Understanding BatchNorm in Ternary Training

Eyyüb Sari — Huawei Noah's Ark Lab, QC, Canada
Vahid Partovi Nia — Huawei Noah's Ark Lab, QC, Canada

Abstract

Neural networks are comprised of two components, weights and activation function. Ternary weight neural networks (TNNs) achieve a good performance and offer up to 16x compression ratio. TNNs are difficult to train without BatchNorm and there has been no study to clarify the role of BatchNorm in a ternary network. Benefiting from a study in binary networks, we show how BatchNorm helps in resolving the exploding gradients issue.

1 Introduction

Compression of Deep Neural Networks (DNNs) is crucial for deploying these huge and energy-hungry model on edge devices. Quantization methods are a set of techniques targeting reduced bit-precision representation. Two well-known extreme cases are binary and and ternary networks, that allow up to 32x and 16x compression rate, respectively. Contrary to binary weights $-1,+1$, ternary weights $-1,0,+1$ allow for representing 0. Greater flexibility is provided by this scheme, because it offers discarding a value as a builtin operation, which is specially helpful in keeping accuracy in the presence of point-wise, depth-wise convolution. The role of BatchNorm in binary networks with binary activations is already studied in [1]. [2] reports BatchNorm helps training binary *and* ternary networks. As ternary networks with full-precision activations are very different models, we are wondering if BatchNorm plays a similar role in ternary networks.

2 Notation

Ternary neural networks (TNNs) use full-precision weights during training which are ternarized during forward propagation. The full-precision weights act as latent parameters and allow for incremental updates. Let $x \in \mathbb{R}$, given a threshold Δ we define the ternary function as,

$$\text{tern}(x) = \begin{cases} -1 & \text{if } x < -\Delta \\ +1 & \text{if } x > \Delta \\ 0 & \text{if } -\Delta \le x \le \Delta \end{cases} \tag{1}$$

For a weight w for which we apply $\text{tern}(w)$ during forward propagation, we define its gradient with respect to a loss function $\mathcal{L}(.)$ as $\frac{\partial \mathcal{L}}{\partial w}\Big|_{w=\text{tern}(w)}$. The gradient is evaluated for the ternarized weight but accumulated in full-precision. However, at the initialization step, weights are drawn from a given random number generator. Thus, applying ternary function on them can be seen as applying a transformation on a random variable, $\tilde{w}_t$.

$$\text{tern}(w) = \tilde{w}_t = \begin{cases} -1 & \text{with } p_1 = \mathbb{P}(w < -\Delta) \\ +1 & \text{with } p_2 = \mathbb{P}(w > \Delta) \\ 0 & \text{with } p_3 = 1 - p_1 - p_2 \end{cases} \tag{2}$$

This setting is very similar to the binary setting of [1], for $\Delta = 0$, or equivalently $p_3 = 0$. This property helps to use the result of [1] and generalize it towards the ternary case.

Following the initialization schemes such as [3] or [4], weights are drawn from symmetric distributions about zero (eg. uniform). Therefore $p_1 = p_2$ and we obtain the following identities for the expectation and variance of the transformed random variable

$$\mathbb{E}[\tilde{w}_t] = 0 \qquad \mathbb{V}(\tilde{w}_t) = 2p_1. \tag{3}$$

The variance of the initial full-precision random variable plays a crucial role in the variance of the ternary random variable, and this is where this study differs from [1]. We follow the notation of [1] to save space. Denote the dot product $\mathbf{s}_b^l \in \mathbb{R}^{K_l}$ of the batch sample b in the neural network layer l, that has K_l number of neurons. Define f to be the element-wise activation function, and $\mathbf{x}_b$ to be the input, $\mathbf{W}^l \in \mathbb{R}^{K_{l-1} \times K_l}$ with elements $\mathbf{W}^l = [w^l_{kk'}]$ to be the weight matrix;. Here we use $w^l[i,j]$ to refer to a single element of $\mathbf{W}^l$, in which all of them are i.i.d. so we drop the index $[i,j]$ and simply denote it by w^l.

$$\frac{\partial \mathcal{L}}{\partial s^l_{bk}} = f'(s^l_{bk}) \sum_{k'=1}^{K_{l+1}} w^{l+1}_{kk'} \frac{\partial \mathcal{L}}{\partial s^{l+1}_{bk'}}, \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial w^l_{k'k}} = \sum_{b=1}^{B} s^{l-1}_{bk'} \frac{\partial \mathcal{L}}{\partial s^l_{bk}}. \tag{5}$$

for the details of development see [1].

Assume that the feature element x and the weight element w are centred and i.i.d. Let k denote the current neuron and k' denote the previous or the next layer neuron. For the ReLU activation function, one can show $\mathbb{V}(s^l_{bk}) = \mathbb{V}(x) \prod_{l'=1}^{l-1} \frac{1}{2} K_{l'} \mathbb{V}(w^{l'})$, where $\mathbb{V}(w^{l'})$ is the variance of the weight in layer l' if w is drawn from a uniform distribution symmetric about zero.

By applying similar mathematical mechanics of [1], the variance of the gradient for a neuron is

$$\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}\right) = \mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^L}\right) \prod_{l'=l+1}^{L} \frac{1}{2} K_{l'} \mathbb{V}(w^{l'}), \tag{6}$$

which explodes or vanishes depending on $\mathbb{V}(w^{l'})$. This is the main reason common full-precision initialization methods suggest $\mathbb{V}(w^l) = \frac{2}{K_l}$.

3 BatchNorm Role

To control the variance throughout layers during backpropagation we know $\mathbb{V}(w^l) = \frac{2}{K_l}$ is needed. In the ternary weight case, we need $\mathbb{V}(\tilde{w}^l_t) = 2p_1 = \frac{2}{K_l}$. It is easy to see $p_1 = \frac{1}{K_l}$ is required. Common initialization [4] draws $\tilde{w}^l \sim \mathbb{U}(-\sqrt{\frac{6}{K_l}}, \sqrt{\frac{6}{K_l}})$

$$\mathbb{P}(w^l < -\Delta) = p_1 = \frac{1}{2} - \frac{\Delta}{2\sqrt{\frac{6}{K_l}}} \tag{7}$$

To satisfy (7), the threshold has to be properly set

$$\Delta = 2\sqrt{\frac{6}{K_l}}\left(\frac{1}{2} - \frac{1}{K_l}\right). \tag{8}$$

In real world settings, e.g. for a convolutional layer with 3×3 kernel and 128 filters, $p_1 \approx 8 \times 10^{-4}$. Therefore, the threshold would be set so that more than 99% of the weights are zero to control the variance. As a big downfall, learning is made impossible in this case as most of the weights are set to zero. Contrary to our conclusion, let's suppose the threshold is given so that the learning is feasible, for instance Δ is given so that $< 50\%$ of ternary weights are set to zero

$$\mathbb{V}(\tilde{w}^l_t) = 2p_1 = 1 - \frac{\Delta}{\sqrt{\frac{6}{K_l}}}, \tag{9}$$

for any given Δ. In the literature [5] suggests to set $\Delta_l = 0.7\mathbb{E}[|w^l|]$. Following common initialization schemes

$$\Delta_l = \frac{0.7}{2}\sqrt{\frac{6}{K_l}} \tag{10}$$

and (9) reduces to $\mathbb{V}(\tilde{w}^l_t) = 1 - \frac{0.7}{2} = 0.65$. In this setting, variance is bigger than $\frac{2}{K_l}$ which produces exploding gradients. The situation is similar to the binary case reported in [1], giving us a reason to take a closer look to BatchNorm in ternary setting.

Suppose a mini batch of size B for a given neuron k. Let $\hat{\mu}_k, \hat{\sigma}_k$ be the mean and the standard deviation of the dot product $s^l_{bk}, b = 1, \dots B$. For a given layer l, BatchNorm is defined as $\text{BN}(s_{bk}) \equiv z_{bk} =$

$\gamma_k \hat{s}_{bk} + \beta_k$, where $\hat{s}_{bk} = \frac{s_{bk} - \hat{\mu}_k}{\hat{\sigma}_k}$ is the standardized dot product and the pair (γ_k, β_k) is trainable, and often initialized to $(1, 0)$. Following [1], it is easy to show

$$\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}\right) = \left(\frac{\gamma^l_k}{B\hat{\sigma}^l_k}\right)^2 \{B^2 + 2B - 1 + \mathbb{V}(\hat{s}^{l^2}_{bk})\} \frac{1}{2} K_{l+1} \mathbb{V}(\tilde{w}^{l+1}_t) \mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^{l+1}}\right). \quad (11)$$

Following common full precision initialization [4] assumptions, i.e. weights and activation are i.i.d. and weights are centred about zero, for a layer l,

$$\hat{\sigma}^2_k = K_{l-1} \frac{1}{2} \mathbb{V}(\hat{s}^{l-1}_b) \mathbb{V}(\tilde{w}^l_t) = K_{l-1} \frac{1}{2} \mathbb{V}(\tilde{w}^l_t). \quad (12)$$

Therefore (11) reduces to,

$$\mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^l_{bk}}\right) = \frac{\{B^2 + 2B - 1 + \mathbb{V}(\hat{s}^{l^2}_{bk})\}}{B^2} \frac{K_{l+1}}{K_{l-1}} \mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^{l+1}}\right) \quad (13)$$

$$= \left\{1 + o\left(\frac{1}{B^{1-\varepsilon}}\right)\right\} \frac{K_{l+1}}{K_{l-1}} \mathbb{V}\left(\frac{\partial \mathcal{L}}{\partial s^{l+1}}\right). \quad (14)$$

The equation (13) gives confirms a similar conclusion as in binary case, i.e. BatchNorm indeed prevents exploding gradients.

4 Numerical Experiment

We evaluate four different scenarios on the CIFAR-10 dataset [6]. It contains $50,000$ training images and $10,000$ test images. Each image is 32×32 pixels with RGB channels. While training, data augmentation is applied. We pad the images with 4 zeroes on each side. After this step, a random crop of 32×32 is taken out the 36×36px padded images. Finally, images are uniformly randomly flipped horizontally. During training and test time, the images are normalized with $\mu = (0.4914, 0.4822, 0.4465)$, $\sigma = (0.247, 0.243, 0.261)$. We use the VGG-7 architecture defined in [5] with BatchNorm, ReLU activation function and ternary weights. Experiments on ResNet-56 [7] are also performed. The shortcut connection can alleviate the exploding gradient issues to some extent. Lightweight model are rather harder to train and are much sensitive to instabilities, therefore we also include a study on MobileNet-v1 that clarifies the effect of exploding gradient. Each model is trained for 150 epochs using SGD optimizer with momentum set to 0.9 and a starting learning rate set to 0.1. The learning rate is decayed by 10 at epochs 80 and 120. L_2 regularization is applied with $\lambda = 10^{-4}$. The mini-batch size is 100.

We experiments four setting for each architectures. i) BatchNorm and TWN threshold [5] (BN), ii) removing BatchNorm but keeping TWN threshold (No BN), iii) using BatchNorm but setting threshold as defined in (10) (Sparse BN), iv) and no BatchNorm with threshold from (10) (Sparse No BN). In fact, i) can be regarded as the baseline, not to be confused the fully full-precision model. ii) provides an interesting observation, relatively shallow model such as VGG-7 still achieve decent accuracy even if BatchNorm is not present, the model is too shallow to be show the exploding gradient effect. On the other side, ResNet-56 which is a deeper model and supposed to suffer from accuracy loss, but recovers because of the short-cut connection. MobileNet diverges without BatchNorm because the model being deeper than VGG-7 and includes no shortcut connection to compensate for the exploding gradient effect. Items iii) and iv) confirms if the thresholds Δ_l are selected to ensure variance control (10), most of the weights are ternarized to zero and the models do not converge due to bad initialization. Even in this setting, ResNet-56 is still able to produce better outputs than a random predictor because of the information being carried on via the shortcut connections.

5 Conclusion

We find that theoretically, gradient explosion could be prevented without the use of BatchNorm by setting a proper threshold for mapping to zero and these results are backed up with numerical experiments. In our theoretical finding choosing an appropriate threshold Δ sets most of the weights to zero, which in practice do not allow TNNs to converge. Also, BatchNorm indeed also prevents gradient explosion independent of the chosen Δ.

	BN	No BN	Sparse BN	Sparse No BN
VGG-7	93.5	78.1	-	-
ResNet-56	92.7	85.7	38.9	39.3
MobileNet	88.3	-	-	-

Table 1: Ablation of BatchNorm and threshold on VGG-7, ResNet-56 and MobileNet, maximum accuracy achieved after training. BatchNorm and TWN threshold [5] (BN), removing BatchNorm but keeping TWN threshold (No BN), using BatchNorm but setting threshold as defined in (10) (Sparse BN), and no BatchNorm with threshold from (10) (Sparse No BN). Results are not reported if the network did not converge (i.e. not better than random)

6 Acknowledgement

We would like to thank Huawei CBG Software Shanghai colleagues Mohan Liu and Li Zhou for their fruitful technical discussions. We also thank Yanhui Geng and Jin Tang for their support throughout the project.

References

[1] Eyyüb Sari, Mouloud Belbahri, and Vahid Partovi Nia. How does batch normalization help binary training? *arXiv*, abs/1909.09139, 2019.

[2] Arash Ardakani, Zhengyun Ji, Sean C. Smithson, Brett H. Meyer, and Warren J. Gross. Learning recurrent binary/ternary weights. In *International Conference on Learning Representations*, 2019.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[5] Fengfu Li and Bin Liu. Ternary weight networks. *arXiv*, abs/1605.04711, 2016.

[6] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.