BNN+: IMPROVED BINARY NETWORK TRAINING

Sajad Darabi,
Department of Computer Science
University of California,
Los Angeles, CA, USA
sajad.darabi@cs.ucla.edu

Mouloud Belbahri
Department of Mathematics and Statistics,
University of Montreal
Montreal, Quebec, Canada
belbahrim@dms.umontreal.ca

Matthieu Courbariaux
Montreal Institute of Learning Algorithms, University of Montreal
Montreal, Quebec, Canada
matthieu.courbariaux@gmail.com

Vahid Partovi Nia
Huawei Technologies, Montreal Research Center
Montreal, Quebec, Canada
vahid.partovinia@huawei.com

ABSTRACT

The deployment of Deep neural networks (DNN) on edge devices has been difficult because they are resource hungry. Binary neural networks (BNN) help to alleviate the prohibitive resource requirements of DNN, where both activations and weights are limited to 1-bit. There is however a significant performance gap between BNNs and floating point DNNs. To reduce this gap, We propose an improved binary training method, by introducing a new regularization function that encourages training weights around binary values. In addition, we add trainable scaling factors to our regularization functions. We also introduce an improved approximation of the derivative of the sign activation function in the backward computation. These modifications are based on linear operations that are easily implementable into the binary training framework. We show experimental results on CIFAR-10 obtaining an accuracy of 87.4%, on AlexNet and 83.9% with DoReFa network. On ImageNet, our method also outperforms the traditional BNN method and XNOR-net, using AlexNet by a margin of 4% and 2% top-1 accuracy respectively. In other words, we significantly reduce the gap between BNNs and floating point DNNs.

1 INTRODUCTION

Deep neural networks (DNNs) have demonstrated success for many supervised learning tasks ranging from voice recognition to object detection (Szegedy et al., 2015; Simonyan & Zisserman, 2014; Iandola et al., 2016). The focus has been on increasing accuracy, in particular for image tasks, where deep convolutional neural networks (CNNs) are widely used. However, their increasing complexity poses a new challenge, and has become an impediment to widespread deployment in many applications, specifically when trying to deploy such models to resource-constrained and lower-power devices. A typical DNN architecture contains tens to thousands of layers, resulting in millions of parameters. As an example, AlexNet (Krizhevsky et al., 2012) requires 200MB of memory, VGG-Net (Simonyan & Zisserman, 2014) requires 500MB memory. Large model sizes are further exacerbated by their computational cost, requiring GPU implementation to allow real-time inference. Such requirements evidently cannot be accustomed by edge devices as they have limited memory, computation power, and battery. This motivated the community to investigate methods for compressing and reducing computation cost of DNNs.

To make DNNs compatible with the resource constraints of low power devices, there have been several approaches developed, such as network pruning (LeCun et al., 1990), architecture design (Sandler et al., 2018), and quantization (Courbariaux et al., 2015; Han et al., 2015). In particular, weight compression using quantization can achieve very large savings in memory, where binary (1-bit), and ternary (2-bit) approaches have been shown to obtain competitive accuracy as compared to their full precision counterpart (Hubara et al., 2016; Zhu et al., 2016; Tang et al., 2017). Using such schemes reduces model sizes by 8x to 32x depending on the bit resolution used for computations. In addition to this, the speed by quantizing the activation layers. This way, both the weights and activations are quantized so that one can replace the expensive dot products and activation function evaluations with bitwise operations. This reduction in bit-width benefits hardware accelerators such as FPGAs and neural network chips.

An issue with using low-bit DNNs is the drop in accuracy compared to its full precision counterpart on complex datasets, and this is made even more severe upon quantizing the activations. This problem is largely due to noise and lack of precision in the training objective of the networks during back-propagation (Lin et al., 2017). Although, quantizing the weights and activations have been attracting large interests thanks to their computational benefits, closing the gap in accuracy between the full precision and the quantized version remains a challenge. Indeed, quantizing weights cause drastic information loss and make neural networks harder to train due to a large number of sign fluctuations in the weights. Therefore, how to control the stability of this training procedure is of high importance. In theory, it is infeasible to back-propagate in a quantized setting as the weights and activations employed are discontinuous and discrete. Instead, heuristics and approximations are proposed to match the forward and backward passes. Often weights at different layers of DNNs follow a certain structure. How to quantize the weights locally, and maintain a global structure to minimize a common cost function is important (Li et al., 2017).

Our contribution consists of three ideas that can be easily implemented in the binary training framework presented by Hubara et al. (2016) to improve convergence and generalization of binary neural networks. First, we improve the straight-through estimator introduced in Hubara et al. (2016), i.e., we modify the approximation of the sign function in the backward pass. Second, we propose a new regularization function that encourage training weights around binary values. Third, a scaling factor is introduced in the regularization function as well as network building blocks to mitigate accuracy drop due to hard binarization. Our method is evaluated on CIFAR-10 and ImageNet datasets and compared to other binary methods. We show accuracy gains to compared traditional binary training.

In the following sections we first introduce related work on binary networks in section 2, and highlight several approaches to binary network training. Then we proceed with the binary training framework and introduce our modifications in section 3. In section 4, experiments are explained on both CIFAR10 and ImageNet datasets demonstrating the effectiveness of the proposed method. Finally we conclude with a discussion of our results in section 4.3.

2 Related Work

We focus on challenges faced when training binary neural networks. The training procedure emulates binary operations by restricting the weights and activations to single-bit so that computations of neural networks can be implemented using arithmetic logic units (ALU) using XNOR and popcount operations. More specifically, XNOR and popcount instructions are readily available on most CPU and GPU processing units. Using more bits, such as 2 to 4 bits would incure quadratic cost compared to 1-bit when implementing popcount and related operations. The goal of this binary training is to reduce the model size and gain inference speedups without performance degradation.

Primary work done by Courbariaux et al. (2015) (BinaryConnect) trains deep neural networks with binary weights $\{-1, +1\}$. They propose to quantize real values using the sign function. On the backward pass, the propagated gradient apply updated to weights satisfying $|w| \leq 1$. Once the weights are outside of this region they are no longer updated, which is implemented by clipping the weights between $\{-1, +1\}$ after each optimization step. In that work, they did not consider binarizing the activation functions. BNN (Hubara et al., 2016) is the first purely binary network quantizing both the weights and activations. They achieve comparable accuracy to their prior work on BinaryConnect, and achieve significantly close performance to full-precision, by using large and deep networks. However, they performed poorly on large datasets like ImageNet (Russakovsky

et al., 2015). The resulting network presented in their work obtains $32\times$ compression rate and approximately $7\times$ increase in inference speed.

To alleviate the accuracy drop of BNN on larger datasets, Rastegari et al. (2016) propose XNOR-Net, where they strike a trade-off between compression and accuracy through the use of scaling factors for both weights and activation functions. They show performance gains compared to BNN on ImageNet classification. During training, the scaling factors for both the weights and activations are computed dynamically, hindering performance. As training binary networks take longer to converge compared to full precision networks, this could be a problem. Further, they introduced an additional complexity in implementing the convolution operations on the hardware, slightly reducing compression rate and speed up gains. DoReFa-Net (Zhou et al., 2016) further improves XNOR-Net by approximating the activations with more bits. The proposed rounding mechanism allows for low bit back-propagation as well, this can enable faster training of binary networks. Although they perform multi-bit quantization, their model still suffers from large accuracy drop upon quantizing the last layer.

Later in ABC-Net, Tang et al. (2017) propose several strategies for improving the accuracy of BNN and adjusting the learning rate for larger datasets. They show BNN achieve similar accuracy as XNOR-Net without the scaling overhead by adding a regularizer term which allows binary networks to generalize better. They also propose a modified BNN, where they adopt the strategy of increasing the number of filters, to compensate for accuracy loss similar to wide reduced-precision networks (Mishra et al., 2017). Friesen & Domingos (2017) study limiting activations to hard thresholding units $(-1, 1)$ along with the backward STE and devise an algorithm for improved approximations. More recently, Liu et al. (2018) develop a second-order approximation to the sign activation function for a more accurate backward update. In addition to this, they pre-train the network in which they want to binarize in full precision using the hard tangent hyperbolic (htanh) activation, which is illustrated in our Figure 4. They use the pre-trained network weights as an initialization for the binary network to obtain state of the art performance.

3 Improved Binary Training

Training a binary neural network faces two major challenges: quantizing the weights, and the activation functions. As both weights and activations are binary, the traditional continuous optimization methods such as SGD cannot be directly applied. Instead, a continuous approximation is used for the sign activation during the backward pass. Further, the gradient of the loss with respect to the weights are small. As a result, when the training progresses, weights' signs remain unchanged. These are both addressed in our proposed method. In this section, we present our approach to training 1-bit CNNs in detail.

3.1 Binary training

We quickly revisit quantization through binary training as first presented by (Courbariaux et al., 2015). In (Hubara et al., 2016), the weights are quantized by using the sign function which is $+1$ if $w > 0$ and -1 otherwise.

In the forward pass, the real-valued weights are binarized to w^b, and the resulting loss is computed using binary weights throughout the network. For hidden units, the sign function non-linearity is used to obtain binary activations. Prior to binarizing, the real weights are stored in a temporary variable w. The variables w are stored because one cannot back-propagate through the sign operation as its gradient is zero everywhere, and hence disturbs learning. To alleviate this problem the authors suggest using a straight through estimator (Hinton, 2012) for the gradient of the sign function. This method is a heuristic way of approximating the gradient of a neuron,

$$\frac{dL(w)}{dw} \approx \frac{dL}{dw}\bigg|_{w=w^b} \mathbf{1}_{\{|w|\leq 1\}} \tag{1}$$

where L is the loss function and $\mathbf{1}(.)$ is the indicator function. The gradients in the backward pass are then applied to weights that are within $[-1, +1]$. The training process is summarized in Figure

1. As weights undergo gradient updates, they are eventually pushed out of the center region and instead make two modes, one at -1 and another at $+1$. This progression is also shown in Figure 2. Also we highlight the differences in terms of computation between the full precision network and the binary network in Figure 3.

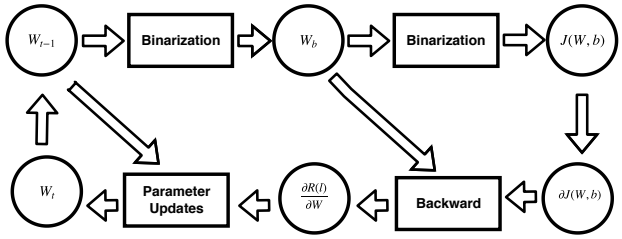

Figure 1: Binary training where arrows indicate operands flowing into operations or blocks.

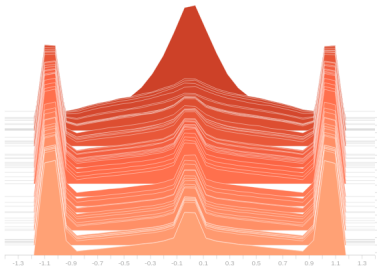

Figure 2: Progression of the weights training in BNN (Hubara et al., 2016) . As training progresses the weights create three modes: at -1, 0, and at $+1$.

3.2 Improved Training Method

Our first modification is on closing the discrepancy between the forward pass and backward pass. Originally, the sign function derivative is approximated using the $\mathrm{htanh}(x)$ activation derivative as shown in Figure 4. Instead, we modify the Swish-like activation (Ramachandran et al., 2017; Elfwing et al., 2018; Hendrycks & Gimpel, 2016), which has been shown to outperform other activation functions on various tasks. The modifications are performed by taking its derivative and centering it around 0. Let

$$\mathrm{SS}_\beta(x) = 2\sigma(\beta x)\left[1 + \beta x\{1 - \sigma(\beta x)\}\right] - 1, \tag{2}$$

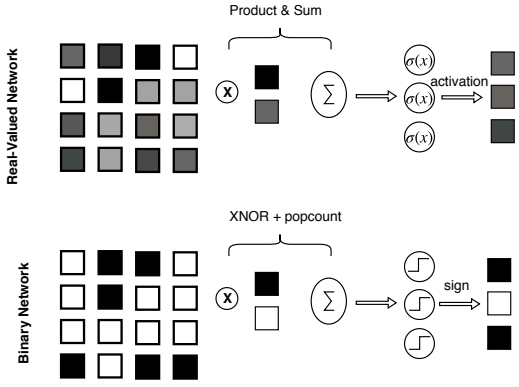

Figure 3: Full-precision network with floating-values (top panel) and binary neural network, with -1 and $+1$ activation and values (bottom panel).

where $\sigma(z)$ is the sigmoid function and the scale $\beta > 0$ controls how fast the activation function asymptotes to -1 and $+1$. The β parameter can be learned by the network or be hand-tuned as a hyperparameter. As opposed to the Swish function, where it is unbounded on the right side, the modification makes it bounded and a valid approximator of the sign function. As a result, we call this activation SignSwish or SSwish, and its gradient is

$$\frac{d\mathrm{SS}_\beta(x)}{dx} = \frac{\beta\{2 - \beta x \tanh(\frac{\beta x}{2})\}}{1 + \cosh(\beta x)}, \tag{3}$$

where $\tanh$ is the hyperbolic tangent and $\cosh$ is the hyperbolic cosine. Depending on the value of β, (3) is a closer approximation function to the derivative of the sign function compared to the derivative of the htanh activation. Comparisons are made in Figure 4.

Hubara et al. (2016) noted that the STE fails to learn weights near the borders of -1 and $+1$. As depicted in Figure 4, our proposed SignSwish activation alleviates this, as it remains differentiable near -1 and $+1$ allowing weights to change signs during training if necessary.

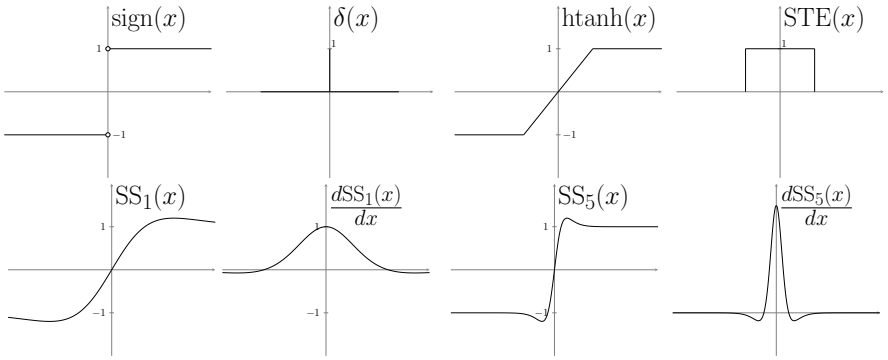

Figure 4: Forward and backward approximations. (Top-Left) The true forward and backward functions. (Top-Right) BNN training approximation. (Bottom-Left)SwishSwish $SS(x)$ for $\beta = 1$. (Bottom-Right) The modified activation, in this case $SS(x)$ is plotted for $\beta = 5$.

Note that the derivative $\frac{d}{dx}\mathrm{SS}_\beta(x)$ is zero at two points, controlled by β. Indeed, it is simple to show that the derivative is zero for $x \approx \pm 2.4/\beta$. By adjusting this parameter, it is possible to adjust the location at which the gradients start saturating, in contrast with the STE estimators where it is fixed.

3.2.1 Regularization function

In general, a regularization term is added to a model to prevent over-fitting and to obtain robust generalization. The two most commonly used regularization terms are L_1 and L_2 norms. If one were to embed these regularization functions in binary training, it would encourage the weights to be near zero, though this does not align with the objective of a binary network. A regularization function for binary networks should vanish upon the quantized values. Following this intuition, we define a function that encourages the weights around -1 and $+1$.

The Manhattan regularization function is defined as

$$R_1(w) = |\alpha - |w||, \tag{4}$$

whereas the Euclidean version is defined as

$$R_2(w) = (\alpha - |w|)^2, \tag{5}$$

where $\alpha \in \Re^+$. In Figure 5, we depict the different regularization terms to help with intuition.

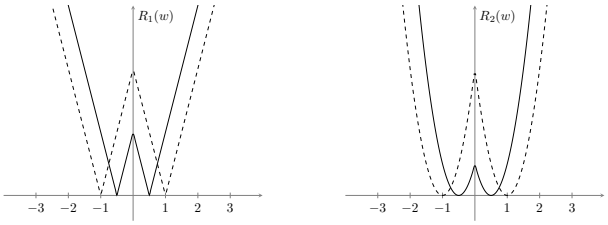

Figure 5: $R_1(w)$ (left) and $R_2(w)$ (right) regularization functions for $\alpha = 0.5$ (solid line) and $\alpha = 1$ (dashed line). The scaling factor α is trainable, as a result the regularization functions can adapt accordingly.

These regularization functions encode a quantization structure, when added to the overall loss function of the network encouraging weights to binary values. The difference between the two is in the rate at which weights are penalized when far from the binary objective, L_1 linearly penalizes the weights and is non-smooth compared to the L_2 version where weights are penalized quadritically. We further relax the hard thresholding of binary values $\{-1, 1\}$ by introducing scales α in the regularization function. This results in a symmetric regularization function with two minimums, one at $-\alpha$ and another at $+\alpha$. The scales are then added to the networks and multiplied into the weights after the binarization operation (refer to Figure 1). As these scales are introduced in the regularization function and are embedded into the layers of the network they can be learned using back-propagation. This is in contrast with the scales introduced in (Rastegari et al., 2016), where they compute the scales dynamically during training using the statistics of the weights after every training batch.

As depicted in Figure 5, in the case of $\alpha = 1$, the weights are penalized at varying degrees upon moving away from the objective quantization values, in this case, $\{-1, +1\}$.

One may note that the proposed regularizing terms are inline with the wisdom of the regularization function $R(w) = (1 - w^2)\mathbf{1}_{\{|w|\leq 1\}}$ as introduced in (Tang et al., 2017). A primary difference is in formulating it such that the gradients capture appropriate sign updates to the weights. Further, We also introduce trainable scaling factors, allowing weights to be quantized to intermediate values. Lastly, the regularization introduced in (Tang et al., 2017) does not penalize weights that are outside of $[-1, +1]$. One can re-define their function to include a scaling factor as $R(w) = (\alpha - w^2)\mathbf{1}_{\{|w|\leq\alpha\}}$. We no longer consider their regularization function and instead focus on the two presented in (4) and (5).

3.3 Training Method

Combining both the regularization and modified STE ideas, we adapt the training procedure by replacing the sign backward approximation with that of the derivative of SS_β activation (2). During training, the real weights are no longer clipped as in BNN training, as the network can back-propagate through the SS_β activation and update the weights correspondingly.

Additional scales are introduced to the network, which multiplies into the weights of the layers. The regularization terms introduced are then added to the total loss function,

$$J(W, b) = L(W, b) + \lambda_t \sum_h R(W_h, \alpha_h) \tag{6}$$

where $L(W, b)$ is the cost function, W and b are the sets of all weights and biases in the network, W_h is the set weights at layer h and α_h is the corresponding scaling factor. Here, $R(.)$ is the regularization function (4) or (5). Further, λ_t controls the effect of the regularization term, and can change as training progresses. To introduce meaningful scales, they are added to the basic blocks composing a typical convolutional neural network. For example, for convolutions, the scale is multiplied with the quantized weights after the convolution operation. Similarly, in a linear layer, the scales are multiplied into the quantized weights after the dot product operation. This is made more clear in Algorithm 1.

The scale α is a single scalar per layer, or as proposed in Rastegari et al. (2016) is a scalar for each filter in a convolutional layer. For example, given a CNN block with weight dimensionality $(C_{\text{in}}, C_{\text{out}}, H, W)$, where C_{in} is the number of input channels, C_{out} is the number of output channels, and H, W, the height and width of the filter respectively, then the scale parameter would be a vector of dimension C_{out}, that factors into each filter.

As the scales are learned jointly with the network through back-propagation, it is important to initialize them appropriately. In the case of the Manhattan penalizing term (4), given a scale factor α and a weight filter, the objective is to solve

$$\min_{\alpha} \sum_{h,w} |\alpha - |W_{h,w}||. \tag{7}$$

The minimum of the above is obtained when

$$\alpha^* = \text{median}(|W|) \tag{8}$$

Similarly, in the case of the Euclidean penalty (5), the minimum is obtained when

$$\alpha^* = \text{mean}(|W|), \tag{9}$$

which is similar to the optimal scaling factor derived in Rastegari et al. (2016). The difference here is that we have the choice to learn it with back-propagation in our framework, as opposed to computing the values dynamically. Hence, the scales are initialized with the corresponding optimal values after weights have been initialized first. The final resulting binary training method is detailed in Algorithm 1.

4 Experimental Results

We evaluate our proposed binary neural networks training framework by comparing the accuracy performance versus other proposed binary networks, Hubara et al. (2016); Rastegari et al. (2016); Tang et al. (2017), on CIFAR-10 and ImageNet datasets. We show accuracy gains and we discuss the results in their respective sections below.

4.1 CIFAR-10

The CIFAR-10 data Krizhevsky & Hinton (2009) consists of $50,000$ train images and a test set of $10,000$. We apply the common data augmentation strategy for CIFAR-10, where images are padded by 4 pixels on each side, a random 32×32 crop is taken, and a horizontal flip with probability 0.5, which then the image is normalized with the dataset statistics.

In this section we run an ablation study in order to better understand our proposed additions to the binary training framework. We train both, AlexNet[1] Krizhevsky et al. (2012), and DoReFA-Net Zhou et al. (2016) using the ADAM Kingma & Ba (2014) optimizer. For both networks batch normalization layers are added prior to activations.

To train AlexNet we use batch size of 64 for training. The initial learning rates were set to 0.005 and decayed by 0.1 on epoch 30 and 45 for a total of 55 epochs. For DoReFa-Net, the initial learning rates were set to 0.001 and decayed by 0.1 on epochs 100, 150 and 200 for a total of 250 epochs. In all of the experiments, we set the regularization parameter λ to $5 * 10^{-7}$. Lastly, the networks are trained from scratch where weights are initialized using Glorot & Bengio (2010) initialization. In the case of R_1 and R_2, scales are introduced for each convolution filter, and are initialized using the median or the mean of the of the absolute value of the filter weights. The performance of these scales is also compared to the original XNOR-net implementation. Lastly, for our ablation study on the parameter of the SwishSign, we try three different settings: one with trainable parameter β, and the two others with $\beta = 5$ and $\beta = 10$ fixed. The results for the ablation study are summarized in Table 1.

[1]For AlexNet we use the image transformations explained in the ImageNet section

[3]We follow the implementation `https://github.com/allenai/XNOR-Net` and add the various STEs to the implementation

[3]We use the SVHN model `https://github.com/ppwwyyxx/tensorpack/blob/master/examples/DoReFa-Net/svhn-digit-dorefa.py`

Algorithm 1 Binary training. L is the unregularized loss function. λ and R_j are the regularization terms (4) for $j = 1$ or (5) for $j = 2$. SS_β is the SignSwish function (2) and $(\mathrm{SS}_\beta)'$ is its derivative (3). N is the number of layers. $\circ$ indicates element-wise multiplication. BatchNorm() specifies how to batch-normalize the activation and BackBatchNorm() how to back-propagate through the normalization. ADAM() specifies how to update the parameters when their gradients are known.

Require: a minibatch of inputs and targets (x_0, x^*), previous weights W, previous weights' scaling factors α, and previous BatchNorm parameters θ.

Ensure: updated weights W^{t+1}, updated weights' scaling factors α^{t+1} and updated BatchNorm parameters θ^{t+1}.

$\{1.$ Forward propagation:$\}$
$s_0 \leftarrow x_0 W_0$ {We do not quantize the first layer.}
$x_1 \leftarrow \mathrm{BatchNorm}(s_0, \theta_0)$
for $k = 1$ to $N - 1$ **do**
  $x_k^b \leftarrow \mathrm{sign}(x_k)$
  $W_k^b \leftarrow \mathrm{sign}(W_k)$
  $s_k \leftarrow \alpha_k x_k^b W_k^b$ {This step can be done using mostly bitwise operations.}
  $x_{k+1} \leftarrow \mathrm{BatchNorm}(s_k, \theta_k)$
end for

$\{2.$ Backward propagation:$\}$
Compute $g_{x_N} = \frac{\partial L}{\partial x_N}$ knowing x_N and x^*
for $k = N - 1$ to 1 **do**
  $(g_{s_k}, g_{\theta_k}) \leftarrow \mathrm{BackBatchNorm}(g_{x_{k+1}}, s_k, \theta_k)$
  $g_{\alpha_k} \leftarrow g_{s_k} x_k^b W_k^b + \lambda \frac{\partial R_j}{\partial \alpha_k}$
  $g_{W_k^b} \leftarrow g_{s_k}^\top \alpha_k x_k^b$
  $g_{x_k^b} \leftarrow g_{s_k} \alpha_k W_k^b$
  {We use our modified straight-through estimator to back-propagate through sign:}
  $g_{W_k} \leftarrow g_{W_k^b} \circ (\mathrm{SS}_\beta)'(W_k) + \lambda \frac{\partial R_j}{\partial W_k}$
  $g_{x_k} \leftarrow g_{x_k^b} \circ (\mathrm{SS}_\beta)'(x_k)$
end for
$(g_{s_0}, g_{\theta_0}) \leftarrow \mathrm{BackBatchNorm}(g_{x_1}, s_0, \theta_0)$
$g_{W_0} \leftarrow g_{s_0} x_0$ {We did not quantize the first layer.}

$\{3.$ The update:$\}$
for $k = 0$ to $N - 1$ **do**
  $\theta_k^{t+1}, W_k^{t+1}, \alpha_k^{t+1} \leftarrow \mathrm{ADAM}(\eta, \theta_k, W_k, \alpha_k, g_{\theta_k}, g_{W_k}, g_{\alpha_k})$
end for

4.2 IMAGENET

The ILSVRC-2012 **?** dataset consists of $\sim$ 1.2M training images, and 1000 classes. For pre-processing the dataset we follow the typical augmentation: the images are resized to 256×256, then are randomly cropped to 224×224 and the data is normalized using the mean and standard deviation statistics of the train inputs; no additional augmentation is done. At inference time, the images are first scaled to 256×256, center cropped to 224×224 and then normalized.

We evaluate the performance of our training method on two architectures: AlexNet and Resnet-18 (He et al., 2016). Following previous work, we used batch-normalization before each activation function. Additionally, we keep the first and last layers to be in full precision, as we lose $2 - 3\%$ accuracy otherwise. This approach is followed by other binary methods that we compare to (Hubara et al., 2016; Rastegari et al., 2016; Tang et al., 2017). The results are summarized in Table 2. In all the experiments involving R_1 regularization we set the λ to 10^{-7} and R_2 regularization in ranges of $10^{-5} - 10^{-7}$. Also, in every network, the scales are introduced per filter in convolutional layers, and per column in fully connected layers. The weights are initialized using a pre-trained model with htan activation function as done in Liu et al. (2018). Then the learning rate for AlexNet is set to 2.33×10^-3 and multiplied by 0.1 at the 12^{th} and 18^{th} epochs for a total of 25 epochs trained. For

Table 1: Top-1 accuracies (in percentage) on CIFAR-10, using regularization functions (4) and (5) with AlexNet and DoReFa-Net. STE stands for Straigth-Through Estimator and SS_t for SwishSign with a trainable β. XNOR scales experiments are done using the original XNOR implementation.

Scale	**STE**	**AlexNet**	**DoReFa-Net**[2]
		Top-1	Top-1
R_1	SS_t	87.16	83.92
	SS_{10}	86.80	83.21
	SS_5	86.85	83.72
	tanh	87.06	82.79
	bireal	87.04	82.64
	htanh	86.90	82.78
R_2	SS_t	87.30	83.52
	SS_{10}	86.39	82.43
	SS_5	86.80	83.58
	tanh	87.13	82.33
	bireal	87.39	82.93
	htanh	87.20	82.33
XNOR[3]	SS_t	84.94	82.58
	SS_{10}	83.64	83.12
	SS_5	85.74	82.54
	tanh	80.30	81.68
	bireal	83.84	82.58
	htanh	83.87	82.08
None	SS_t	86.24	83.00
	SS_{10}	84.70	82.98
	SS_5	84.85	83.14
	tanh	86.12	82.33
	bireal	86.24	82.56
	htanh	85.97	82.51

the 18-layers ResNet, the learning rate is started from 0.01 and multiplied by 0.1 at the 10^{th}, 20^{th} and 30^{th} epochs. On the ImageNet dataset, we run a small ablation study of our regularized binary network training method with fixed β parameters.

Table 2: Top-1 and top-5 accuracies (in percentage) on ImageNet dataset, of different combinations of the proposed technical novelties on different architectures.

Reg.	**Activation**	**AlexNet**		**Resnet-18**	
		Top-1	Top-5	Top-1	Top-5
R_1	SS_5	46.11	75.70	52.64	72.98
	SS_{10}	46.08	75.75	51.13	74.94
	htanh	41.58	69.90	50.72	73.48
R_2	SS_5	45.62	70.13	53.01	72.55
	SS_{10}	45.79	75.06	49.06	70.25
	htanh	40.68	68.88	48.13	72.72
None	SS_5	45.25	75.30	43.23	68.51
	SS_{10}	45.60	75.30	44.50	64.54
	htanh	39.18	69.88	42.46	67.56

4.3 DISCUSSION

We proposed two regularization functions (4) and (5) with a trainable scaling factor α and an activation function (2) with a trainable parameter β. We ran several experiments to better understand the effect of the different modifications to the binary neural networks training method, especially using different regularizations and scale parameters β. The parameter β is trainable and adds only one equation through back-propagation. However, we fixed β throughout our ImageNet experiments to explicit values in order to speed-up the training.

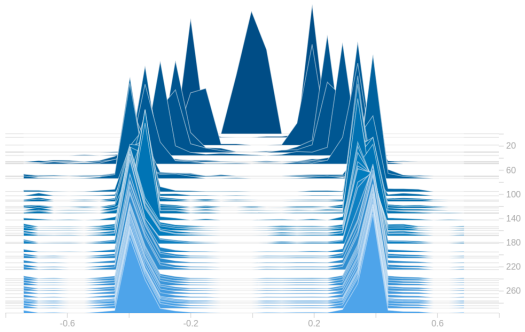

Figure 6: Weights progression of a single filter in a convolutional layer. As depicted, the weights are initialized with a normal distribution and as training progresses, the weights converge to two modes at $-\alpha$ and $+\alpha$.

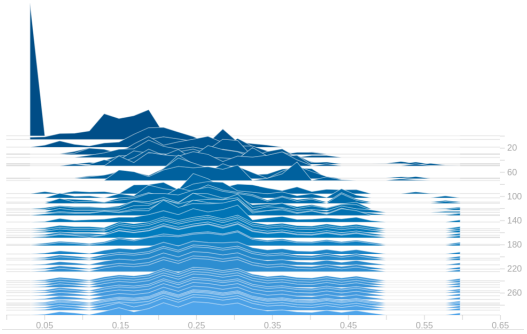

Figure 7: Scaling factors progression of a convolutional layer during training, with Manhattan regularization.

Through our experiments, we found that adding regularizing terms with heavy penalization degrades the networks ability to converge, as the term would result in total loss be largely due to the regularizing term and not the target cross entropy loss. Instead, the regularization term should be added and increased gradually to enforce quantized weights. Similarly, the regularizing constant λ was set to small values in Tang et al. (2017). As a result, we used λ with reasonably small values: $10^{-5} - 10^{-7}$, so that the scales move slowly as the weights gradually converge to stable values. Some preliminary experimentation was to gradually increase the regularization with respect to batch iterations updates done in training, though this approach requires careful tuning and was not pursued further. To see the effect of the regularization, in Figure6, the weight histogram with respect to training epochs is depicted. As shown, in contrast to Figure 2, two modes appear at $-\alpha$ and $+\alpha$. In addition to this, the histogram of the scaling factors for every filter of this convolutional layer is shown in Figure 7. The scales initialized with the median of absolute values finds a wider spread, empirically showing that the network is benefiting from this additional flexibility in scales as opposed to hard thresholding at the binary values -1 and $+1$.

To study the effect of our SwishSign, we compare it to the different backward approximations most recently proposed: bireal and the soft-hinge Friesen & Domingos (2017). From the DoReFa-Net experiments in Table 1 the SS_t with trainable parameter β performs the highest, achieving 83.92% top-1 accuracy; a 1% accuracy drop compared to the experiments ran in Friesen & Domingos (2017),

with hard thresholding units and real weights. The results of AlexNet on CIFAR-10 also show the same effect, where SwishSign improves the results. From Table 2, and referring to networks without regularization, we see the benefit of using SwishSign approximation versus the STE. This was also noted in Liu et al. (2018), where their second approximation provided better results.

There is not much difference between using R_1 versus R_2 towards model generalization although since the loss metric used was the cross-entropy loss, the order of R_1 better matches the loss metric with respect to the weights and hence could potentially help with tuning the regularization effect. Lastly, it seems that moderate values of β is better than small or large values. Intuitively, this happens because for small values of β, the gradient approximation is not good enough and as β increases the gradients become too large, hence small noise could cause large fluctuations in the sign of the weights.

We have shown that our method compares favorably with the method of Liu et al. (2018) on CIFAR-10. However, we had some difficulties doing a fair comparison on ImageNet. From our understanding, Liu et al. (2018) added some extra shortcut connections to their DNNs, which significantly improve the accuracy of both their BNNs and their full-precision baseline. As a result, we believe that their method may be, at least to some extent, complementary with ours.

As a final remark, we note that the learning rate is of great importance and properly tuning is required to achieve convergence. Table 3 summarizes the best results of the ablation study and compares with BinaryNet, XNOR-Net, and ABC-Net.

Table 3: Comparison of top-1 and top-5 accuracies of our method with BinaryNet, XNOR-Net and ABC-Net on ImageNet, summarized from Table 2. The results of BNN, XNOR, & ABC-Net are reported from the corresponding papers (Rastegari et al., 2016; Hubara et al., 2016; Tang et al., 2017). Results for ABC-NET on AlexNet were not available, and so is not reported.

Method	**AlexNet**		**Resnet-18**	
	Top-1	Top-5	Top-1	Top-5
Ours	$\mathbf{46.1}\%$	$\mathbf{75.7}\%$	$\mathbf{53.0}\%$	72.6%
BinaryNet	41.2%	65.6%	42.2%	67.1%
XNOR-Net	44.2%	69.2%	51.2%	$\mathbf{73.2}\%$
ABC-Net	-	-	42.7%	67.6%
Full-Precision	56.6%	80.2%	69.3%	89.2%

5 Conclusion

In this paper, we proposed training binary neural networks with two novel regularization functions along with learnable scaling factors parameters. Additionally, a SwishSign function is proposed for improving the backward approximation to the sign function, replacing the traditional STE. We demonstrate improvements on various architectures on both CIFAR-10 and ImageNet datasets. For future work, multi-bit regularization motivated by the same nature could be of interest, as well as formulating activations as soft thresholds into hard-thresholds using similar regularization techniques.

References

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 2018.

Abram L. Friesen and Pedro M. Domingos. Deep learning as a mixed convex-combinatorial optimization problem. *CoRR*, abs/1710.11573, 2017.

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*, pp. 249–256, 2010.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.

Geoffrey Hinton. Neural networks for machine learning, coursera. *URL: http://coursera. org/course/neuralnets*, 2012.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡1mb model size. *CoRR*, abs/1602.07360, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In *NIPS*, 2017.

Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pp. 345–353, 2017.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018.

Asit K. Mishra, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr. Wrpn: Wide reduced-precision networks. *CoRR*, abs/1709.01134, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *AAAI*, pp. 2625–2631, 2017.

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016.