = 1, \dots, m,$$

where m denotes the total number of blocks partitioned. For non-convex optimization problems like the above, a good initialization is desired, which motivates us to extract the weights from a trained full-precision DNN to initialize W^r.

3 Experiments

In this section, we report experiment results on the MNIST and CIFAR-10 benchmark datasets. Our TNN is implemented in the Theano [15] framework. We benchmark Ternary weight networks (TWN) [10], BinaryConnect [1] and Binarized Neural Networks (Binarized Net) [2]. We use the same network architecture, the same number of scaling factors, the same regularization method (L2 weight decay), and the same update rule (SGD with momentum) as TWN [10]. The major difference is that the ternarization we performed is accurate as discussed in section 2. Algorithm 1 converges fast when initialized with pre-trained full-precision weights, so a small epoch size is sufficient and efficient. The detailed configurations are provided in Table 1. Table 2 summarizes the best validation accuracy rates on MNIST and CIFAR-10 trained with 70 epochs. Table 3 shows the percentages of positive, negative and zero weights of our trained TNN.

3.1 MNIST

In MNIST, we adopt the LeNet-5 [9] architecture, which is

$$\text{32-C5 + MP2 + 64-C5 + MP2 + 512FC + SVM.}$$

Each convolutional layer has 32/64 filters of size 5×5 with a max-pooling layer of stride 2. There are two convolutional layers, followed by a fully connected layer. The top layer is a SVM classifier with 10 labels.

The objective function has a hinge loss with L2 regularization, optimized. We did not use any data-augmentation, preprocessing or unsupervised pretraining to improve experiment performance.

[1] LR is divided by 10 after the first 15 epochs, and decay again after each set of 10 epochs.
[2] Experiment data from [10] (version 1) with the same network architectures.

	MNIST	CIFAR-10
Network architecture	LeNet-5	VGG7-K
L2 weight decay	1e-4	1e-4
Mini batch size	50	100
Initial learning rate(LR)	0.01	0.1
LR decay rate	0.1	0.1
LR decay epochs[1]	15, 25	15, 25
Activation	ReLU	ReLU
Momentum	0.9	0.9

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

Neural Networks	MNIST	CIFAR-10(VGG7-16)	CIFAR-10(VGG7-32)
TNN	**99.44**	**88.46**	**91.62**
TWN[2]	99.35	$\sim$ 86	$\sim$ 89
BinaryConnect	98.82	-	-
Binarized Net	88.6	-	-

Table 2: Results of experiments. Validation accuracies (%).

We use Batch Normalization with a minibatch of size 50 to speed up the training. As typically done, we use the last 10000 samples of the training set for validation. See Figure 1 for its validation accuracy curve.

3.2 CIFAR-10

In CIFAR-10 experiment, we use the same network architecture as that of TWN [10], VGG7-K, that is

$$2\times\text{(K-C3)} + \text{MP2} + 2\times\text{(2K-C3)} + \text{MP2} + 2\times\text{(4K-C3)} + \text{MP2} + \text{8K-FC} + \text{Softmax}.$$

There are three convolutional layers, followed by a fully connected layer with 8K neurons. Each convolutional block consists of two convolutional layers with K/2K/4K 3×3 filters and a 2×2 max-pooling layer. This architecture is inspired by VGG [14].

We use Batch Normalization with minibatch of size 100 to speed up the training. The last 5000 samples of the training set are used for validation. The original 32×32 RGB images are subtracted with their pixel means during data preprocessing. The data augmentation techniques as in [5] are applied 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 image. We choose $K = 16, 32$ in our experiments and plot their validation curves, as shown in Figure 2.

4 Conclusion

A stochastic proximal gradient method based on an exact proximal operator for ternarization, is proposed for training DNN with ternary weights, which gives the state-of-the-art performance on

Dataset	+	−	0
MNIST	23.63	29.56	46.80
CIFAR-10(VGG7-16)	27.47	33.57	38.96
CIFAR-10(VGG7-32)	27.41	34.54	38.05

Table 3: Percentages (%) of positive (+), negative (−), and zero (0) weights of the trained TNN.

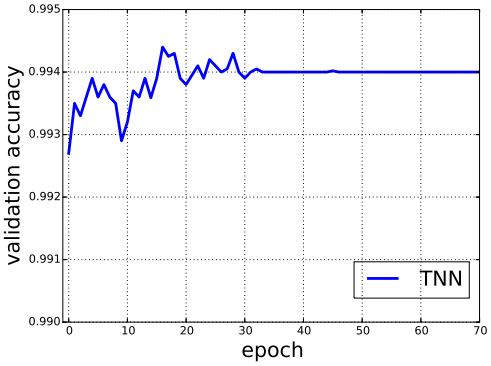

Figure 1: MNIST

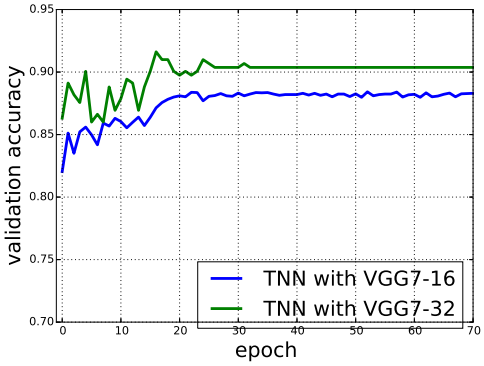

Figure 2: CIFAR-10

MNIST and CIFAR-10 benchmarks. The experiments show that the derived proximal operator tends to favor the ternarization more than the heuristic one of TWN [10].

References

[1] M. Courbariaux, Y. Bengio, and J. David. BinaryConnect: Training Deep Neural Networks with Binary Weights during Propagations. In *Advances in Neural Information Processing Systems*, page 3123–3131, 2015.

[2] M. Courbariaux, I. Hubara, D. Soudry, and R. El-Yaniv, Y. Bengio. Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR*, 2016.

[3] J. Duchi and Y. Singer. Efficient Online and Batch Learning Using Forward Backward Splitting. *Journal of Machine Learning Research*, 10:2899-2934, 2009.

[4] S. Han, H. Mao, and W. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations*, 2016.

[5] K.He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385*, 2015.

[6] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, 2015.

[7] D. Kingma and J. Ba. ADAM: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, page 1097-1105, 2012.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, 86(11):2278-2324, 1998.

[10] F. Li, B. Zhang, and B. Liu. Ternary Weight Networks. *arXiv:1605.04711*, 2016.

[11] N. Parikh and S. Boyd. Proximal Algorithms. *Foundations and Trends in Optimization*, 1(3):123-231, 2013.

[12] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, *arXiv:1603.05279*, 2016.

[13] L. Rosasco, S. Villa, and B. Vu. Convergence of Stochastic Proximal Gradient Algorithm. *arXiv:1403.5074*, 2014.

[14] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv:1409.1556*, 2014.

[15] Theano Development Team. Theano: A Python Framework for Fast Computation of Mathematical Expressions. *arXiv:1605.02688*, 2016.

[16] C. Zhu, S. Han, H. Miao, and W. Dally. Trained Ternary Quantization. Available at https://openreview.net/pdf?id=S1_pAu9xl.