AdaBin: Improving Binary Neural Networks with Adaptive Binary Sets

Zhijun Tu[1,2], Xinghao Chen[2(✉)], Pengju Ren[1(✉)], and Yunhe Wang[2]

[1] Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University
tuzhijun123@stu.xjtu.edu.cn, pengjuren@xjtu.edu.cn
[2] Huawei Noah's Ark Lab
{zhijun.tu, xinghao.chen, yunhe.wang}@huawei.com

1 Appendix

In this section, we demonstrate the proof of weight equalization, conduct a detailed analysis of complexity for our proposed AdaBin, visualize the feature map and weight KL divergence, provide more experimental results, explain the difference of the binary methods among AdaBin and previous BNNs for weights and activations and show the updating process of the center and distance in activation optimization.

1.1 The proof of Weight Equalization

The Kullback-Leibler divergence (KLD) of real-valued distribution and binary distribution can be represented as Eq. 1.

$$D_{KL}(P_r||P_b) = \int_{x \in \mathrm{w} \& \mathrm{w}_b} P_r(x) \log \frac{P_r(x)}{P_b(x)} dx, \tag{1}$$

where the $P_r(x)$ and $P_b(x)$ denote the probability distributions of real-valued weights and binarized weights. Under the assumption that the binarized data follows the Bernoulli distribution [11], there are only two items left to calculate the KLD, then we can get:

$$D_{KL} = P_r(\mathrm{w}_{b1}) \log \frac{P_r(\mathrm{w}_{b1})}{P_b(\mathrm{w}_{b1})} + P_r(\mathrm{w}_{b2}) \log \frac{P_r(\mathrm{w}_{b2})}{P_b(\mathrm{w}_{b2})}. \tag{2}$$

As the real-valued weight follows a bell-shaped distribution and the both sides are symmetrical on the center (position of mean value), we can set the $P_r(\mathrm{w}_{b1}) = P_r(\mathrm{w}_{b2}) = p$, and $P_b(\mathrm{w}_{b1}) = P_b(\mathrm{w}_{b2}) = 0.5$, then we can further infer as follow:

$$\begin{aligned} D_{KL} &= p \log \frac{p}{0.5} + p \log \frac{p}{0.5} \\ &= 2p \times \log 2p. \end{aligned} \tag{3}$$

✉ Corresponding author

In order to find the optimal solution, we take the derivative of D_{KL} with respect to p and set it to zero:

$$^{*}p = 2^{-\frac{1}{\ln 2}-1} \approx 0.1839. \tag{4}$$

Since there is no convinced formula of weight distribution for neural networks, it is difficult to determine the α_w accurately. However, the weight is widely believed to roughly obey a Laplace distribution [8] or a Gaussian distribution [1,2,13]. Given that the weight w follows a standard Laplace ($\mathrm{w} \sim La(0,1)$) or Gaussian distribution ($\mathrm{w} \sim \mathcal{N}(0,1)$), which means that $\beta_w = 0$, then we can get:

$$f_{Gussian}(\mathrm{w}^1 = 1.245) = f_{Laplace}(\mathrm{w}^2 = 1) \approx 0.1839 \approx {}^{*}p. \tag{5}$$

As show in Fig. 1, these two distributions are similar, and the w^1 and w^2 are close to the standard deviations of these two distributions, which are 1.414 and 1, respectively.

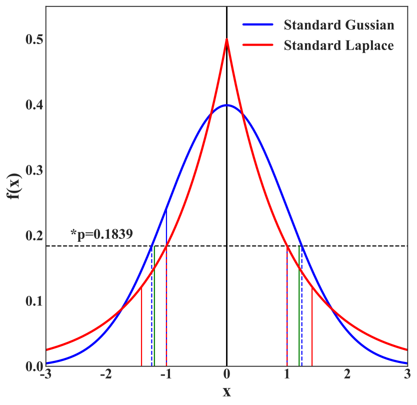


Fig. 1: The distributions of standard Laplace and standard Gaussian. The blue dash lines and the red dash lines represent the position of w^1 and w^2, the blue vertical solid line and the red vertical solid line represent the position of the standard deviations of these two distributions, and the green vertical solid lines represent the estimated α_w.

Therefore, we estimate that α_w is on the position of the standard deviation of w with the consideration of these two distributions in this special case. And as the training goes on, the center of weights is shifted and the shape of distribution gets changed, to make the best distribution-match, the α_w should be changed with the real-valued weight, thus we estimate it as the standard deviation of the weights at each step during training:

$$\alpha_w = \frac{\|\mathrm{w} - \beta_w\|_2}{\sqrt{c \times k \times k}}. \tag{6}$$

Our proof relies on a rough assumption that the distribution of weights is similar to Gaussian distribution and Laplace distribution, which is not very strict but the deduced solution works well in practice.

1.2 Analysis on the Complexity

The goal of BNNs is to replace the computationally expensive multiplication and accumulation with XNOR and BitCount operations, therefore, we will elaborate on how to accelerate our AdaBin in this section.

Binary Convolution With simple linear transformation, our method can also be accelerated with bit-wise operations:

$$\begin{aligned}
y =& \mathrm{Conv}(a_b, \mathrm{w}_b) \\
y =& \mathrm{Conv}(\alpha_a \times b_a + \beta_a, \alpha_w \times b_w + \beta_w) \\
=& \alpha_a \alpha_w \mathrm{Conv}(b_a, b_w) + \alpha_a \beta_w \mathrm{Conv}(b_a, I) + \alpha_w \beta_a \mathrm{Conv}(b_w, I) + \beta_a \beta_w \\
=& \alpha_a \alpha_w (b_a \odot b_w) + \alpha_a \beta_w \sum b_a + F(\mathrm{w}),
\end{aligned} \tag{7}$$

where

$$F(\mathrm{w}) = \alpha_w \beta_a \sum b_w + \beta_a \beta_w .$$

I is the identity matrix with the same shape as weights. As shown in Eq. 7, we get three terms in the convolution with our binary method. The first term is the same as the previous BNNs, the accumulation of the second only needs to accumulation for one output channel, which can be replaced by BitCount, and the third term can be pre-computed in the inference process for that it has nothing to do with the input data.

Parameters and Operations Since the new method involves more full-precision scalars compared with conventional BNN methods, here we will thoroughly analyze the memory usage and computational complexity of the proposed method and other binarization methods with the same neural architecture. Following the analysis method of [2,5,9,12,15], we can get the parameters of previous BNNs and AdaBin as Eq. 8.

$$\begin{aligned}
\mathrm{Params}^{pre} &= c \times n \times k \times k + 32 \times n, \\
\mathrm{Params}^{adabin} &= c \times n \times k \times k + 2 \times 32 \times n + 2,
\end{aligned} \tag{8}$$

where the c denotes the channel number of input data, n and k are the filter number and kernel size of weights. And the operations of previous BNNs and

AdaBin can be obtained in Eq. 9:

$$
\begin{aligned}
\mathrm{OPs}^{pre} =& \frac{\mathrm{BOPs}^{pre}}{64} + \mathrm{FLOPs}^{pre} \\
=& \frac{c \times n \times k \times k \times w' \times h'}{64} + n \times w' \times h', \\
\mathrm{OPs}^{adabin} =& \frac{\mathrm{BOPs}^{adabin}}{64} + \mathrm{FLOPs}^{adabin} \\
=& \frac{(c \times n \times k \times k \times w' \times h' + oc \times w' \times h')}{64} \\
& + 2n \times w' \times h',
\end{aligned} \tag{9}
$$

where h' and w' denote the height and width of weights, the BOPs include the bit-wise XNOR and BitCount, the FLOPs denote the floating point multiplication and accumulation.

Complexity on Single Convolution Layer We fix the parameters as previous BNNs [12]: $n = c = 256$, $k = 3$, $w' = h' = 14$, the comparison results are illustrated in Table 1. We can see that our method only increases 2.74% operations and 1.37% parameters, which are negligible compared to the total complexity, and AdaBin could achieve 60.85× acceleration and 31× memory savings in theory, which is similar to the previous BNNs.

Method	BOPs (M)	FLOPs (M)	OPs (M)	Params (Mbit)	Acceleration	Memory Saving
Full Precision	0	115.61	115.61	18.88	1×	1×
Previous BNNs	115.61	0.05	1.85	0.598	62.46×	32×
AdaBin (Ours)	115.65	0.10	1.90	0.606	60.85×	31×

Table 1: Efficiency Analysis of single convolution layer for different methods.

Complexity on ResNet-18 To further prove the efficiency of our proposed AdaBin, we also calculate the parameters and operations of the complete ResNet-18 structure with different input resolutions, and compare with some classical binary neural networks, such as BNN [3], XNOR-Net [12], IR-Net [11], and ReActNet [9]. As show in Table 2, our method only increase 3.5% operations than IR-Net, but obtain 1.6% accuracy improvement on CIFAR-10 dataset. And we exceed the ReActNet by 0.9% on ImageNet with only extra 2.5% operations. Besides, our method only increases less than 1% memory footprint compared to other methods, which is much negligible to the total complexity.

Speed-Accuracy In addition, we test the inference speed of ResNet-18 and ResNet-34 with the proposed AdaBin, BiRealNet[10] and ReCU[16] on a 1.5GHz

Dataset	Resolution	Method	BOPs ($\times10^9$)	FLOPs ($\times10^8$)	OPs ($\times10^8$)	Params ($\times10^8$ bit)	Top-1 (%)
CIFAR-10	$32 \times 32 \times 3$	Full-Precision	0	5.63	5.63	3.58	94.8
		BNN [3]	0.55	0.16	0.24	0.17	-
		XNOR-Net [12]	0.55	0.17	0.26	0.18	-
		IR-Net [11]	0.55	0.17	0.25	0.18	91.5
		AdaBin (Ours)	0.56	0.17	0.26	0.18	93.1
ImageNet	$224 \times 224 \times 3$	Full-Precision	0	18.12	18.12	3.74	69.6
		BNN [3]	1.68	1.21	1.47	0.34	42.2
		XNOR-Net [12]	1.68	1.41	1.67	0.34	51.2
		IR-Net [11]	1.68	1.37	1.63	0.34	58.1
		ReActNet [9]	1.68	1.37	1.63	0.34	65.5
		AdaBin (Ours)	1.69	1.41	1.67	0.34	66.4

Table 2: Efficiency Analysis of ResNet-18 for different methods.

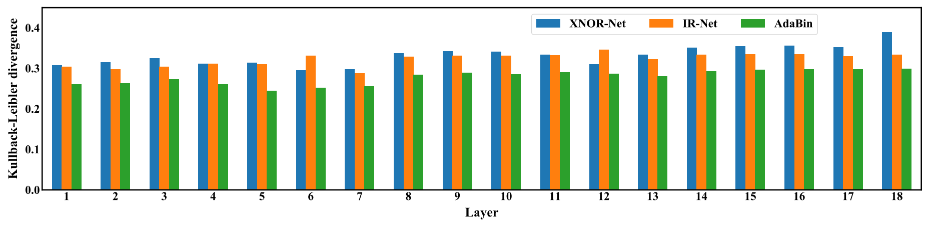


Fig. 2: The Kullback-Leibler divergence of weight binarization for each layer on ResNet-20. Following the previous BNNs [10,11,14], the first convolution and last full-connected layers are kept in 32-bit.

ARM core using the Bolt framework[3], respectively, as shown in Fig. 3b. Specifically, our proposed AdaBin achieves quite similar latency with previous BNNs, BiRealNet and ReCU, but with much better classification accuracy on ImageNet. Besides, The second term of Eq. 7 actually could be accelerated by BitCount operation, but the Bolt framework is not optimized for this, which brings additional latency and could be optimized for better speed.

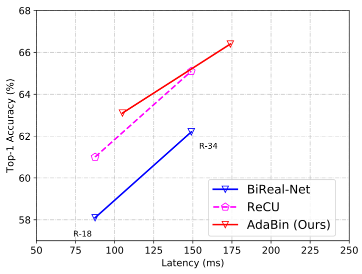
1.3 Visualization

Visualization on activations. We visualize the activation with different methods in the Fig. 1b of the full paper. As we can see, real-valued activations contain a lot of feature details. However, the activations binarized by sign function will result in a large blank area and lose much feature information in some cases, which fails to facilitate the learning of features for the current layer. In contrast, AdaBin could retain much texture feature of real-valued activations, and enable the binary neural networks to perform adequate learning and efficiently update parameters during training.

[3] https://github.com/huawei-noah/bolt

Networks	Methods	W/A	Top-1 (%)	Top-5 (%)
ResNet-18	Full-Precision	32/32	69.6	89.2
	SQBWN [4]	1/32	58.4	81.6
	BWN [12]		60.8	83.0
	HWGQ [7]		61.3	83.2
	BWHN [6]		64.3	85.9
	IR-Net [11]		66.5	86.8
	AdaBin (Ours)		**68.0**	**87.9**
ResNet-34	FP	32/32	73.3	91.3
	IR-Net [11]	1/32	70.4	89.5
	AdaBin (Ours)		**71.0**	**89.6**

(a) Results on ImageNet



(b) Speed-accuracy tradeoff.

Table 3: (a) Comparison with state-of-the-art methods on ImageNet. W/A denotes the bit width of weights and activations. (b) Accuracy vs. Latency on ImageNet datasets.

Visualization on weights. We use different methods to extract the weights of the ResNet-20 structure on the CIFAR-10 dataset, and calculate the Kullback-Leibler divergence between the real-valued distribution and the binarized distribution for each layer. As shown in Figure 2, Using our proposed AdaBin can significantly reduce the Kullback-Leibler divergence compared to XNOR and IR-Net, which justifies the motivation of our method. It also demonstrates that AdaBin could make the binary weights best match the real-valued weights, which help to improve the feature extraction capacity of the binary convolution kernel.

1.4 More Experimental Results

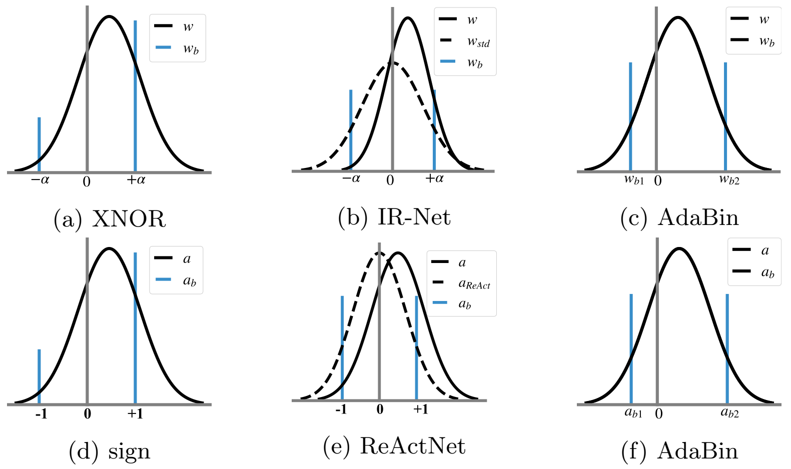
To further demonstrate the effectiveness of our AdaBin, we evaluate it on ResNet-18 and ResNet-34 with 1-bit weights and 32-bit activations on the ImageNet dataset. Table 3a shows that AdaBin achieves 68.0% and 71.0% Top-1 accuracy, which surpasses the IR-Net by 1.5% adn 0.6%, and close the performance gap between binary neural networks and real-valued networks.

1.5 Comparison with Previous BNNs

To demonstrate the difference of binary methods between our AdaBin and previous BNNs, we plot the binarization process of several methods for weights and activations.

The weight binarization of XNOR-Net [12] is the most common method in binary neural networks as shown in Fig. 3a, which extracts the ℓ_1-norm of real-valued weights (denoted as α) to recover the information of binarized weights. XNOR-Net proves that this method could minimize the quantization error with the constraint that the binary values of w_b must be a pair of opposite numbers, which could define as $\alpha \times \{-1, +1\}$. To get the largest information entropy and smallest quantization error, IR-Net [11] first normalizes the weights, forces w into

w_{std}, and then binarizes the w_{std} to $\{-\alpha, +\alpha\}$ just like XNOR-Net as shown in Fig. 3b. Although IR-Net could obtain the best information entropy, the binarized weights can not match the distribution of initial real-valued weights. Besides, the optimal scale α is calculated based on w_{std}, which is also not optimal for initial real-valued weights. Fig. 3c shows our proposed AdaBin, which removes the restriction of $\{-\alpha, +\alpha\}$. AdaBin obtains the analytic solution of binary values w_{b1} and w_{b1}, which could get a much better match between binary weights and initial real-valued weights regardless of how the distribution of weights change during training. It is clear that AdaBin could truly minimize the Kullback-Leibler divergence of real-valued weights and binarized weights. Sign



(a) XNOR (b) IR-Net (c) AdaBin

(d) sign (e) ReActNet (f) AdaBin

Fig. 3: The binarization process for weights and activations of different methods. The first row represents the weights, and the second row represents the activations.

function is the most common technique for binarizing activations to either +1 or -1 as shown in Fig. 3d, which is hardware-friendly but provides rough feature maps. To enable explicit learning of the distribution shift, ReActNet [9] proposes the ReAct sigh function (RSign) to replace the sigh function but still binarize the activations to either +1 or -1 as shown in Fig. 3e. However, we argue that the offset and amplitude are still important for binarized activations to retain more information of initial real-valued activations. Therefore, the fixed binary set $\{-1, +1\}$ cannot provide diverse feature maps and thus limits the feature representation. Fig. 3f shows the activation binarization of our proposed AdaBin, we remove the restriction of the fixed set $\{-1, +1\}$, and expand the binary values to arbitrary values, which could retain more texture feature of real-valued activation (as shown in Fig. 1b of section 4.4) compared to sign function.

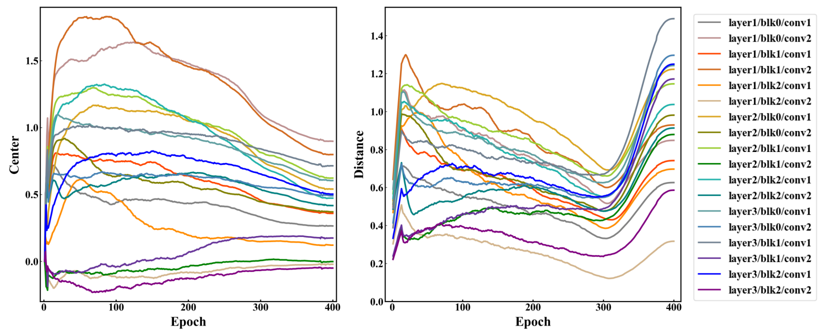


Fig. 4: Updating of center β_a and distance α_a during training using ResNet-20 structure on CIFAR-10 dataset.

Overall, the binary methods for weights and activations of AdaBin are much different from previous BNNs, and could significantly enhance the performance of binary neural networks.

1.6 The updating of α_a and β_a

To prove the efficiency of the gradient-based optimization method for activation binarization, we visualize the center β_a and distance α_a of different epoch during training. As shown in Fig. 4, we set the initial values of center and distance to 0 and 1, so that the initial binary quantizer is equivalent to the sign function. Then these two parameters of different layers are dynamically updated via gradient descent-based training, and converge to the adaptive center and distance values, which is much different from the unified use of the sign function in the previous BNNs.

References

1. Baskin, C., Liss, N., Schwartz, E., Zheltonozhskii, E., Giryes, R., Bronstein, A.M., Mendelson, A.: Uniq: Uniform noise injection for non-uniform quantization of neural networks. ACM Transactions on Computer Systems (TOCS) **37**(1-4), 1–15 (2021)
2. Chen, H., Wang, Y., Xu, C., Shi, B., Xu, C., Tian, Q., Xu, C.: Addernet: Do we really need multiplications in deep learning? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1468–1477 (2020)
3. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
4. Dong, Y., Ni, R., Li, J., Chen, Y., Zhu, J., Su, H.: Learning accurate low-bit deep neural networks with stochastic quantization. arXiv preprint arXiv:1708.01001 (2017)
5. Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C.: Ghostnet: More features from cheap operations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
6. Hu, Q., Wang, P., Cheng, J.: From hashing to cnns: Training binary weight networks via hashing. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 32 (2018)
7. Li, Z., Ni, B., Zhang, W., Yang, X., Gao, W.: Performance guaranteed network acceleration via high-order residual quantization. In: Proceedings of the IEEE international conference on computer vision. pp. 2584–2592 (2017)
8. Lin, M., Ji, R., Xu, Z., Zhang, B., Chao, F., Xu, M., Lin, C.W., Shao, L.: Siman: Sign-to-magnitude network binarization. arXiv preprint arXiv:2102.07981 (2021)
9. Liu, Z., Shen, Z., Savvides, M., Cheng, K.T.: Reactnet: Towards precise binary neural network with generalized activation functions. In: European Conference on Computer Vision. pp. 143–159. Springer (2020)
10. 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. In: Proceedings of the European conference on computer vision (ECCV). pp. 722–737 (2018)
11. Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., Song, J.: Forward and backward information retention for accurate binary neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2250–2259 (2020)
12. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European conference on computer vision. pp. 525–542. Springer (2016)
13. Rennie, J.: On l2-norm regularization and the gaussian prior (2003)
14. Wang, P., He, X., Li, G., Zhao, T., Cheng, J.: Sparsity-inducing binarized neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 12192–12199 (2020)
15. Wang, Y., Xu, C., You, S., Tao, D., Xu, C.: Cnnpack: Packing convolutional neural networks in the frequency domain. In: NIPS. vol. 1, p. 3 (2016)
16. 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. arXiv preprint arXiv:2103.12369 (2021)