Binary Quantizer

Vahid Partovi Nia — Huawei Technologies, Canada
Mouloud Belbahri — Huawei Technologies, Canada

Abstract

One-bit quantization is a general tool to execute a complex model, such as deep neural networks, on a device with limited resources, such as cell phones. Naively compressing weights into one bit yields an extensive accuracy loss. One-bit models, therefore, require careful re-training. Here we introduce a class functions devised to be used as a regularizer for re-training one-bit models. Using a regularization function, specifically devised for binary quantization, avoids heuristic touch of the optimization scheme and saves considerable coding effort.

1 Introduction

Automating complex tasks often requires complex models with too many parameters. Deep neural networks proved to be a general tool for automating object detection, image classification, speech recognition, and recently in machine translation, and text generation. An edge device such as cell phone, smart watch, smart wearable, autonomous vehicle, wireless base station, etc have limited power which restricts the use of such complex models. One-bit quantization can be regarded as a general method for overcoming the edge device implementation challenge. However, naively compressing full-precision 32-bit parameters to only one bit, destroys the model accuracy.

Neural networks include two main components: i) weights ii) activation function. Both components are computed in full-precision 32-bits most of the time. One-bit quantized version of the weights is implemented by taking the sign of the weights. One bit version of the activation function is implemented by replacing the activation $\sigma(x)$ with the sign function. Here we focus on providing a general methodology for quantizing the weights.

There are mainly two approaches for weight quantization: i) quantizing after full-precision training ii) quantizing during training. The first approach often leads to an extensive accuracy degradation. We focus on the second approach and demonstrate how to quantize weights properly using back propagation.

We propose adding a regularization function to the loss function and deploy the back propagation to tune the weights appropriately around binary values as the epochs evolve. The end result of a proper regularization scheme yields quantized weights with minimal accuracy loss.

Courbariaux et al. (2015) introduced *BinaryConnect* that trains deep neural networks with binary weights restricted on $\{-1,+1\}$. The weights are quantized using the sign function. The forward pass uses the binary weights, and the back propagation step uses the full-precision weights, but the gradient is evaluated at the binarized weight. Hubara et al. (2016) early after introduced *BinaryNet* which quantizes both weights and activations. Most of binary networks achieve $24\times$ to $32\times$ compression rate. A careful implementation often provides $4\times$ to $16\times$ inference speed up, depending on the hardware, and saves extensive memory and hardware resource.

Rastegari et al. (2016) introduced *XNORNet*, and proposed to add a scaling factor to binary weights as a trade-off between binary quantization and full-precision computation. While training, the binarized weights and the scaling factors are estimated simultaneously. Zhou et al. (2016) introduced *DoReFaNet* and generalized XNORNet, approximating the full-precision weights with more than one bit. Lin et al. (2017) introduced *ABCNet* and show how to train a binary network by adding a regularization function. This work formalizes Tang et al. (2017) for binary networks, but generalizes it for scaling factors. We provide a coherent framework to construct variety of regularization functions using a certain class of base functions.

Binary quantization is a general method to save computation, memory, and energy simultaneously. Each weight is stored in only one bit, so a complex network with million parameters like AlexNet (Krizhevsky et al., 2012a) can be compressed from 250MB (32-bit version) to 7MB (one-bit version). Neuronal computation can be decomposed into two major steps

1. Linear combination computation $\mathbf{x}^\top\mathbf{w} = \sum_n x_n w_n$ where x_n is the output of previous layer and w_n is the neuron weight and n indexes the neuron in the layer.

2. Nonlinear activation computation $\sigma(\mathbf{x}^\top\mathbf{w})$.

In 1., for binary $\mathbf{x} \in \{-1,+1\}$ and $\mathbf{w} \in \{-1,+1\}$, the multiplication operator $x_n w_n \in \{-1,+1\}$ is equivalent to the XNOR operation. The sum $\mathbf{x}^\top\mathbf{w} = \sum_n x_n w_n$ is, therefore, reduced to count.

This computation simplification allows to compute large networks fast and efficiently on a low-resource device using *logical operation* and *count instruction*. However, the main challenge is to re-train binary weights so that the same network with binary weights still performs with a reasonable accuracy.

2 Regularization

Back propagation training includes a loss function $\mathcal{L}(\mathbf{w})$ combined with a regularization $R(\mathbf{w})$. Regularization is not only included to improve model accuracy, but is required to maintain stable numerical computation. The loss function $\mathcal{L}(\mathbf{w})$ is often the squared error for continuous output and entropy or hinge loss for discrete output. The back propagation ultimately optimizes

$$\mathcal{L}(\mathbf{w}) + \lambda R(\mathbf{w}),$$

where λ is the regularization constant, usually tuned using cross-validation.

The loss function is often regularized by ℓ_1 norm called the *lasso* (Tibshirani, 1996), ℓ_2 norm called the *ridge* (Hoerl and Kennard, 1970), or a linear combination of these two called the *elastic net* (Zou and Hastie, 2005)

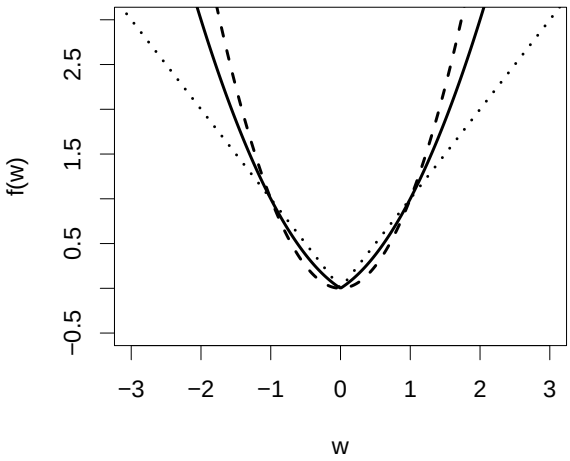

Fig. 1: Examples of several convex functions. The ℓ_1 norm (1) dotted curve , ℓ_2 norm (2) dashed curve, and the elastic net (3) solid curve.

$$f(w) = |w|, \quad (1)$$
$$f(w) = w^2, \quad (2)$$
$$f(w) = \delta|w| + (1-\delta)w^2. \quad (3)$$

However, all regularization techniques are not explicitly defined as above. Implicit regularization such as early stopping, drop out, data augmentation, model averaging, etc, are employed along with explicit regularization in practice. Common explicit regularization functions (1), (2), and (3) encourage training weights about the origin. In binary training, unlike full-precision, weights must concentrate around -1 and $+1$.

3 Quantizer Construction

Common regularization functions encourage weights to remain about the origin. It makes more sense for binary networks that regularization encourages weights about -1 and $+1$, suitable for binary quantization. Here we formalize a more general regularization function to encourage weights about $\alpha \times \{-1,+1\}$. So it also includes the scaling factor α. This allows learning the scaling factor α by only adding another equation to back propagation for α. Using such regularizers in binary training provides some advantages compared to the other heuristic training techniques

1. Back propagation is automatically modified, so no subjective optimization heuristics is necessary. The regularization function carries the subjective part, but back propagation remain the same.
2. Scaling factor training is embedded within back propagation.
3. Any neural architecture can be compressed and quantized with any of these regularization functions. However, some architectures over some data sets are quantized more effectively with certain class of these regularization functions.
4. Binary training requires little implementation effort, as back propagation runs only with a slightly modified objective function.

Suppose $f(w)$ is a quasiconvex function. The class of quasiconvex functions is larger than convex, i.e. all convex functions are quasiconvex, but the converse may bot be true. By the definition of a quasiconvex function the area below of a section of a quasiconvex function is a convex set, i.e. $f(w)$ is quasiconvex if $\forall a \in \mathbb{R}, A = \{w \mid f(w) < a\}$ is a convex set. Equivalently, a function is quasiconvex if for any two values w_1, w_2

$$f(\lambda w_1 + (1-\lambda) w_2) < \max\{f(w_1), f(w_2)\},$$

see Fig. 2 for few examples. The quasiconvex functions of Fig. 2,

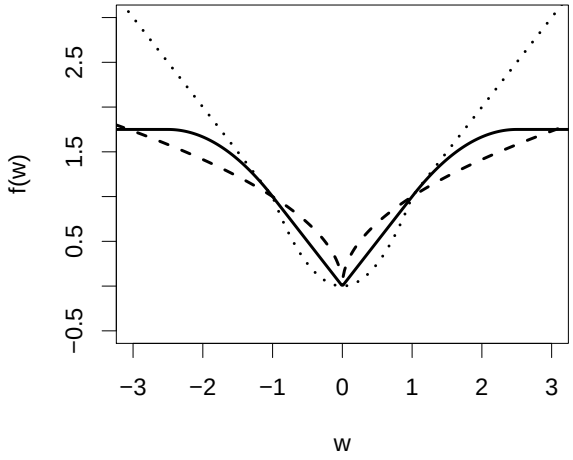

Fig. 2: Examples of several quasiconvex functions. The scad function (Fan and Li, 2001) with linear a core and constant tails (solid curve), with a quadratic core and linear tails (dotted curve), with a squared root core and tails (dashed curve).

like the convex functions of Fig. 1, encourage weights about the origin. However, binary quantization requires weights to be trained about $\{-1,+1\}$ (Courbariaux et al., 2015). Fully binary weights cannot achieve the accuracy of full-precision networks. A compensation is made by adding a scaling factor α, so the weights are encouraged about $\{-\alpha,+\alpha\}$. Here we recommend the following modification to construct a convenient regularizer.

Suppose $f(w)$ is a quasiconvex function

1. centered at the origin, i.e. $f(0) = 0$,
2. symmetric about zero, i.e. $f(w) = f(-w)$.

Define the f-positive part $\overset{+}{f}(w)$

$$\overset{+}{f}(w) = f(w)\mathbb{I}_{[0,\infty]}(w),$$

where $\mathbb{I}_A(w)$ is the indicator function on the set A. For an $\alpha > 0$ define the inverse image $\overset{+}{f}^{-1}(\alpha)$, where $\overset{+}{f}^{-1}(w)$ is the inverse of $f(w)$ for $w > 0$. Belbahri et al. (2019) defines a class of quasiconvex functions equivalent to Huber-M convex function (Huber et al., 1964), an attractive qausiconvex base $f(w)$.

Here we show only two ways (out of many) for constructing a binary quantizer

$$R(w) = f(w - \alpha\,\mathrm{sign}(w)) \quad (4)$$

$$R(w) = |f(w) - \overset{+}{f}^{-1}(\alpha)|^p \quad (5)$$

These regularizers encourage weights about $\{-\alpha,+\alpha\}$. A regularization function always shrinks weights towards $\{-\alpha,+\alpha\}$, but sets weights exactly to these values if the regularization function is non-differentiable at the minimum (Fan and Li, 2001). Setting values exactly to $\{-\alpha,+\alpha\}$ is the target of binary training, therefore non-differentiable functions at $\{-\alpha,+\alpha\}$ are more of interest.

The quantizer (5) is always non-differentiable at $\{-\alpha,+\alpha\}$ for $p < 2$, and is always differentiable for $p = 2$, but regularizer (4) inherits its differentiation behaviour at minimum from the base function, see Fig. 3.

Training scheme of Rastegari et al. (2016) improvises the regularization (5) for $p = 2$ implicitly. Regularization functions proposed in Darabi et al. (2019) are a special case of (5) with $f(w) = |w|$.

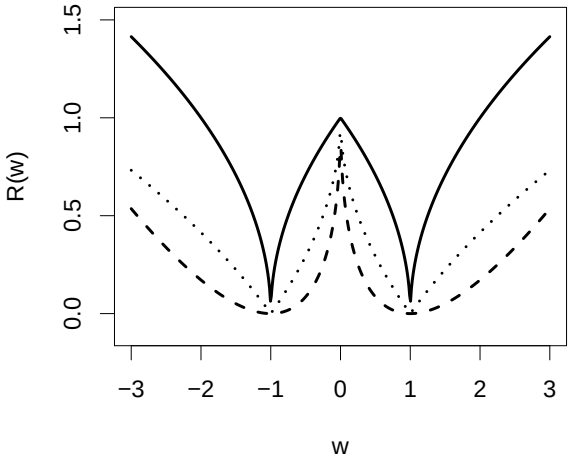

Fig. 3: Examples of several binary quantizers using the base function $f(w) = \sqrt{|w|}$, to create three different regularizers $R(w)$. The solid curve, is the regularizer of Equation (4) , dashed curve is (5) with $p = 2$, and dotted curve is (5) with $p = 1$.

4 Binary Training

Training a neural network with binary values is straightforward with binary quantizer regularization. It is only required to define $R(\mathbf{w}) = \sum_l \sum_n R(w_{ln})$, where l denotes the layers, and n denotes the neurons per layer to quantize all w_{ln}. The weights that are absent in the regularization term will remain unquantized.

The training objective function $\mathcal{L}(\mathbf{w}) + \lambda R(\mathbf{w})$ quantizes the weights towards $\{-\alpha,+\alpha\}$ for a large λ. The value of λ plays an important role in properly training the network. Our experiments show starting training with $\lambda = 0$ (no regularization) and increasing λ with logarithmic rate with epochs will train the network faster. Formally, we suggest $\lambda \approx \varepsilon\eta\log(t)$ where ε is a small positive scalar, η is the learning rate, and t is the number of epochs. Fig. 4 shows an example of training weight distribution as the number of epochs evolve. The number of scaling factors plays a key role in competing with full-precision networks. Too many scaling factors yields slow inference implementation, but closes the gap with full-precision networks. No scaling factor (setting $\alpha = 1$) provides easy XNOR implementation, but may lead to a large accuracy loss. Vanishing scaling factor (setting $\alpha = 0$) dismisses the neuron and is equivalent to pruning. A scaling factor per layer for a fully-connected layer, and a scaling factor per filter for a convolutional layer is a good compromise.

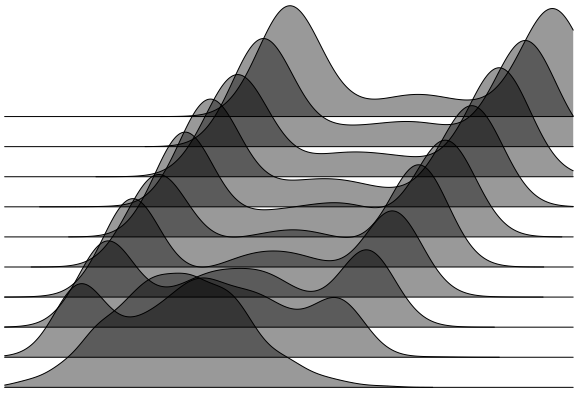

Fig. 4: Weight distribution evolution as the number of epochs increases. Weights are pushed away from zero and concentrate more about $\{-\alpha, +\alpha\}$ as training goes on.

Table 1: Top-1 Accuracy on CIFAR-10 for several binary quantizers (BQ), quantizing AlexNet and VGG architectures.

Model	AlexNet	VGG
BQ $R_1^{(2)}$	86.72%	90.73%
BQ $R_{1.5}^{(2)}$	86.37%	90.63%
BQ $R_2^{(2)}$	86.82%	90.82%
BQ $R_{1,50}^{(1)}$	86.70%	90.94%
BQ $R_{2,50}^{(1)}$	86.85%	**91.07%**
BQ $R_{0.1,10}^{(1)}$	86.93%	90.83%
BQ $R_{0.1,100}^{(1)}$	**87.21%**	91.00%
BQ $R_{0.5,10}^{(1)}$	86.97%	90.86%
BQ $R_{0.5,100}^{(1)}$	87.00%	90.85%
Full-Precision	88.58%	91.72%

Table 2: Top-1 Accuracy on ImageNet for several binary quantizers (BQ), using on AlexNet architecture, compered to BinaryNet and XNORNet.

Model	Fully Binary	AlexNet
BQ $R_1^{(2)}$	Yes	41.4%
BQ $R_{1.5}^{(2)}$	Yes	41.3%
BQ $R_{0.1,100}^{(1)}$	Yes	42.4%
BQ $R_{0.1,100}^{(1)}$	No	**44.2%**
BinaryNet	No	27.9%
XNORNet	No	**44.2%**
Full-Precision	No	57.1%

5 Application

The CIFAR-10 image dataset (Krizhevsky and Hinton, 2009) contains 60000 labeled for 10 classes 32×32 images separated into 50000 for training set and 10000 for test set. The ImageNet dataset (Russakovsky et al., 2015) consists of $\sim$ 1.2M training images, and 1000 classes. The images are resized to 256×256 and then are randomly cropped to 224×224. We investigate the influence of the regularization functions on training binary networks for these two datasets. First, we consider the binary quantizer (4) with the quasiconvex base function (6) from Belbahri et al. (2019)

$$f_{\gamma,\beta}(w) = \gamma w \tanh\left(\frac{\beta w}{2}\right), \tag{6}$$

where $\tanh(.)$ is the hyperbolic tangent function, $\gamma > 0$ is a shape parameter and $\beta > 0$ is a scale parameter. We test several (γ, β) combinations. Then, we take the absolute value function $f(w) = |w|$ as a base for (5) with $p = 1$, $p = 1.5$ and $p = 2$. The resulting regularization functions are

$$R_{\gamma,\beta}^{(1)}(w) = f_{\gamma,\beta}(w - \alpha\,\mathrm{sign}(w)), \tag{7}$$

$$R_p^{(2)}(w) = ||w| - \alpha|^p. \tag{8}$$

Somehow the quasiconvex base (7) is the analog of the convex base (8), but the former is more flexible.

We train two different architectures AlexNet (Krizhevsky et al., 2012b) and VGG (Simonyan and Zisserman, 2014) using the ADAM optimizer (Kingma and Ba, 2014) on CIFAR-10. We also quantize AlexNet on ImageNet data. The goal of this study is to show that our binary quantizer formalizes XNORNet (Rastegari et al., 2016). The XNORNet architecture is similar to AlexNet, so this is a fair comparison. The results are summarized in Table 1 and Table 2.

Top-1 accuracy of AlexNet and VGG full-precision models on CIFAR-10 are 88.58% and 91.72% respectively. Using the binary quantizers (BQ), the accuracy loss is minimal. Quantizing the network using (8) helps the binary network's training. The flexibility of the regularization function (7) helps learning through proper adaptation of the objective function, and thus the network to achieve accuracy close to its full-precision counterpart. Indeed, the loss accuracy is 1.37% and 0.65% for AlexNet and VGG respectively. Ideally, the parameters γ and β would be learned with back propagation, and different values can be used for different layers (or even filters for convolution layers).

Results on ImageNet dataset is summarized in Table 2. We compare our strategy for different regularization functions to other binary neural networks: BinaryNet (Hubara et al., 2016) and XNORNet (Rastegari et al., 2016) on AlexNet architecture. Obviously, since BinaryNet does not use the scaling factor, we see that the accuracy loss is much higher than that of XNORNet and binary quantizer (BQ). Training ImageNet is slow, thus we decided not to fine-tune the hyper-parameters, and only choose a learning rate, a regularization constant ascent rate of λ. We launch three experiments with the regularization functions (8) and (7) for 100 epochs. Once again, the flexibility of (7) allows to obtain a better generalization of the learned binary network compared to (8). The results are inferior but in the same order of magnitude as those published in Rastegari et al. (2016), because we quantize the whole network while XNORNet leaves the first and last layers unquantized. To scale our method with XNORNet we left the first and last layers unquantized, the accuracy reaches to 44.2% the same as XNORNet. We believe that fine-tuning thank to the binary quantizers flexibility allows to beat the existing quantization methods on various architectures.

6 Conclusion

Binary networks are suitable for edge devices, which deal with computation resource, inference speed, memory compression, and low-power consumption constraints. However, such networks are difficult to train. Back propagation is suitable for full-precision values and taking the sign of the weights of the full-precision is often too naive and leads to a great accuracy loss. Recently authors started modifying back propagation to improve training binary networks, but still there is a big gap between accuracy of binary networks with full-precision networks. Adding a scaling factor resolves this problem by compromising between binary and full-precision, but estimating the scaling factor requires another training strategy.

Here we showed how to construct a regularization function that suites binary training with an embedded scaling factor. This approach modifies back propagation with minimal effort and scaling factor estimation is embedded in back propagation. As the weights train, the scaling factors are estimated along with weights from their own updating equation. Our numerical results show this approach quantizes various architectures successfully.

References

Belbahri, M., S. Darabi, and V. Partovi Nia
2019. Foothill regularizer. Submitted.

Courbariaux, M., Y. Bengio, and J.-P. David
2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, Pp. 3123–3131.

Darabi, S., M. Belbahri, M. Courbariaux, and V. Partovi Nia
2019. BNN+: Improved binary network training. Submitted.

Fan, J. and R. Li
2001. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360.

Hoerl, A. E. and R. W. Kennard
1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio
2016. Binarized neural networks. In *Advances in neural information processing systems*, Pp. 4107–4115.

Huber, P. J. et al.
1964. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101.

Kingma, D. P. and J. Ba
2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

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

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

Lin, X., C. Zhao, and W. Pan
2017. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, Pp. 345–353.

Rastegari, M., V. Ordonez, J. Redmon, and A. Farhadi
2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, Pp. 525–542. Springer.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al.
2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

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

Tang, W., G. Hua, and L. Wang
2017. How to train a compact binary neural network with high accuracy? In *AAAI*, Pp. 2625–2631.

Tibshirani, R.
1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, Pp. 267–288.

Zhou, S., Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou
2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.

Zou, H. and T. Hastie
2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, 67(2):301–320.