Uncertainty-based quantization method for stable training of binary neural networks

A.V. Trusov [1,2,3], D.N. Putintsev [2,3], E.E. Limonova [2,3]
[1] Moscow Institute of Physics and Technology (National Research University), 141701, Russia, Dolgoprudnii, Institutskiy per. 9;
[2] Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, 19333, Russia, Moscow, Vavilova str. 44, corp. 2;
[3] LLC "Smart Engines Service", 117312, Russia, Moscow, prospect 60-letiya Oktyabrya 9

Abstract

Binary neural networks (BNNs) have gained attention due to their computational efficiency. However, training BNNs has proven to be challenging. Existing algorithms either fail to produce stable and high-quality results or are overly complex for practical use. In this paper, we introduce a novel quantizer called UBQ (Uncertainty-based quantizer) for BNNs, which combines the advantages of existing methods, resulting in stable training and high-quality BNNs even with a low number of trainable parameters. We also propose a training method involving gradual network freezing and batch normalization replacement, facilitating a smooth transition from training mode to execution mode for BNNs.

To evaluate UBQ, we conducted experiments on the MNIST and CIFAR-10 datasets and compared our method to existing algorithms. The results demonstrate that UBQ outperforms previous methods for smaller networks and achieves comparable results for larger networks.

Keywords: binary networks, neural networks training, quantization, gradient estimation, approximation.

Citation: Trusov AV, Putintsev DN, Limonova EE. Uncertainty-based quantization method for stable training of binary neural networks. Computer Optics 2024; 48(4): 573-581. DOI: 10.18287/2412-6179-CO-1427.

Introduction

Nowadays, artificial neural networks (ANNs) are widely used in various applications. They are ranging from large-scale language models [1] and image-generation networks [2] with billions of parameters that require specialized servers, to small and fast networks for character recognition [3] designed to run on mobile and embedded devices. The latter case, involving small networks on edge devices, is particularly interesting for practical applications. For instance, a real-time document recognition system running on a mobile phone [4] may utilize ANNs to compute patch descriptors for template-matching [5], localize and recognize text characters [6], and perform other small tasks. In order for the system to work effectively on a mobile phone, the networks must be small enough to avoid RAM overflow during execution and prevent an undesired increase in the size of the mobile application. Additionally, they must be fast enough to meet the real-time operation requirements. Other systems that require fast and lightweight ANNs include autonomous driving [7], internet of things (IoT) applications [8], and even medical diagnostics [9].

One approach to making an ANN faster and smaller in terms of memory involves replacing its 32-bit weights with discrete integer values, a process known as network quantization [10]. The extreme form of quantization is binarization, which provides binary neural networks (BNNs), where the only possible values for weights and activations (layer inputs) are -1 and 1. In this case, a network occupies 32 times less memory than its real-valued counterpart. Moreover, BNNs allow for the computation of dot products, which are the main building blocks of matrix multiplications and convolutions (which are the most computationally expensive operations in modern ANNs), using simple bitwise operations, bit-counts, and additions [11]. These operations are significantly simpler and faster from a hardware perspective compared to multiplications or multiply-accumulate operations used in real-valued networks. As a result, BNNs can be efficiently implemented on various computing devices such as FPGAs [12], GPUs [13], and CPUs [14]. BNNs can be applied to many tasks, which rely on computationally-demanding ANNs, including but not limited to: image recognition [11], semantic segmentation [15], and speech recognition [16].

It is worth noting that BNNs are not the only multiplication-free networks. Ternary neural networks, where weights and activations are constrained to the set $\{-1, 0, 1\}$, can also be efficiently implemented using bitwise operations [17]. There are variations of binary and ternary neural networks where only weights are quantized and the activations remain floating-point [18, 19]. There are also variations where inputs and weights are binary for each layer of the network, but a floating-point skip-connection exists around the layer [20].

Finally, there are neural networks with completely different models of neurons, such as AdderNet [21] that computes L_1-distance between the input and the weights instead of dot-product, and a Bipolar morphological network that computes sums and maximums in the inner-most loop of matrix multiplication and may use fast approximations of non-linear functions like exponent and logarithm [22]. However, all of these methods are less computationally or memory-efficient than pure BNNs. Therefore, this paper is dedicated to the training of pure BNNs.

Training Binary Neural Networks (BNNs) and other few-bit quantized networks is a more challenging task compared to training floating-point networks. The main difficulty arises from the fact that the standard gradient descent optimization method, based on error back-propagation, is not directly applicable due to the nature of the binarization function (sign function, which has zero derivative at each point except for zero, where it is non-differentiable). To address this problem, the common solution is to use the straight through estimator (STE), which involves computing the sign function as is during the forward pass and using a piecewise differentiable approximation (e.g., $hardtanh(x) = \min(1, \max(-1, x))$ function) during the backward pass to compute gradients [23]. This simple solution has been shown to work surprisingly well when training BNNs from scratch [24].

Unfortunately, STE method has its own limitations. It leads to weights oscillation around zero, resulting in frequent sign changes and slower and less stable training [25]. It also causes gradient mismatch, when the gradients in the backward pass disagree with function in the forward one [26]. In contrast to STE, quantization methods, based on smooth approximations of the sign function [27] (e.g. $\tanh(vx) \approx \operatorname{sign}(x)$, if $v \to \infty$ [28]), do not suffer from gradient mismatch. These methods provide a way to convert floating-point model to binary smoothly by gradually adjusting of approximation accuracy. Such training is usually faster and more stable compared to STE. However, it introduces a gap between the binary model and its smooth approximation, leading to optimization objective mismatch [29]. While there are more complex approaches to training BNNs, such as minimizing quantization error [11] or using Bayesian learning rules [30], this paper focuses primarily on STE and smooth quantization due to their ease of implementation and satisfactory results in terms of quality (in comparison to other methods) [24, 28].

Despite the existence of various training methods for BNNs, there is still a significant accuracy gap between full precision (floating-point) ANNs and BNNs with the same architecture. This is why BNNs are mainly studied in academic settings, unlike 8-bit quantization, which is widely used in practice [31] and natively implemented in popular machine-learning frameworks like PyTorch [32] or Keras [33]. However, recent studies have shown that by adjusting the network structure, BNNs can outperform ANNs under the same computational budget [34]. This means that a more accurate BNN with more parameters and operations can fit into the same computational budget as its floating-point counterpart. This makes BNNs interesting not only from a theoretical perspective but also from a practical standpoint, especially for devices that do not support floating-point multiplications. That why is stable and accurate algorithms for BNN training are of great interest.

In this paper we propose a new method for training BNNs from scratch. Our method can be viewed as an extension of smooth quantization methods but it has several important differences:

- It relies on uncertainty-based activations (see Section 3.1), which provide a more accurate smooth approximation of a sign function and helps us to improve the performance of the BNNs.
- It uses stochastic binarization with STE as a regularization technique, which helps us to minimize the mismatch between the BNN and its approximation, and leads to better performance.
- During training, it replaces the standard batch normalization [35] technique with a simpler intermediate module. It simplifies the transition to binary inference, where there is no batch normalization at all.

To evaluate our proposed model, we trained three small Convolutional Neural Networks (CNNs) on the MNIST dataset [36] and three larger CNNs from the VGG family [37] on the CIFAR-10 dataset [38]. We compared our training method with STE [39] and smooth self-binarization methods [28] empirically. The results of our experiments confirm that our method combines the strengths of both approaches: the stable training of smooth binarization and the direct transition to binary inference of the STE-based methods. Our method also achieves higher accuracy compared to STE, especially on models with a small number of parameters.

1. Preliminaries

Our method of stable training binary neural network is majorly inspired by commonly-used Straight Through Estimator [39] and soft quantization methods [28, 29]. However, before going into the details let us first formalize what a BNN is, what the *quantizers* are and how they allow for training of BNNs.

1.1. BNN

As we mentioned above, a binary neural network is an ANN, in which weights and activations (layer inputs) are binary (i.e. belong to the set $\{-1, 1\}$). Such networks consists of binary layers. Each binary layer takes a binary vector as an input, computes dot-products with binary weights using bit-wise operations and sums the intermediate results in integer accumulators. After that, those integer results are binarized using sign function and passed to the next binary layer.

It is important to note, that the input and the output of a network can be non-binary vectors. In the image classification task, an input is an image, which is usually interpreted as a real-valued array, and the output is real-valued class confidences. For example, to build such a network we can binarize the input by a certain threshold for the first layer, and use integer results of binary dot-product as an input of softmax function in the last layer. However, the first and the last layer are usually easy-to-compute, because there are few channels in the input, in comparison to the rest of the network. That is why they are usually, preserved in real-valued form (not binarized). It allows for higher accuracy and simpler training, at the cost of negligible performance overhead [11]. Such networks with almost all binary layers are usually also referred to as BNNs.

1.2. BNN inference

Let us start with a description of computations in already trained BNN.

The core operation in a linear layer, whether it is fully-connected or convolutional, is a dot product which can be represented as:

$$z = \sum_{i=1}^{N} x_i w_i, \tag{1}$$

where x_i and w_i are the input and weight values respectively, and z is an output, which will further undergo non-linear activation (binarization for BNNs).

In the case of a binary neural network, the dot products can be computed using XNOR (exclusive NOR) and bitcount operations [11]. The XNOR operation compares each bit of the input and weight values, returning 1 if they are both 1 or both 0, and 0 otherwise. Thus,

$$\tilde{x}_i \oplus \tilde{w}_i = 1 \Leftrightarrow x_i w_i = 1,$$

where $\tilde{x}_i$ and $\tilde{w}_i$ are the binary encodings of input x_i and weight w_i (e.g. $\tilde{x}_i = 2 - x_i$, since $x_i \in \{-1,1\}$).

The bitcount operation counts the number of 1 bits. Therefore, equation (1) can be rewritten as:

$$z = 2 \cdot \text{bitcount}\bigoplus(\tilde{\mathbf{x}}, \tilde{\mathbf{w}}) - N, \tag{2}$$

where $\oplus$ stands for bit-wise XNOR operation over binary vectors $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{w}}$, and N is the number of elements in those vectors.

After the dot product is computed, the result (integer value z) is binarized according to a certain threshold b, which can be seen as an addition of an integer bias, followed by binarization with the sign function. Thus, the output value (y) of a binary layer can be computed as:

$$y = \begin{cases} 1, & \text{if } 2 \cdot \bigoplus(\tilde{\mathbf{x}}, \tilde{\mathbf{w}}) - b - N \geq 0 \\ -1, & \text{otherwise} \end{cases}. \tag{3}$$

Having the equation for the dot product, we can compute matrix multiplications in fully-connected layers and convolutions in convolutional layers. With some additional tricks (special reordering, 16-bit accumulators for sums, vector operations, etc.), they can be efficiently implemented on different computing devices [12 – 14]. We will keep in mind the implementation [40] and remember that a layer in the trained BNN should have binary input, binary weights, integer bias, and no floating-point operations. Fig. 1*a* demonstrates how trained BNN layer works.

1.3. Quantizers and BNN training

The equation (3) involves operations over discrete values. So, in that formulation, a BNN cannot be trained using gradient descent. That is why during training, quantizers are required. Quantizers are the modules of a neural network that map real-valued input to the discrete quantized set. They can have their own adjustable parameters [41].

In this paper, we will consider quantizers in a broader sense: a quantizer is a module of a neural network that approximates quantization (in our case, binarization) operation and allows for error backpropagation.

Another useful module for neural network training is batch normalization [35]. Let us remind that it is a linear transformation of a vector:

$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} \cdot \gamma + \beta, \tag{4}$$

where $\hat{x}_i$ is the normalized value of the i-th element of the input vector x, μ and σ^2 are estimated mean and variance, γ and β are trainable parameters, and ε is a small constant added for numerical stability.

The combination of quantizers and batch normalization allows for the creation of a trainable BNN block, as shown in Fig. 1*b*. Now let us show how quantizers work in different BNN training algorithms.

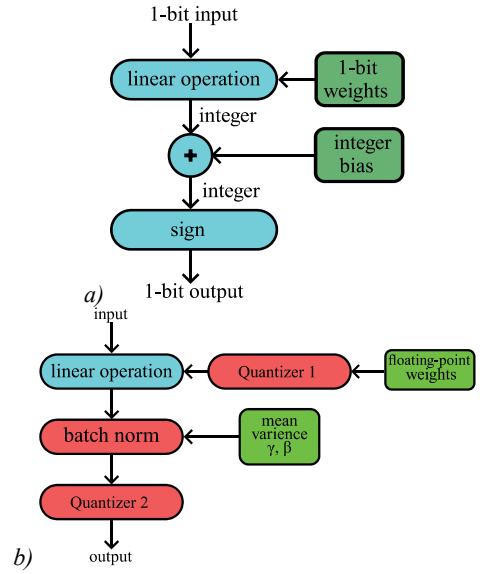

Fig. 1. BNN layer structure in the execution and training modes. (a) Execution mode, (b) training mode

1.3.1. STE

The most commonly used quantizer is STE. It computes the sign function in the forward pass:

$$y = \mathrm{STE}(x) = \mathrm{sign}(x) = \begin{cases} 1, & \text{if } x \ge 0, \\ -1, & \text{otherwise.} \end{cases} \quad (5)$$

Here and later in the article, we consider $\mathrm{sign}(0) = 1$ (not 0), so that the output is binary. The forward pass described above is used in modern BNNs [24, 42], while earlier versions used a more complex stochastic scheme involving sampling from a Bernoulli distribution:

$$y = \mathrm{STE}_{\mathrm{old}}(x) = \\ = \begin{cases} 1, & \text{if with probability } \mathrm{hardsigmoid}(x) \\ -1, & \text{otherwise} \end{cases}, \quad (6)$$

where $\mathrm{hardsigmoid}(x) = \max(0, \min(1, (x+1)/2))$ [18, 39]. Although this complication does not seem to be necessary for BNNs to achieve high quality, we will use a similar approach as a regularization (see Section 2.2).

In the backward path, the gradient is computed as if in the forward pass there was a piecewise differentiable function, such as hardtanh [24, 39], AproxSign [20], or any other appropriate approximation [42] of sign.

The major strengths of STE quantizer are the ease of its implementation, direct correspondence of a forward pass to that of BNN in evaluation mode (linear operation over binary values), and the fact that it achieves acceptable quality [24]. Its main weaknesses are unstable training caused by frequent sign changes of weights and the gradient mismatch [25].

1.3.2. Self binarizing quantizer

The self-binarizing quantizer (SBQ) was proposed by Lahoud et al. [28] to achieve smooth quantization of BNN.

In the forward pass, the SBQ computes the function

$$y = \mathrm{SBQ}(x) = \tanh(vx), \quad (7)$$

where v is a hyperparameter that controls the approximation of the sign function. By increasing the value of v, the output of the SBQ approximates the sign function. During training, v is gradually increased.

In the backward pass, the gradient is computed using the derivative of $\tanh(vx)$, which is $v(1-\tanh^2(vx))$. So, there is no gradient mismatch like in STE. Moreover, the weights do not oscillate around zero and the training is more stable.

Unfortunately, there is no direct correspondence between the training and evaluation mode with SBQ quantizer [29], so there is a gap in quality. Also, there is a mismatch of optimization objectives in the early (when v is low) and late (when v is large) stages of training, so the resulting quality might be lower than expected.

1.4. Transaction from training to evaluation

To convert a neural network from training mode to evaluation mode (see Fig. 1), one needs to apply quantization (5) to the weights. Then, Quantizer1 is no longer required. Quantizer2 is replaced by a sign function (5).

After that, the only remaining excess module is batch normalization. Some authors do not consider the presence of a few floating-point operations as a problem and preserve batch norm as is [11, 39]. Lahoud et al. [28] noticed that batch normalization followed by the sign function can be replaced with integer bias addition followed by sign multiplication by the sign of γ in equation (4). We would like to point out that the latter multiplication is also excessive and this sign can be "folded" into the signs of corresponding weights and biases, similar to the "folding" of the batch normalization layer during 8-bit quantization in the work of Benoit et al. [31]. In Section 2.3, we propose an alternative to standard batch normalization simplifying the transformation even further.

Thus, the transformation of a BNN layer from training mode to integer-only inference is completed.

2. Uncertainty based quantization

To overcome the weaknesses and combine strengths od STE and SBQ quantizers, we propose our own quantizer and BNN trainig procedure which are described in this section.

2.1. Uncertainty-based quantizer

The key feature of our quantized is uncertainty-based activation:

$$\phi(x,u) = \begin{cases} \tanh\left(\frac{x}{u+\varepsilon}\right), & \text{if } u \ge \tau, \\ \mathrm{sign}(x), & \text{otherwise,} \end{cases} \quad (8)$$

where the real value $u \in [0,1]$ denotes uncertainty of the input x, τ is a threshold at which smooth quantization switches to a hard one (we use $\tau = 10^{-5}$), and ε is a small enough value added for numerical stability (we set $\varepsilon = 10^{-7}$). If uncertainty u is below the threshold τ, the error gradient is no longer propagated through this activation (which may happen only for a part of the input vector, since $\phi(x,u)$ is an element-wise operation).

It may seem that we reinvented SBQ with $v = (u+\varepsilon)^{-1}$ (see eq. (7)), and with an additional threshold τ, but the major difference is how we compute u. This process varies for weight and activation quantizers (Quantizers 1 and 2 in Fig. 2 respectively).

We define the uncertainty of a real variable $t \in [0,1]$ as $u_t = 1 - t^2$. If $t = \pm 1$, we are certain of its value ($u = 0$), and if t is close to zero, uncertainty is close to one – we are not certain, if it should be -1 or $+1$ in a trained BNN. For a sum of variables

$$T = \sum_{i=1}^{N} t_i,$$

uncertainty is defined as the mean uncertainty of the variables:

$$u_T = \sum_{i=1}^{N} u(t_i) / N = 1 - \frac{1}{N} \sum_{i=1}^{N} t_i^2 .$$

Thus, we can compute uncertainty for a dot-product in BNN (1):

$$u_z = 1 - \frac{1}{N} \sum_{i=1}^{N} x_i^2 w_i^2 . \quad (9)$$

In the case when only weights of a layer are binary (for example, in the input layer with non-binary the input):

$$u_z = 1 - \frac{1}{N} \sum_{i=1}^{N} w_i^2 . \quad (10)$$

We call the latter case BWN (Binary Weight Network) mode.

Now, for a BNN layer to work, we need weights and uncertainties for the weight quantizer. To that end, we denote inner (hidden) vector of weights as $\mathbf{v}$ (trainable parameters); and inner (hidden) vector of uncertainties as υ (they are sampled from a normal distribution $\mathcal{N}(0,1)$); these values do not change during training; η is single adjustable value per layer that controls the uncertainty. Then, the uncertainties of weights are computed as:

$$u_{w_i} = \sigma(\upsilon_i + \eta), \quad (11)$$

where $\sigma(\cdot)$ denotes the sigmoid function: $\sigma(t) = e^t / (e^t + 1)$.

Thus, the forward pass through BNN with our quantizer works as follows:

1. Compute weight uncertainties $u_w(\upsilon, \eta)$ using eq. (11).
2. Compute weights using quantizer $w = \phi(v, u_w)$ using eq. (8).
3. Compute dot-products in matrix multiplication or convolution $z(x, w)$ using eq. (1).
4. Compute batch normalization $\hat{z}(z)$ using eq. (4).
5. Compute uncertainty $u_z(x, w)$ using eq. (9) or (10).
6. Finally, compute output $y = \phi(\hat{z}, u_z)$ using (8).

Usually, there are multiple layers in a neural network. Thus, there are many adjustable parameters η. We initially set their values to 8, so that u_w is very close to 1. Then we gradually (linearly) decrease them during training. When the value of η in the layer reaches -12, the layer is considered "frozen", and it is converted to evaluation mode. Moreover, we set the "freezing" speed of different layers so that the upper layers of the network are "frozen" prior to the bottom layers and so that the bottom layers have some time to adapt. This is similar to the sequential quantization of a neural network demonstrated in [43].

2.2. STE as regularization

One possible disadvantage of SBQ and our proposed Uncertainty Based Quantizer (UBQ) is that during training, some weights and activations may appear to be close to zero. The network may adapt so that those values are expected to be close to zero and during binarization, the quality will drop. To prevent this from happening, we introduce STE-regularization.

After the output of a quantizer $y = \phi(x, u)$ is computed according to (8), a fixed part $p \in [0, 1]$ of its output is randomly binarized according to (6). In the backward pass, the binarization is completely ignored, and the network is trained as if quantization was smooth. It means that the $1-p$ part of weights is trained using smooth quantization, while the p part is trained using STE.

This approach can be viewed as an analog of dropout regularization [44], as it serves the same goal – preventing a network from adapting to certain features. Here, we use the term "regularization" in a broad sense – as a method of preventing a network from overfitting (not only to the traning data but to hidden representation as well). As dropout regularization prevents complex co-adaptations of neurons in ANNs, our regularization prevents their adaptation to close-to-zero inputs, which only exist in the training stage.

2.3. Alternative normalization

As we mentioned above, binarizing the batch normalization layer may be a bit tricky. To simplify it we introduce the following procedure.

We start with standard batch normalization (4). After a few epochs of training, we replace it with simpler normalization:

$$\hat{x}_i = \frac{x_i + b_i}{\sqrt{\kappa_i^2 + \varepsilon}} \cdot |\alpha|, \quad (12)$$

where b is fixed (not learnable) integer bias, α is a trainable floating-point parameter, ε is a small constant added for numerical stability, and finally κ^2 is a running mean for $(x+b)^2$ (which is a sort of a biased variance).

The transaction from that normalization to the trained BNN layer is extremely simple: b becomes the bias of the network, and the rest of the normalization is omitted because multiplication by positive value does not affect the sign function.

The transaction from standard batch norm to our is as follows:

$$\kappa^2 = \sigma^2, \ \alpha = |\gamma|, \ b = \text{sign}(\gamma) \left\lfloor \frac{\sqrt{\sigma^2 + \varepsilon}}{\gamma} \cdot \beta - \mu \right\rfloor$$

and the weights of the layer are element-wise multiplied by the corresponding value of sign (γ_i).

3. Experiments

We evaluated STE, SBQ, and the proposed UBQ quantizers on MNIST handwritten digits [36] and CIFAR-10 images [38] datasets. These datasets pose 10-class image classification problems. We train several neural networks to solve them and compare the recognition accuracies of the networks, trained by different methods.

In our neural networks, all the layers except the first and the last one are binary. As we mentioned in Section

1.1, it is a common practice, that does not significantly affect the performance of ANN. That is why such networks can be seen as representative examples of BNNs used in practical computer vision tasks.

Our networks take a digital image (as 3-dimentional real-valued array) as an input and compute real-valued class confidences for that image.

3.1. Neural networks

We used two families of networks in our work. First are small convolutional networks used for training on MNIST. Their architecture is presented in Fig. 2*a*. Convolutional layers *conv1* and *conv2* also have output stride 2. Convolutional layers and fully-connected layer *fc1* are binary, while the last layer has real-valued weights (a common practice for better convergence).

For CIFAR-10 we used VGG-like [37] models similar to Small VGG of LQ-Nets [41]. Their architecture is shown in Fig. 2*b*. In those models, an input of each convolutional layer is padded by one pixel to preserve size, and *pool* denotes the max-pooling layer. Once again, the last layer has real-valued weights, and convolutional layers *conv1-conv5* are binary. But, here the first (*conv0*) layer has real-valued input and binary weights (BWN mode).

It is worth mentioning that in our work both MNIST CNNs and CIFAR-10 VGGs do not have layers with both real-valued input and weights, thus they can run in multiplication-free mode (if we do not consider sign inversion a multiplication). The major amount of computations of such networks lies within convolutional layers thus they are computationally efficient.

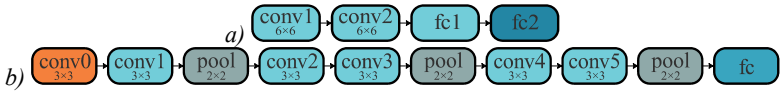

Fig. 2. Architectures of the considered networks

The number of neurons in tree CNNs and tree VGGs is presented in Tab. 1 and 2 respectively. Tab. 3 represents the number of learnable parameters in all the networks.

Tab. 1. The number of neurons in MNIST CNNs

	conv1	conv2	fc1
cnn1	16	32	64
cnn2	32	64	128
cnn3	64	128	128

Tab. 2. The number of neurons in CIFAR-10 CNNs

	conv0	conv1	conv2	conv3	conv4	conv5
vgg/16	32	32	64	64	128	128
vgg/4	64	64	128	128	256	256
vgg	128	128	256	256	512	512

Tab. 3. The number of trainable parameters in the networks

cnn1	cnn2	cnn3	vgg/16	vgg/4	vgg
52.7K	208K	561K	308K	1.19M	4.66M

3.2. Experimental setup

3.2.1. MNIST

MNIST is a dataset of handwritten digits represented as single-channel images with a resolution of 28×28. During training, we applied random rotations of up to 9 degrees and random shifts by ±2 as augmentations. We also binarized the input by applying a threshold of 0.22 (considering pixel values in [0, 1] interval).

We trained our networks for 200 epochs using the Adam optimizer [45] with a batch size of 100 and varying learning rates.

For the SBQ quantizer, we exponentially increased the parameter ν from 1 at the beginning to 1000 as suggested in the original paper [28].

For UBQ, we trained the network for 30 epochs and then replaced batch normalization as described in Section 2.3. After that, we gradually decreased the η values in the layers so that the *conv1*, *conv2*, and *fc* layers were frozen at epochs 132, 158, and 173, respectively. For cnn2 and cnn3, the epochs for freezing were 149, 168, and 173. During the experiments with various learning rates, p was set to 0.2. During the experiments with various values of p, the learning rate was set to 10^{-3}.

3.3. CIFAR-10

For CIFAR-10, which consists of 3-channel 32×32 images in 10 classes, we used simpler augmentations: random horizontal flips and random shifts by ±4 pixels. We trained all the networks using the Adam optimizer with a learning rate of 10^{-3} for all methods, for 500 epochs, with a batch size of 256. The settings for SBQ were the same as in the MNIST experiments. For UBQ, batch normalization was replaced at the 50th epoch. The frozen epochs for all the networks were 320, 365, 388, 410, 455, and 478 for the convolutional layers *conv0* – *conv5*, respectively. The parameter p was set to 0.1.

3.4. Experimental results

We conducted 5 experiments (training a BNN from scratch each time) for each neural network and parameter setting.

Tab. 4 represents a series of experiments with our quantizer with different values of the parameter p (the part of the STE-binarized values in the quantizer). We can see that increasing p to some point helps to improve the accuracy of the result. However, a very high value of p results in lower accuracy. The latter is likely a result of the less stable training process caused by STE. Thus we recommend setting p in the range of 0.1 – 0.2.

Tab. 5 represents a series of experiments with all the quantizers and different learning rates. We report the *highest*, median, and the lowest accuracies for each experiment. We can see that higher learning rates result

in better accuracies. That is likely a result of using the Adam optimizer, which attentively decreases the learning rate for the weights with often gradient direction change (which happens in BNNs, especially with STE quantizer, at the end of the training process), thus eliminating the need to decrease the learning rate manually.

Tab. 4. The results using the UBQ on MNIST with varying p. The median accuracy over 5 experiments, %

BNN \ p	0.02	0.05	0.1	0.2	0.5
cnn1	97.60	97.98	98.08	**98.10**	98.05
cnn2	98.68	98.75	**98.78**	98.72	98.65
cnn3	98.88	98.97	99.00	**99.02**	99.00

Tab. 5. The results on MNIST with varying learning rate. The $\overline{highest}$ (upper), the median (middle), and the $\underline{lowest}$ (lower) obtained accuracy, %

BNN	Quant. \ lr	10^{-4}	$2\cdot10^{-4}$	$5\cdot10^{-4}$	10^{-3}	$2\cdot10^{-3}$
cnn1	STE	$\overline{97.67}$ 96.29 $\underline{86.29}$	$\overline{98.04}$ 95.76 $\underline{92.70}$	$\overline{98.00}$ 97.50 $\underline{95.99}$	$\overline{97.99}$ 97.50 $\underline{95.99}$	$\overline{97.91}$ 97.69 $\underline{92.23}$
	SBQ	$\overline{95.49}$ 89.19 $\underline{87.32}$	$\overline{97.06}$ 96.88 $\underline{88.86}$	$\overline{97.07}$ 96.26 $\underline{88.70}$	$\overline{97.26}$ 96.42 $\underline{94.09}$	$\overline{93.30}$ 91.89 $\underline{90.98}$
	UBQ (ours)	$\overline{97.52}$ **97.26** $\underline{97.02}$	$\overline{97.76}$ **97.60** $\underline{97.43}$	$\overline{98.17}$ **97.93** $\underline{97.79}$	$\overline{98.33}$ **98.07** $\underline{98.02}$	$\overline{98.35}$ **98.10** $\underline{97.99}$
cnn2	STE	$\overline{98.66}$ 98.19 $\underline{96.39}$	$\overline{98.58}$ 98.04 $\underline{97.23}$	$\overline{98.88}$ **98.72** $\underline{97.96}$	$\overline{98.98}$ 98.70 $\underline{97.81}$	$\overline{98.88}$ 98.78 $\underline{98.49}$
	SBQ	$\overline{97.14}$ 96.25 $\underline{88.28}$	$\overline{97.65}$ 97.31 $\underline{96.69}$	$\overline{98.38}$ 97.93 $\underline{97.28}$	$\overline{98.48}$ 98.40 $\underline{98.27}$	$\overline{98.13}$ 98.07 $\underline{97.89}$
	UBQ (ours)	$\overline{98.51}$ **98.33** $\underline{98.27}$	$\overline{98.61}$ **98.50** $\underline{98.41}$	$\overline{98.77}$ 98.66 $\underline{98.58}$	$\overline{98.89}$ **98.82** $\underline{98.76}$	$\overline{98.95}$ **98.86** $\underline{98.79}$
cnn3	STE	$\overline{98.79}$ **98.76** $\underline{98.64}$	$\overline{98.96}$ **98.86** $\underline{98.62}$	$\overline{99.14}$ **99.00** $\underline{98.82}$	$\overline{99.22}$ 98.95 $\underline{98.49}$	$\overline{99.08}$ **99.05** $\underline{99.00}$
	SBQ	$\overline{97.75}$ 97.56 $\underline{96.78}$	$\overline{98.25}$ 97.96 $\underline{96.64}$	$\overline{98.96}$ 98.81 $\underline{98.66}$	$\overline{99.02}$ 98.99 $\underline{98.75}$	$\overline{98.99}$ 98.84 $\underline{98.63}$
	UBQ (ours)	$\overline{98.87}$ 98.69 $\underline{98.57}$	$\overline{98.92}$ 98.82 $\underline{98.68}$	$\overline{99.02}$ 98.94 $\underline{98.85}$	$\overline{99.03}$ **98.99** $\underline{98.84}$	$\overline{99.15}$ 98.99 $\underline{98.84}$

Tab. 6 demonstrates the results of the three considered quantizers on the CIFAR-10 dataset. We also report the $\overline{highest}$, median, and the $\underline{lowest}$ accuracies. From Tables 5 and 6, we can see that the proposed method noticeably outperforms both STE and SBQ for smaller networks (cnn1-2, vgg/16, and vgg/4). For larger networks, it shows results comparable (or maybe slightly lower than STE). Generally, we believe that the latter is a result of the experiment setup: we trained all the networks for a fixed number of epochs (500 for CIFAR-10). STE increased its score during the whole training, while SBQ and our OBQ methods increased the accuracy in the beginning and then started to decrease it when the networks were binarized. It can be seen on training plots (see Fig. 3), showing the median score and score range during training. So more training at the high weight uncertainty stage may have helped our method to achieve higher than STE results.

Tab. 6. The results on CIFAR-10. The $\underline{lowest}$ (left) , the median (middle), and the $\overline{highest}$ (right) obtained accuracy %

BNN \ Quant	STE	SBQ	UBQ (ours)
vggb16	$\underline{60.36}$ 65.41 $\overline{67.96}$	$\underline{32.99}$ 39.11 $\overline{41.40}$	$\underline{70.47}$ 70.70 $\overline{70.93}$
vggb4	$\underline{69.70}$ 74.56 $\overline{78.09}$	$\underline{59.83}$ 63.12 $\overline{67.21}$	$\underline{75.70}$ 76.35 $\overline{77.16}$
vgg	$\underline{78.85}$ 81.22 $\overline{82.05}$	$\underline{73.68}$ 75.43 $\overline{76.16}$	$\underline{79.76}$ 80.05 $\overline{80.41}$

Tabl. 5 – 6 as well as Fig. 3 also suggest that our method is the most stable one, as the deviation between the highest and the lowest accuracies in 5 experiments is lower for our method.

Conclusion

In this paper, we have introduced a novel quantizer called UBQ (Uncertainty-based quantizer) for binary neural networks. UBQ combines the advantages of both STE and SBQ, resulting in stable training and high-quality BNNs even with a low number of trainable parameters. We have also proposed a training method that involves gradual network freezing and batch normalization replacement, which is suitable for UBQ and facilitates a smooth transition from training mode to execution mode for BNNs.

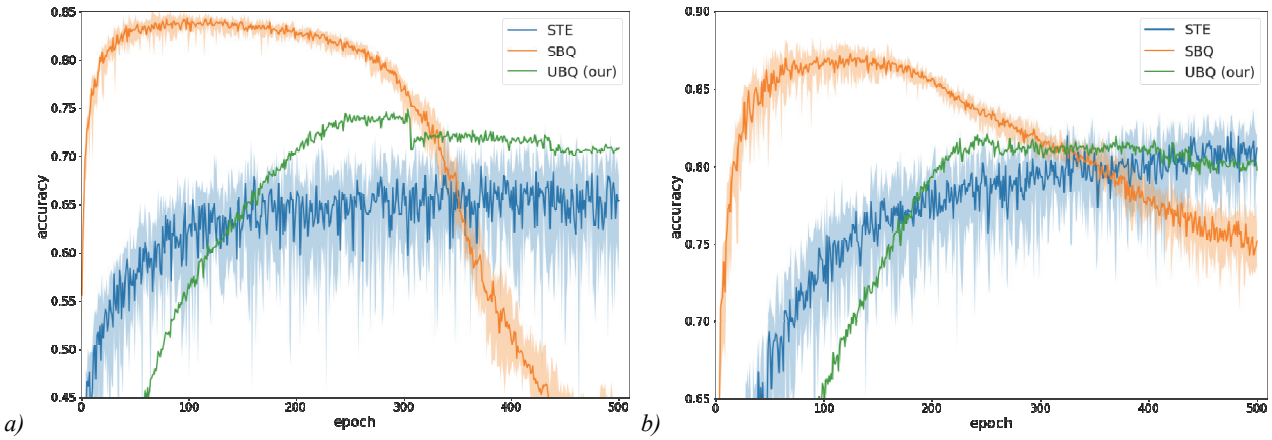

Fig. 3. Training curves of the considered quantizers on CIFAR-10. (a) VGG / 16 (b) VGG

To evaluate the performance of UBQ, we conducted experiments on the MNIST and CIFAR-10 datasets. We trained six different convolutional neural networks with binary layers to solve image classification tasks, using UBQ, STE and SBQ methods. After that, we compared the recognition accuracies. The results demonstrate that UBQ outperforms both methods for smaller networks and achieves comparable results to STE for larger networks.

In terms of future work, there are several directions that can be explored. This includes adapting UBQ parameters (such as freezing time and p) for different tasks and neural networks, experimenting with dynamic weight uncertainty (which is currently manually controlled in UBQ), and applying UBQ ideas to other quantized networks, such as ternary networks. Additionally, it would be interesting to evaluate UBQ against other BNN training algorithms and on a broader variety of tasks.

References

[1] Hoffmann J, Borgeaud S, Mensch A, et al. An empirical analysis of compute-optimal large language model training. Adv Neural Inf Process Syst 2022; 35: 30016-30030.

[2] Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-resolution image synthesis with latent diffusion models. Proc IEEE/CVF Conf on Computer Vision and Pattern Recognition 2022: 10684-10695.

[3] Ilyuhin SA, Sheshkus AV, Arlazarov VL. Recognition of images of korean characters using embedded networks. Proc SPIE 2020; 11433: 1143311. DOI: 10.1117/12.2559453.

[4] Bulatov K, Arlazarov VV, Chernov T, Slavin O, Nikolaev D. Smart IDReader: Document recognition in video stream, 2017 14th IAPR Int Conf on Document Analysis and Recognition (ICDAR) 2017: 39-44. DOI: 10.1109/ICDAR.2017.347.

[5] Sheshkus A, Chirvonaya A, Arlazarov VL. Tiny CNN for feature point description for document analysis: approach and dataset. Computer Optics 2022; 46(3): 429-435. DOI: 10.18287/2412-6179-CO-1016.

[6] Chernyshova YS, Chirvonaya AN, Sheshkus AV. Localization of characters horizontal bounds in text line images with fully convolutional network. Proc SPIE 2020; 11433: 114333F. DOI: 10.1117/12.2559449.

[7] Liang T, Bao H, Pan W, Pan F. ALODAD: An anchor-free lightweight object detector for autonomous driving, IEEE Access 2022; 10: 40701-40714. DOI: 10.1109/ACCESS.2022.3166923.

[8] Sivapalan G, Nundy KK, Dev S, Cardiff B, John D. Annet: a lightweight neural network for ecg anomaly detection in iot edge sensors. IEEE Trans Biomed Circuits Syst 2022; 16(1): 24-35. DOI: 10.1109/TBCAS.2021.3137646.

[9] He Z, Zhang X, Cao Y, Liu Z, Zhang B, Wang X. LiteNet: Lightweight neural network for detecting arrhythmias at resource-constrained mobile devices. Sensors 2018; 18(4): 1229. DOI: 10.3390/s18041229.

[10] Gholami A. Kim S, Dong Z, Yao Z, Mahoney MW, Keutzer K. A survey of quantization methods for efficient neural network inference. In Book: Thiruvathukal GK, Lu Y-H, Kim J, Chen Y, Chen B, eds. Low-power computer vision. New York: Chapman and Hall/CRC; 2022: 291-326.

[11] Rastegari M, Ordonez V, Redmon J, Farhadi A. XNOR-Net: Imagenet classification using binary convolutional neural networks, In Book: Leibe B, Matas J, Sebe N, Welling M, eds. European conference on computer vision. Cham: Springer International Publishing AG; 2016: 525-542. DOI: 10.1007/978-3-319-46493-0_32.

[12] Moss DJ, Nurvitadhi E, Sim J, Mishra A, Marr D, Subhaschandra S, Leong PH. High performance binary neural networks on the Xeon+FPGA™ platform. 2017 27th Int Conf on Field Programmable Logic and Applications (FPL) 2017: 1-4. DOI: 10.23919/FPL.2017.8056823.

[13] He S, Meng H, Zhou Z, Liu Y, Huang K, Chen G. An efficient GPU-accelerated inference engine for binary neural network on mobile phones. J Syst Archit 2021; 117: 102156.

[14] Zhang J, Pan Y, Yao T, Zhao H, Mei T. daBNN: A super fast inference framework for binary neural networks on arm devices. Proc 27th ACM Int Conf on Multimedia 2019: 2272-2275.

[15] Frickenstein A, Vemparala M-R, Mayr J, Nagaraja N-S, Unger C, Tombari F, Stechele W. Binary DAD-Net: Binarized driveable area detection network for autonomous driving. 2020 IEEE Int Conf on Robotics and Automation (ICRA) 2020: 2295-2301.

[16] Xiang X, Qian Y, Yu K. Binary deep neural networks for speech recognition. INTERSPEECH 2017: 533-537.

[17] Alemdar H, Leroy V, Prost-Boucle A, Pétrot F. Ternary neural networks for resource-efficient AI applications. 2017 Int Joint Conf on Neural Networks (IJCNN) 2017: 2547-2554.

[18] Courbariaux M, Bengio Y, David J-P. BinaryConnect: Training deep neural networks with binary weights during propagations. Proc 28th Int Conf on Neural Information Processing Systems (NIPS'15) 2015; 2: 3123-3131.

[19] Liu B, Li F, Wang X, Zhang B, Yan J. Ternary weight networksıo ICASSP 2023-2023 IEEE Int Conf on Acoustics, Speech and Signal Processing (ICASSP) 2023: 1-5. DOI: 10.1109/ICASSP49357.2023.10094626.

[20] Liu Z, Wu B, Luo W, Yang X, Liu W, Cheng K-T. Bi-Real Net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. Proc European Conf on Computer Vision (ECCV) 2018: 722-737.
[21] Chen H, Wang Y, Xu C, Shi B, Xu C, Tian Q, Xu C. AdderNet: Do we really need multiplications in deep learning? Proc IEEE/CVF Conf on Computer Vision and Pattern Recognition 2020: 1468-1477.
[22] Limonova EE. Fast and gate-efficient approximated activations for bipolar morphological neural networks. Informatsionnye Tekhnologii i Vychslitel'nye Sistemy 2022; 2: 3-10. DOI: 10.14357/20718632220201.
[23] Bengio Y, Léonard N, Courville A. Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv Preprint. 2013. Source: <https://arxiv.org/abs/1308.3432>.
[24] Bethge J, Yang H, Bornstein M, Meinel C. Back to simplicity: How to train accurate bnns from scratch? arXiv Preprint. 2019. Source: <https://arxiv.org/abs/1906.08637>.
[25] Bulat A, Tzimiropoulos G. XNOR-Net++: Improved binary neural networks. arXiv Preprint. 2019. Source: <https://arxiv.org/abs/1909.13863>.
[26] Xu Z, Lin M, Liu J, Chen J, Shao L, Gao Y, Tian Y, Ji R. ReCU: Reviving the dead weights in binary neural networks. Proc IEEE/CVF Int Conf on Computer Vision 2021: 5198-5208.
[27] Gong R, Liu X, Jiang S, Li T, Hu P, Lin J, Yu F, Yan J. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. Proc IEEE/CVF Int Conf on Computer Vision 2019: 4852-4861.
[28] Lahoud F, Achanta R, Márquez-Neila P, Süsstrunk S. Self-binarizing networks. arXiv Preprint. 2019. Source: <https://arxiv.org/abs/1902.00730>.
[29] Yang J, Shen X, Xing J, Tian X, Li H, Deng B, Huang J, Hua X-s. Quantization networks. Proc IEEE/CVF Conf on Computer Vision and Pattern Recognition 2019: 7308-7316.
[30] Meng X, Bachmann R, Khan ME. Training binary neural networks using the bayesian learning rule. Int Conf on Machine Learning (PMLR) 2020: 6852-6861.
[31] Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proc IEEE Conf on Computer Vision and Pattern Recognition 2018: 2704-2713.
[32] Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library. Proc 33rd Int Conf on Neural Information Processing Systems (NIPS'19) 2019: 8026-8037.
[33] Keras: Simple. Flexible. Powerful. 2023. Source: <https://keras.io>.
[34] Xue P, Lu Y, Chang J, Wei X, Wei Z. Fast and accurate binary neural networks based on depth-width reshaping. Proc AAAI Conf on Artificial Intelligence 2023; 37: 10684-10692.
[35] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. Proc 32nd Int Conf on on Machine Learning (ICML'15) 2015; 37: 448-456.
[36] LeCun Y. The mnist database of handwritten digits. 1998. Source: <http://yann.lecun.com/exdb/mnist/>.
[37] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv Preprint. 2014. Source: <https://arxiv.org/abs/1409.1556>.
[38] Krizhevsky A, Hinton G, et al. Learning multiple layers of features from tiny images. 2009. Source: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.
[39] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Binarized neural networks. In Book: Lee D, Sugiyama M, Luxburg U, Guyon I, Garnett R, eds. Advances in Neural Information Processing Systems 29 (NIPS 2016). 2016. Source: <https://proceedings.neurips.cc/paper_files/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf>.
[40] Trusov AV, Limonova EE, Nikolaev DP, Arlazarov VV. Fast matrix multiplication for binary and ternary CNNs on ARM CPU. 2022 26th Int Conf on Pattern Recognition (ICPR) 2022: 3176-3182. DOI: 10.1109/ICPR56361.2022.9956533.
[41] Zhang D, Yang J, Ye D, Hua G. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In Book: Ferrari V, Hebert M, Sminchisescu C, Weiss Y, eds. Computer Vision – ECCV 2018. Cham: Springer Nature Switzerland AG; 2018: 365-382.
[42] Darabi S, Belbahri M, Courbariaux M, Nia VP. Regularized binary network training. arXiv Preprint. 2018. Source: <https://arxiv.org/abs/1812.11800>.
[43] Sher AV, Trusov AV, Limonova EE, Nikolaev DP, Arlazarov VV. Neuron-by-neuron quantization for efficient low-bit qnn training. Mathematics 2023; 11(9): 2112. DOI: 10.3390/math11092112.
[44] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors. arXiv Preprint. 2012. Source: <https://arxiv.org/abs/1207.0580>.
[45] Kingma DP, Ba J. Adam: A method for stochastic optimization, arXiv Preprint. 2014. Source: <https://arxiv.org/abs/1412.6980>.

Authors' information

Anton V. Trusov. Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow Russia, 1-st grade programmer. Smart Engines Services LLC, Moscow, Russia, programmer. Number of publications: 8. Research interests: neural networks, image processing, high performance computing.
E-mail: *trusov.av@smartengines.com*

Dmitry N. Putintsev. Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia, senior scientist. Smart Engines Services LLC, Moscow, Russia. Number of publications: 35.
E-mail: *putincevd@gmail.com*

Elena E. Limonova. Federal Research Center "Computer Science and Control" of Russian Academy of Sciences, Moscow, Russia, PhD in Computer Science, researcher. Smart Engines Services LLC, Moscow, Russia, programmer. Number of publications: 38. Research interests: neural networks, image processing, pattern recognition on mobile devices. E-mail: *limonova@smartengines.com*

Code of State Categories Scientific and Technical Information (in Russian – GRNTI)): 28.23.37
Received September 19, 2023. The final version – November 20, 2023.