BINARYDUO: REDUCING GRADIENT MISMATCH IN BINARY ACTIVATION NETWORK BY COUPLING BINARY ACTIVATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Binary Neural Network (BNN) has been gaining interest thanks to its computing cost reduction and memory saving. However, BNN suffers from performance degradation mainly due to the gradient mismatch caused by binarizing activations. Previous works tried to address the gradient mismatch by reducing the discrepancy between activation functions used at forward and backward passes, which is an indirect measure. In this work, we introduce coordinate discrete gradient (CDG) to better estimate the gradient mismatch. Analysis using the CDG indicates that using higher precision for activation is more effective than modifying the backward pass of binary activation function. Based on the observation, we propose a new training scheme for binary activation network called BinaryDuo in which two binary activations are coupled into a ternary activation during training. Experimental results show that BinaryDuo outperforms state-of-the-art BNNs on various benchmarks with the same amount of parameters and computing cost.

1 INTRODUCTION

Deep neural networks (DNNs) have been achieving remarkable performance improvement in various cognitive tasks. However, the performance improvement generally comes from the increase in the size of the DNN model, which also greatly increases the burden on memory usage and computing cost. While server-level environment with large number of GPUs can easily handle such a large model, deploying a large-size DNN model on resource-constrained platforms such as mobile application is still a challenge.

To address the challenge, several network compression techniques such as quantization (Wu et al., 2018b; Zhou et al., 2016; Lin et al., 2016), pruning (Han et al., 2015; Hu et al., 2016; Wen et al., 2016), and efficient architecture design (Howard et al., 2017; Wu et al., 2018a; Tan & Le, 2019) were introduced. For quantization, previous works tried to reduce the precision of the weights and/or activations of a DNN model to even 1-bit (Courbariaux et al., 2016; Rastegari et al., 2016). In the Binary Neural Network (BNN), in which both weights and activations have 1-bit precision, high-precision multiplication and accumulation can be replaced by XNOR and pop-count logics which are much cheaper in terms of hardware cost. In addition, the model parameter size of a BNN is 32x less than its full-precision counterpart.

Although BNN can exploit various hardware-friendly features, its main drawback is the accuracy loss. An interesting observation is that binarization of activation causes much larger accuracy drop than the binarization of weights (Cai et al., 2017; Mishra et al., 2018). In particular, it can be observed that the performance degradation is much larger when the bit precision is reduced from 2-bit to 1-bit than other multi-bit cases (>2-bit) as shown in Table 1. To explain the poor performance of binary activation networks, the gradient mismatch concept was introduced (Lin & Talathi, 2016; Cai et al., 2017) and several works proposed to modify the backward pass of activation function to reduce the gradient mismatch problem (Cai et al., 2017; Darabi et al., 2018; Liu et al., 2018).

In this work, we first show that previous approaches to minimize the gradient mismatch have not been very successful in reducing accuracy loss because the target measure which the approaches tried to minimize is not effective. To back up the claim, we propose a new measure for better estimation of gradient mismatch. Then, we propose a new training scheme for binary activation net-

Table 1: Classification accuracy of quantized neural network on ImageNet dataset.

WRPN AlexNet 2x-wide (Mishra et al., 2018)						ABC-Net ResNet-18 (Lin et al., 2017)			
W\A	32b	8b	4b	2b	1b	W\A	5b	3b	1b
32b	60.5	58.9	58.6	57.5	**52.0**	5b	65.0	62.5	**54.1**
8b	-	59.0	58.8	57.1	**50.8**	3b	63.1	61.0	**49.1**

work called *BinaryDuo* based on the observation that ternary (or higher bitwidth) activation suffers much less from gradient mismatch problem than the binary activation. We show that BinaryDuo can achieve state-of-the-art performance on various benchmarks (VGG-7, AlexNet and ResNet). We also provide our reference implementation code online[1].

2 Related works

Binary Neural Network BNN was first introduced by Courbariaux et al. (2016). In BNN, both weights and activations are constrained to +1 or -1 and sign function is used to binarize weights and activations. Since sign function is not differentiable at a single point and has zero gradient everywhere else, the concept of straight-through estimator (STE) was used for back-propagation (Bengio et al., 2013). Courbariaux et al. (2016) used HardTanh function as STE so that its derivative is used to approximate the backward pass of the sign function. Rastegari et al. (2016) proposed similar form of BNN, called XNOR-Net. XNOR-Net used $+\alpha/-\alpha$ instead of +1/-1, where α is a channel-wise full-precision scaling factor.

Sophisticated STEs Although using STE enabled the back-propagation of the quantized activation function, a gradient mismatch caused by the approximation occurs across the input range of binary activation function. To mitigate the gradient mismatch problem, Darabi et al. (2018) proposed to use SwishSign function as STE, while Liu et al. (2018) proposed to use the polynomial function.

Continuous binarization Few recent work proposed to use a continuous activation function that increasingly resembles a binary activation function during training, thereby eliminating approximation process across activation function. Sakr et al. (2018) used a piecewise linear function of which the slope gradually increases, while Yang et al. (2019) and Gong et al. (2019) proposed to use sigmoid and tanh function, respectively.

Additional shortcut Most recent models have shortcut connections across layers. Since the parameter size and computing cost of shortcut connection is negligible, shortcut data often maintains full-precision even in BNN. Bi-Real-Net (Liu et al., 2018) pointed out that these information-rich shortcut paths help improving performance of BNN, and hence proposed to increase the number of shortcut connections in ResNet architecture. Since additional shortcuts increase element-wise additions only, the overhead is negligible while the effect on performance is noticeable.

3 Sharp accuracy drop from 2-bit to 1-bit activation

3.1 Experimental confirmation of the poor performance of binary activation

In this section, we first verify that performance drop of a neural network is particularly large when the bit-precision of the activation is reduced from 2-bit to 1-bit (Table 1). We used VGG-7 (Simonyan & Zisserman, 2014) network on CIFAR-10 dataset for our experiment. We trained the benchmark network with full-precision, 4-bit, 3-bit, 2-bit and 1-bit activation while keeping all weights in full-precision to solely investigate the effect of activation quantization. Activation quantization was carried out by clipping inputs outside of [0,1] and then applying uniform quantization as in Eq. 1.

$$Q(x) = \frac{1}{2^k - 1} \cdot \text{round}\big(\text{clip}(x, 0, 1) \cdot (2^k - 1)\big) \tag{1}$$

Here, x and Q(x) are input and output of quantization function, and k is number of bits. For consistency, we used Eq. 1 for binary activation function instead of the sign function which is another

[1]The source code will be available after review process.

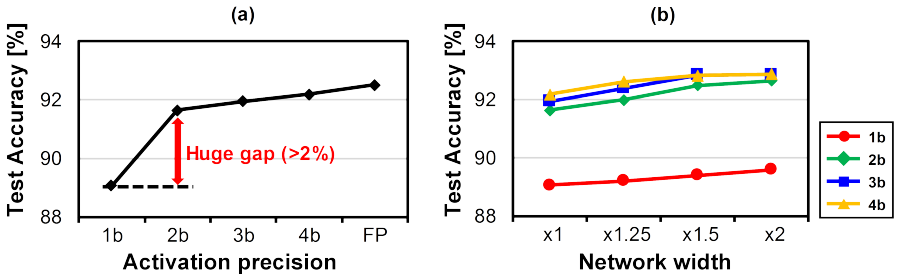

Figure 1: Training results of VGG-7 on CIFAR-10 dataset. (a) Test accuracy with various activation precision. (b) Test accuracy of various activation precision and various network width.

widely used option (Cai et al., 2017; Zhou et al., 2016; Mishra et al., 2018). We trained each network for 200 epochs and detailed experimental setups are described in Appendix A. The best test accuracy from each precision is plotted in Figure 1a. It is clearly shown that reducing activation bitwidth from 2-bit to 1-bit results in the substantial drop in the test accuracy.

One might argue that the sharp accuracy drop occurred because the capacity of the binary activation network was not large enough for the given task while that of the network with 2-bit or higher bitwidth was. To verify, we increased the model size by modifying the network width (not depth) for each activation precisions (Figure 1b). We found that the large accuracy gap still existed between 2-bit activation network and 1-bit activation network regardless of the network width. For example, 1-bit activation network with 2 times wider layers needs more parameters and more computations, but shows much worse accuracy than 2-bit activation network with original width. In addition, poor performance of binary activation was also observed in different networks (e.g. AlexNet and ResNet) and different datasets as in Table 1. Therefore, we argue that the sharp accuracy drop for the binary activation stems from the inefficient training method, not the capacity of the model.

3.2 Gradient mismatch and sophisticated STEs

When training a neural network with quantized activation function such as thresholding function, STE is often used for back-propagation. In such case, we call the back-propagated gradient using STE as *coarse gradient* following Yin et al. (2019). However, training with the coarse gradient suffers from gradient mismatch problem caused by the discrepancy between the presumed and the actual activation function (Lin & Talathi, 2016). The gradient mismatch makes the training process noisy thereby resulting in poor performance.

Unfortunately, it is not possible to measure the amount of gradient mismatch directly because the *true gradient*[2] of a quantized activation function is zero almost everywhere. Therefore, previous works used an indirect criterion to measure gradient mismatch based on the discrepancy between the actual activation function and the STE (Darabi et al., 2018; Liu et al., 2018). In Liu et al. (2018), cumulative difference between actual activation function and STE (Eq.2) was used as a criterion for gradient mismatch.

$$\text{cumulative difference} = \int_{-\infty}^{\infty} |f(x) - g(x)| dx \tag{2}$$

Here, f(x) and g(x) are actual activation function and STE, respectively. Previous works tried to improve the accuracy by designing STE which has the small cumulative difference from the binary activation function. Figure 2 shows different options for STE proposed by previous works (Darabi et al., 2018; Liu et al., 2018). The shaded area in each activation function represents cumulative difference with thresholding function. We conducted the same experiment for a binary activation network as in previous section with various sophisticated STEs. For each STE, the cumulative difference and test accuracy are shown in Figure 2. As proposed in previous works, cumulative difference of sophisticated STEs are much lower than conventional STE (ReLU1). However, its effect on network performance is quite limited in our experiment. As shown in the Figure 2, large accuracy gap still exists between binary activation network and 2-bit activation network even if

[2]We call the original gradient as 'true gradient' to distinguish with the coarse gradient.

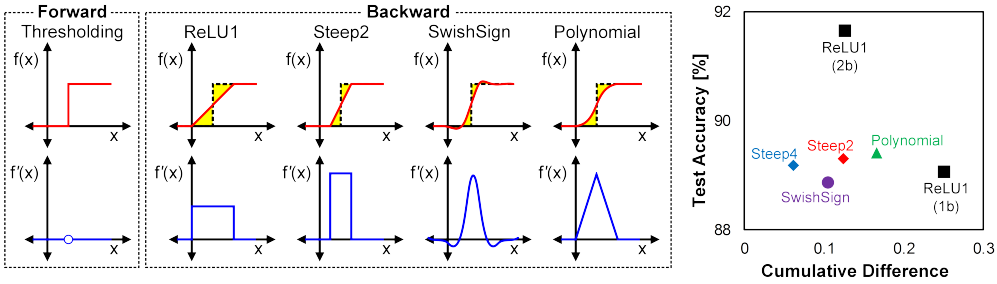

Figure 2: Activation functions and their derivatives used at forward and backward passes (left). The relationship between the best test accuracy and the cumulative difference of various STEs (right). Steep4 is extended version of steep2 with slope of 4.

the sophisticated STEs are used. From this observation, we think that current method of measuring gradient mismatch using the cumulative difference cannot accurately describe the amount of gradient mismatch. The same observation was also reported by Bethge et al. (2019) recently. Therefore, in the following section, we introduce a better method for estimating the effect of gradient mismatch problem.

4 Estimating the Gradient Mismatch

4.1 Coordinate Discrete Gradient

Since the true gradient of quantized activation network is zero almost everywhere, we cannot use the true gradient to find the steepest descent direction, which is the direction toward the point with the smallest loss at given distance. This is the main reason why there has been no proper criterion for estimating gradient mismatch. Recently, Yin et al. (2019) introduced a method to analyze the effect of different STEs (Identity, ReLU, Clipped ReLU) using the true gradient for the expected loss given the distribution of the data. However, the distribution of the dataset is usually unknown and the gradient of the layers other than the first layer are also zero in multi-layered networks.

Here, we introduce *coordinate discrete gradient* (CDG) to approximate the steepest descent direction without using the true gradient. We define the CDG as follows:

$$\overline{\nabla}_{\varepsilon,\boldsymbol{x}} f = \Big(\frac{f(\boldsymbol{x} + \varepsilon \cdot e^{(1)}) - f(\boldsymbol{x} - \varepsilon \cdot e^{(1)})}{2\varepsilon}, ..., \frac{f(\boldsymbol{x} + \varepsilon \cdot e^{(n)}) - f(\boldsymbol{x} - \varepsilon \cdot e^{(n)})}{2\varepsilon} \Big) \tag{3}$$

Here $e^{(i)}$ refers to standard basis vector with a 1 at position i. Note that the formula is very close to the definition of the true gradient which is $\lim_{\varepsilon \to 0} \tilde{\nabla}_{\varepsilon} f(\mathbf{x})$. While the true gradient indicates the instant rate of change of loss ($\varepsilon \to 0$), CDG is based on the average rate of change in each coordinate. Since CDG does not require derivative calculation, the steepest descent direction can be mimicked by CDG even in quantized neural network in which derivative of activation function is zero almost everywhere. We provide a graphical explanation on CDG in Appendix B.

4.2 Estimating the Gradient Mismatch using Cosine Similarity

Using CDG, we can quantitatively estimate 1) the gradient mismatch when different precisions are used for activation, and 2) the effect of sophisticated STEs. By calculating cosine similarity between CDG and coarse gradient in each case, we compare how similar the coarse gradient calculated by back-propagation is to CDG. The detailed algorithm for calculating cosine similarity between coarse gradient and CDG is described in algorithm 1 in Appendix C. Since the computational burden of calculating CDG is heavy even on GPU, we used a toy model and dataset for experiment. We used a feed-forward neural network with 3 hidden layers. Each hidden layer had 32 neurons and we used Gaussian distribution as the dataset as in Yin et al. (2019).

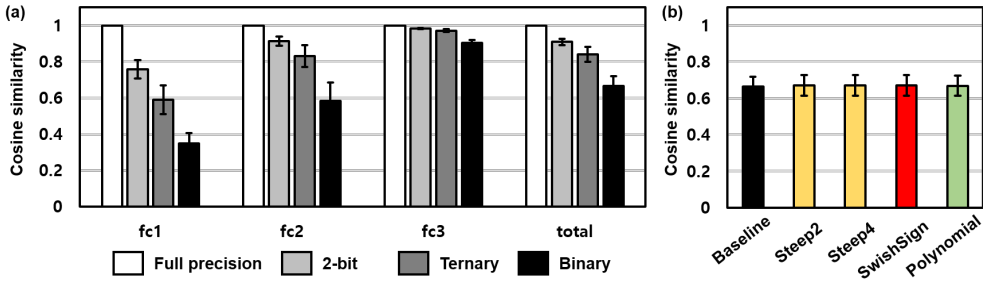

Figure 3: Cosine similarity between true/coarse gradient and CDG in a feed-forward network with 3 hidden layers. (a) Cosine similarity value of each layer when different precision is used for activation. The label *total* refers to the cosine similarity when gradients for each layer are concatenated. (b) Cosine similarity values when sophisticated STEs are used.

Figure 3 shows the cosine similarity between CDG and coarse gradient in different cases. First of all, the cosine similarity between CDG and true gradient in full precision case which has smooth loss surface is almost 1 in every layer. This implies that the CDG points to a similar direction to the one which the true gradient points to if the loss surface is not discrete. We also observed that the cosine similarity of binary activation network was significantly lower than that of higher precision networks and the degradation increases as the layer becomes far from the last layer. This is because the gradient mismatch is accumulated as the gradient propagates through layers from the end of network. On the other hand, we found that using sophisticated STEs does not improve the cosine similarity of binary activation network (Figure 3b), which is consistent with the accuracy results in Figure 2. The results indicate that the cosine similarity between coarse gradient and CDG can explain the relationship between gradient mismatch and performance of model better than previous approaches. In addition, we found that gradient mismatch can be reduced more effectively by using higher precision (e.g. ternary or 2-bit) activations than using sophisticated STEs.

5 BINARYDUO: THE PROPOSED METHOD

In this section, we introduce a simple yet effective training method called BinaryDuo which couples two binary activations during training to exploit better training characteristics of ternary activation. From the observation in the previous section, we learned that ternary activation function suffers much less from gradient mismatch problem than binary activation function with any kind of STE. Therefore, we propose to use such characteristics of ternary activation while training binary activation network. The key idea is that a ternary activation function can be represented by two binary activation functions (or thresholding functions).

The BinaryDuo training scheme consists of two stages. In the first stage, a network with ternary activation function is trained. This pretrained model is then deployed to a binary activation network by decoupling a ternary activation into a couple of binary activations, which provides the initialization values of the target binary activation network. In the next stage, the decoupled binary network is fine-tuned to find the better weight parameters. Detailed description of each step is given as follows.

5.1 DECOUPLING A TERNARY ACTIVATION TO TWO BINARY ACTIVATIONS

We first elaborate how to decouple a ternary activation to two binary activations. Figure 4 shows three different cases of activation function. The weighted-sum value (x) goes through activation function depicted in each case. The activation value is then multiplied by a certain weight and contributes to the next weighted-sum. For simplicity, batch-normalization layer (BN) and pooling layer are omitted in the figure. The binary activation function used in the baseline network is shown in the left in Figure 4. The figure in the middle explains how ternary activation function works. The ternary activation function can produce three different values (0, 0.5 or 1) for an output as depicted in the figure. Since the input to the activation function is 0.7 in the example, the output (rectangle) becomes 0.5. The activation is then multiplied by the corresponding weight of 2.2 and contributes 1.1 to the next weighted-sum. The right figure shows how to decouple the ternary activation into

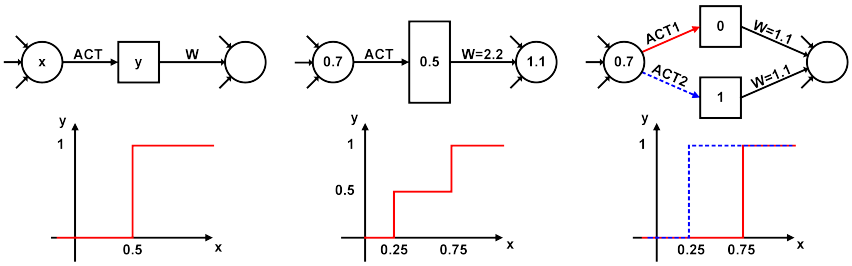

Figure 4: Activation functions of baseline binary (left), ternary (middle) and decoupled binary (right) models. Circle, square and rectangle denote full-precision, binary and ternary data, respectively.

two binary activations. The weighted-sum value of 0.7 now passes through two different activation functions (red solid and blue dashed), where each activation function has different threshold values (0.75 and 0.25) to mimic the ternary activation function. The two binary activation values are then multiplied by each weight that is half of the original value. Therefore, they contribute the same value of 1.1 as the ternary case to the next weighted-sum. Note that decoupling the ternary activation does not break the functionality. In other words, computation result does not change after decoupling.

As mentioned earlier, the decoupled binary activation function layer should have a threshold of 0.25 in half and 0.75 in the other half to mimic the ternary activation function. In practice, instead of using different threshold values in a layer, we shift the bias of BN layer which comes right before the activation function layer. The ternary activation function can be expressed as follows:

$$y_t = f_t(BN(\Sigma, \gamma, \beta)). \tag{4}$$

Here, f_t and y_t are ternary activation function and its output as shown in the middle of Figure 4. BN(x,γ,β) denotes BN function with input x, weight γ and bias β. Similarly, two binary activation functions after decoupling can be expressed as Eq. 5.

$$y_{b1} = f_b(BN(\Sigma, \gamma, \beta + 0.25)), \quad y_{b2} = f_b(BN(\Sigma, \gamma, \beta - 0.25)) \tag{5}$$

Again, f_b and y_b are binary activation function and its output as shown in the left of Figure 4. While γ of BN is copied from the ternary activation case, β is modified during decoupling. When a ternary activation function is decoupled following Eq. 5, $y_t = (y_{b1} + y_{b2})/2$ becomes valid, enabling exact same computation after decoupling. Please note that BN layers followed by binary activation layer can be merged to the threshold of the binary activation layer, incurring no overhead at inference stage.

5.2 Layer width reconfiguration

As shown in Figure 4, the number of weights is doubled after decoupling. Therefore, to match the parameter size of the decoupled model and the baseline model, we must reduce the size of the coupled ternary model. Figure 5 shows an example how we choose the network size. Each diagram shows BN-activation (ACT)-fully connected (FC)-BN-ACT layers in order from the left. In case the baseline model has a 3x3 configuration for the FC layer, the layer has a total of nine weights. In this case, we train the coupled ternary model with 2x2 configuration which has a total of 4 weights. The third diagram shows the corresponding decoupled binary model. As explained in Eq. 5, a weighted-sum value generates two BN outputs with different BN biases. Since decoupling doubles the number of weights in the model, the decoupled model has 8 weights which is still less than the number of weights in the baseline model. To make sure that the number of total weights in the decoupled binary model does not exceed that of the baseline model, we use a simple formula, $N_{coupled} = \lfloor N_{baseline}/\sqrt{2} \rfloor$, where N denotes the number of neurons (or channels in convolution).

5.3 Fine-tuning the decoupled model

After decoupling, two weights derived from the same weight in the coupled model have the same value (1.1 in the example) since they were tied in the ternary activation case. In the fine-tuning

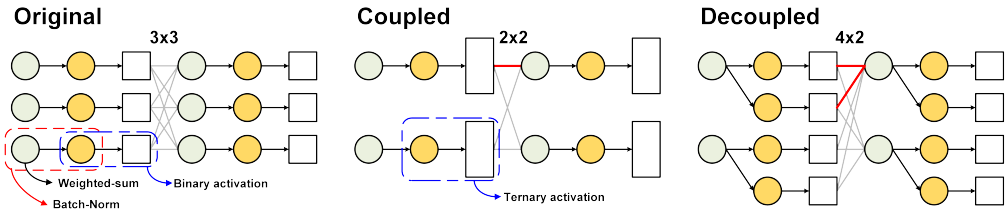

Figure 5: Example model architectures of baseline, coupled ternary model and decoupled binary model with specific width numbers. Two weights highlighted in the decoupled model are half the value of the weight highlighted in the coupled model.

stage, we update each weight independently so that each of them can find a better value since the two weights do not need to share the same value anymore. As mentioned earlier, decoupling itself does not change the computation result, so the same accuracy as that of the coupled ternary model is achieved before fine-tuning and it is expected that the fine-tuning increases the accuracy even further. Since the initial state before fine-tuning can be close to a good local minimum even after decoupling, we used much smaller learning rate for fine-tuning.

6 Experimental Results

6.1 VGG-7 on CIFAR-10

We evaluate the proposed training method in two steps. First, we validate the effectiveness of the BinaryDuo scheme with VGG-7 on CIFAR-10 dataset. We used the same setting with the experiment introduced in Figure 1. We first trained the coupled ternary model, then we decoupled the trained coupled ternary model as explained in section 5.1. The training results in various training schemes and activation precision are summarized in Figure 6. The three contour plots show the hyper parameter search results for each case. Please refer to Appendix A for more details on experimental setup. First of all, note that the best test accuracy of the coupled ternary model is 89.69% which is already higher than 89.07% accuracy of the baseline binary model. In other words, using ternary activation and cutting the model size in half improved the performance of the network. By fine-tuning the decoupled binary model, we could achieve 90.44% test accuracy, which is 1.37% higher than the baseline result. It can be observed that the steep drop of accuracy from 2b activation case to 1b activation case is significantly reduced with our scheme.

Even though the decoupled binary model has the same (or less) number of parameters and computing cost than the baseline model, someone may wonder whether the performance improvement might be an outcome from the modified network topology. Therefore, we conducted another experiment to rule out the possibility. We trained the decoupled binary model from scratch without using the pretrained ternary model. The result is shown in the third contour plot of Figure 6. The best accuracy was 88.93% which is lower than the baseline result. Hence, we conclude that the proposed training scheme plays a major role for improving the performance of binary activation network.

6.2 BNN on ImageNet

We also compare BinaryDuo to other state-of-the-art BNN training methods. We used the BNN version of AlexNet (Krizhevsky et al., 2012) and ResNet-18 (He et al., 2016) on ImageNet (Deng et al., 2009) dataset for comparison. Details on training conditions are in Appendix D.

Table 2 shows the validation accuracy of BNN in various schemes. For fair comparison, we also report the model parameter size and computing cost at inference stage of each scheme. Computing cost (FLoating point OPerations) was calculated following the method used in Liu et al. (2018) and Wang et al. (2019). For ResNet-18 benchmark, we report two results with and without additional shortcut as proposed in Liu et al. (2018). Regardless of the existence of additional shortcut path, BinaryDuo outperforms state-of-the-art results with same level of parameter size and computing cost for both networks. BinaryDuo improved the top-1 validation accuracy by 4.8% for AlexNet and 3.2% for ResNet-18 compared to state-of-the-art results. CBCN (Liu et al., 2019) achieved

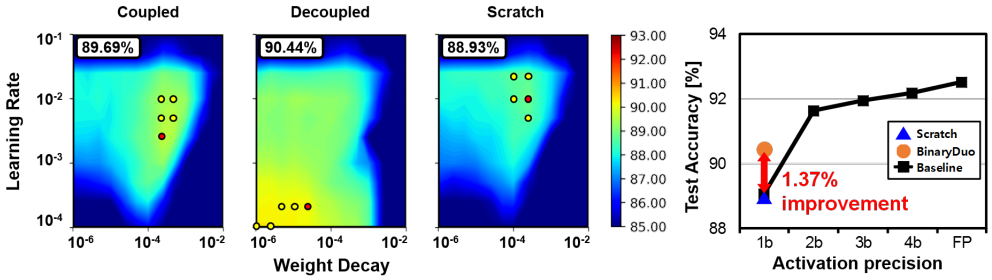

Figure 6: Training results of VGG-7 on CIFAR-10 dataset in the order of coupled ternary model, decoupled binary model and decoupled binary model trained from scratch (left). The best accuracy result is shown in the top left corner of each contour plot. Test accuracy of various models with different activation precision and training schemes (right).

Table 2: Classification accuracy of BNN in various schemes.

	AlexNet				ResNet-18			
	Accuracy		Params	Comp.	Accuracy		Params	Comp.
Network	Top-1	Top-5	(Mbit)	(FLOP)	Top-1	Top-5	(Mbit)	(FLOP)
BNN	41.8	67.1	62.3	82.3M	-	-	-	-
XNOR-Net	44.2	69.2	191	126M	51.2	73.2	33.3	164M
BNN+	46.1	75.7	191	126M	53.0	72.6	33.3	164M
Yang et al. (2019)	47.9	72.5	191	126M	53.6	75.3	33.3	164M
Bi-Real-Net†	-	-	-	-	56.4	79.5	33.3	164M
WRPN 2x	48.3	-	498	283M	-	-	-	-
ABC-Net	-	-	-	-	42.7	67.6	33.3	164M
Group-Net	-	-	-	-	55.6	78.6	33.3	164M
CBCN†	-	-	-	-	61.4	82.8	33.3	656M
Wang et al. (2019)†	-	-	-	-	59.9	84.2	36.7	183M
Ding et al. (2019)	47.8	71.5	191	126M	-	-	-	-
Bulat et al. (2019)	-	-	-	-	55.6	78.5	33.3	164M
PCNN†	-	-	-	-	57.3	80.0	33.7	164M
Bethge et al. (2019)†	-	-	-	-	57.7	80.0	33.3	164M
BinaryDuo	**52.7**	**76.0**	**189**	**119M**	**60.4**	**82.3**	**31.9**	**164M**
BinaryDuo(+sc)†	-	-	-	-	**60.9**	**82.6**	**31.9**	**164M**

† These schemes use ResNet models with additional shortcut proposed by Liu et al. (2018).

slightly higher top-1 accuracy than BinaryDuo but it requires more computing cost than BinaryDuo. CBCN uses similar amount of parameters and computing cost to that of the multi-bit network which uses 4-bit activations and 1-bit weights.

7 Conclusion and Future Work

In this work, we introduced the coordinate discrete gradient (CDG) for the gradient mismatch estimation. The cosine similarity between the proposed CDG and the coarse gradient explains the relationship between gradient mismatch problem and performance of a model much better than previous approaches. Based on the analysis using CDG, we proposed BinaryDuo which is a new training scheme for binary activation network. The proposed training scheme improves top-1 accuracy of BNN version of AlexNet and ResNet-18 for 4.8% and 3.2% compared to state-of-the-art results, respectively, by minimizing gradient mismatch problem during training. In the future, we plan to explore several techniques proposed for multi-bit network to improve the performance of BinaryDuo and to apply the proposed training scheme on other tasks such as speech recognition and object detection.

REFERENCES

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Joseph Bethge, Haojin Yang, and Christoph Meinel. Training accurate binary neural networks from scratch. In *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 899–903. IEEE, 2019.

Adrian Bulat, Jean Kossaifi, Georgios Tzimiropoulos, and Maja Pantic. Matrix and tensor decompositions for training binary neural networks. *arXiv preprint arXiv:1904.07852*, 2019.

Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5918–5926, 2017.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. *Proceedings of the 2nd SysML Conference*, 2019.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. Bnn+: Improved binary network training. *arXiv preprint arXiv:1812.11800*, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11408–11417, 2019.

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. *arXiv preprint arXiv:1908.05033*, 2019.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv: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.

Kaiming He et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4350–4359, 2019.

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.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pp. 2849–2858, 2016.

Darryl D Lin and Sachin S Talathi. Overcoming challenges in fixed point training of deep convolutional networks. *arXiv preprint arXiv:1607.02241*, 2016.

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

Chunlei Liu, Wenrui Ding, Xin Xia, Baochang Zhang, Jiaxin Gu, Jianzhuang Liu, Rongrong Ji, and David Doermann. Circulant binary convolutional networks: Enhancing the performance of 1-bit dcnns with circulant back propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2691–2699, 2019.

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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 722–737, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *International Conference on Learning Representations*, 2019.

Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: wide reduced-precision networks. *International Conference on Learning Representations*, 2018.

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.

Charbel Sakr, Jungwook Choi, Zhuo Wang, Kailash Gopalakrishnan, and Naresh Shanbhag. True gradient-based training of deep binary activated neural networks via continuous binarization. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2346–2350. IEEE, 2018.

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

Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

Ziwei Wang, Jiwen Lu, Chenxin Tao, Jie Zhou, and Qi Tian. Learning channel-wise interactions for binary convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 568–577, 2019.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pp. 2074–2082, 2016.

Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9127–9135, 2018a.

Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *International Conference on Learning Representations*, 2018b.

Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xiansheng Hua. Quantization networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7308–7316, 2019.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. *International Conference on Learning Representations*, 2019.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 365–382, 2018.

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

APPENDIX

A TRAINING CONDITIONS AND RESULTS FOR VGG-7 ON CIFAR-10

For the experiments in section 3.1, we trained various VGG-7 models with different activation precisions and width of layers on CIFAR-10 dataset. Width of a layer indicates number of neurons in fully-connected layers or number of channels in convolultion layers. The baseline VGG-7 model which we used consists of 4 convolution layers with 64, 64, 128 and 128 channels and 3 dense layers with 512, 512 and 10 output neurons. Max pooling layer was used to reduce the spatial domain size after the second, third and forth convolution layers. Full precision weights were used for all cases while the activation function layers except the last one were modified following the activation precision.

Each model was trained for 200 epochs with learning rate scaling by 0.1 at 120th and 160th epochs. We initialized models following He et al. (2015) and the mini-batch size was fixed to 256. We used Adam optimizer with decoupled weight decay (Loshchilov & Hutter, 2019) for easier hyper-parameter search. We tried 13 different exponentially increasing weight decay values and 10 different exponentially increasing initial learning rates. Test accuracy from at least two runs were averaged for each combination of weight decay and initial learning rate. Figure 7 shows the test accuracy of benchmark networks with various activation precisions.

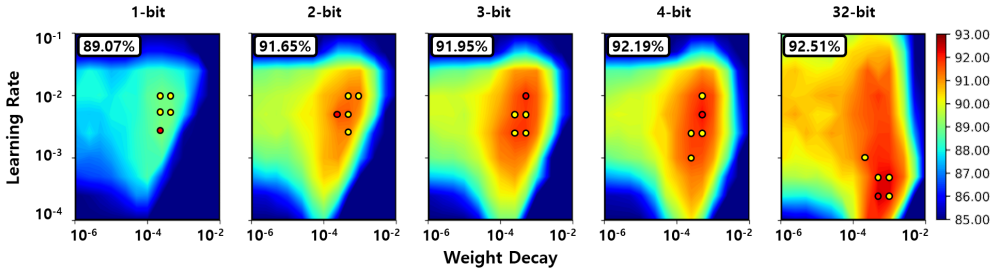

Figure 7: Training results of VGG-7 on CIFAR-10 dataset. 5 circles represent top 5 test accuracy points and the red circle is for the best result. The best test accuracy in each contour plot is shown on top of the contour plot.

B GRAPHICAL EXPLANATION ON CDG

In this section, we elaborate the difference between true gradient and CDG using bivariate loss function. Figure 8 shows graphical explanation of the concept of CDG. Each plot in the figure shows how to calculate the true gradient and CDG, respectively. The green contour plot in each case represents the given loss surface ($\mathcal{L}$) which is a function of w_1 and w_2. Red and blue lines in the edge shows the cross-section of the loss surface in direction of the dashed lines. True gradient at the given point can be derived by calculating the slope of loss surface in each direction. Similarly, CDG can be obtained by calculating how much the loss changes when we move to each direction by given step size (ε). Then, by calculating the cosine similarity between the two vectors, we evaluate how similar the directions of two vectors are.

C COSINE SIMILARITY BETWEEN CDG AND COARSE GRADIENT

C.1 STEP SIZE OF CDG

In order for CDG to point the steepest descent direction more accurately, the size of ε in Eq. 3 has to be carefully chosen. First, the ε has to be small enough to minimize the approximation error from non-linearity of the loss surface. At the same time, it has to be large enough to ignore the noise from rough and uneven stepwise loss surface caused by quantized activation. In addition, the optimal size of ε depends on the number of samples used for loss calculation. If the number of samples used for

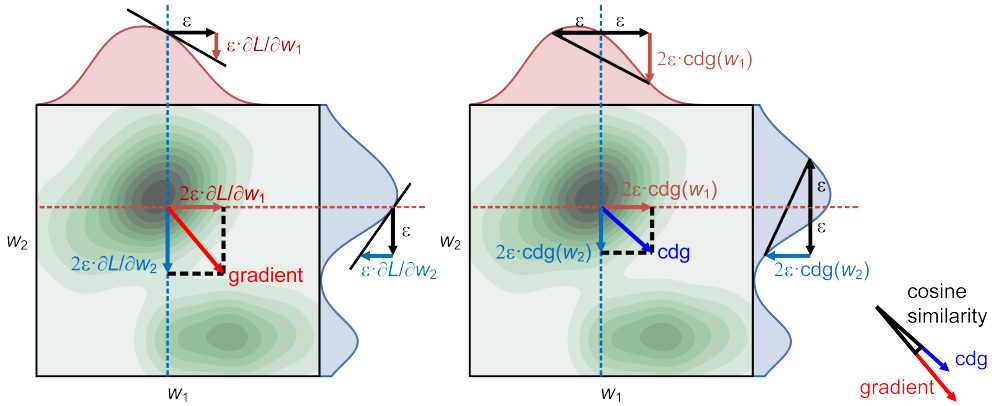

Figure 8: Graphical explanation of CDG using bivariate loss function.

the loss calculation is small, larger size of ε is preferred to minimize the noise from the rough and uneven loss surface (Figure 9a,b). On the other hand, if the number of samples used for the loss calculation is large, smaller size of ε is preferred to better estimate the steepest descent direction (Figure 9c,d). In our experiment, we used one million samples and ε of 0.001 which are in similar range to the dataset size and learning rate.

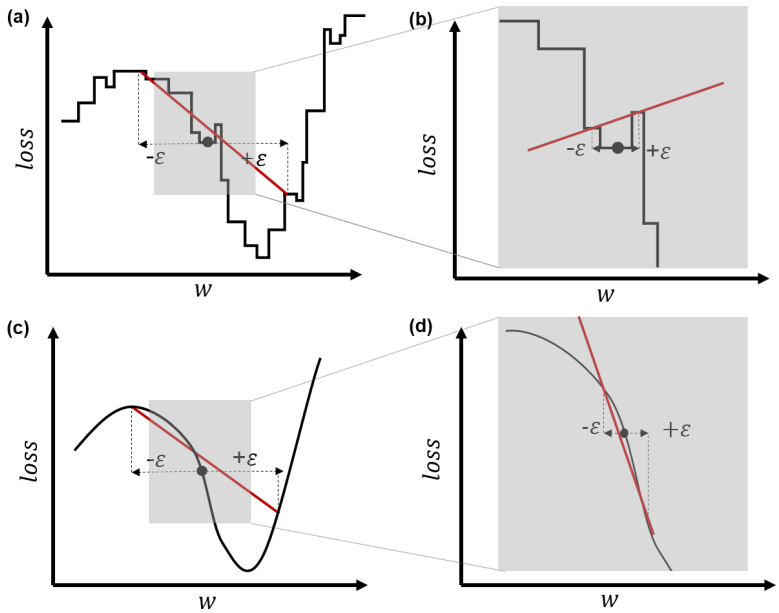

Figure 9: The examples of loss curve and coordinate discrete gradient in a single dimension with different scale of ε and the different number of samples used to calculate the loss curve. The number of samples used is small (a and b) and large (c an d). The size of ε is large (a and c) and small (b and d).

C.2 Algorithm for calculating cosine similarity

Algorithm 1 explains how we calculated the cosine similarity between coarse gradient and CDG in Section 4. First, we generate the dataset by sampling from the standard normal distribution in i.i.d manner. Then we initialize the weight for target model and evaluating model. The target model is used for calculating the target value for computing the loss value of evaluating model. The coarse gradient of each weight matrix is calculated by back-propagation with STE. For calculating the CDG

of a single element, we evaluate the loss at two parameter points $\boldsymbol{W}_i + \varepsilon \cdot e^{(j,k)}$ and $\boldsymbol{W}_i + \varepsilon \cdot e^{(j,k)}$, which are nearby points of current parameter point along the coordinate of the selected element. After evaluating the coarse gradient and CDG, we can estimate the gradient mismatch by calculating cosine similarity between those two.

Algorithm 1 Calculating the cosine similarity between the coordinate discrete gradient and the coarse gradient with respect to ε, f

Generate $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ from standard normal distribution $\mathcal{N}(0, 1)$
Initialize $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{l-1}, \boldsymbol{W}_1^*, \ldots, \boldsymbol{W}_{l-1}^* \in \mathbb{R}^{m \times m}, \boldsymbol{W}_l, \boldsymbol{W}_l^* \in \mathbb{R}^m$
Define the evaluating model $F(\boldsymbol{x}) = \boldsymbol{W}_l(f(\ldots, \boldsymbol{W}_2 f(\boldsymbol{W}_1 \boldsymbol{x})))$
Define the target model $F^*(\mathbf{x}) = \boldsymbol{W}_l^*(f(\ldots, \boldsymbol{W}_2^* f(\boldsymbol{W}_1^* \boldsymbol{x})))$
$\mathcal{L} \leftarrow \frac{1}{2n} \sum_{t=1}^{n} (F(\boldsymbol{X}_{:,t}) - F^*(\boldsymbol{X}_{:,t}))^2$
for $i = 1, 2, \ldots, l$ **do**
  # Calculate the coarse gradient using STE
  $\mathbf{C}_{i,:,:} \leftarrow \tilde{\nabla}_{\boldsymbol{W}_i} \mathcal{L}$
  # Calculate the coordinate discrete gradient
  for $j = 1, 2, \ldots, m$ **do**
    for $k = 1, 2, \ldots, m$ **do**
      $\mathcal{L}_+ = \frac{1}{2n} \sum_{t=1}^{n} (F(\boldsymbol{X}_{:,t}; \boldsymbol{W}_i + \varepsilon \cdot e^{(j,k)}) - F^*(\boldsymbol{X}_{:,t}))^2$
      $\mathcal{L}_- = \frac{1}{2n} \sum_{t=1}^{n} (F(\boldsymbol{X}_{:,t}; \boldsymbol{W}_i - \varepsilon \cdot e^{(j,k)}) - F^*(\boldsymbol{X}_{:,t}))^2$
      $\mathbf{D}_{i,j,k} \leftarrow (\mathcal{L}_+ - \mathcal{L}_-)/2 \cdot \varepsilon$
    end for
  end for
  # Calculate the cosine similarity
  $s_i \leftarrow \frac{\mathbf{C}_{i,:,:} \cdot \mathbf{D}_{i,:,:}}{|\mathbf{C}_{i,:,:}||\mathbf{D}_{i,:,:}|}$
end for
$s_{total} \leftarrow \frac{\mathbf{C} \cdot \mathbf{D}}{|\mathbf{C}||\mathbf{D}|}$
return $\{s_1, s_2, \ldots, s_l, s_{total}\}$

D IMAGENET TRAINING CONDITIONS AND TRAINING CURVE

For comparison with state-of-the-art BNN results, we used ImageNet (Deng et al., 2009) dataset which has 1.28M images for training set and 50K images for validation set. We initialized the coupled ternary model following He et al. (2015) for all cases and the mini-batch size was fixed to 256. Weights are also binarized following Rastegari et al. (2016) which uses layerwise scaling factor. We also clipped weights so that weights are always between -1 and +1. For AlexNet, we used Batch-normalization layer instead of local response normalization layer. Since binarization of weights and activations serves as strong regularizer, we used 0.1 for dropout ratio in AlexNet. The maximum training epoch was 100 for AlexNet and we used a learning rate starting at 1e-3 and decreasing by 1/10 at 41, 71 and 91 epochs. Weight decay of 2e-5 was used and Adam optimizer was used. When fine-tuning the decoupled model, we trained the model for 40 epochs. Initial learning rate was set to 2e-5 and multiplied by 0.1 at 16, 26 and 36 epochs. Weight decay was also lowered to 1e-6 when fine-tuning. For ResNet-18, we used shortcut type of 'B' as in other BNN papers. We used full precision shortcut path since its overhead is negligible as described in (Rastegari et al., 2016; Liu et al., 2018). Maximum number of training epoch was 120 and we did not use weight decay for ResNet-18. Initial learning rate was set to 5e-3 and multiplied by 0.1 at 71, 91 and 111 epochs. For fine-tuning, we also trained the model for 40 epochs. Learning rate was set to 2e-3 and decayed at 21, 31 and 36 epochs. Training curves for both AlexNet and ResNet-18 are shown in Figure 10.

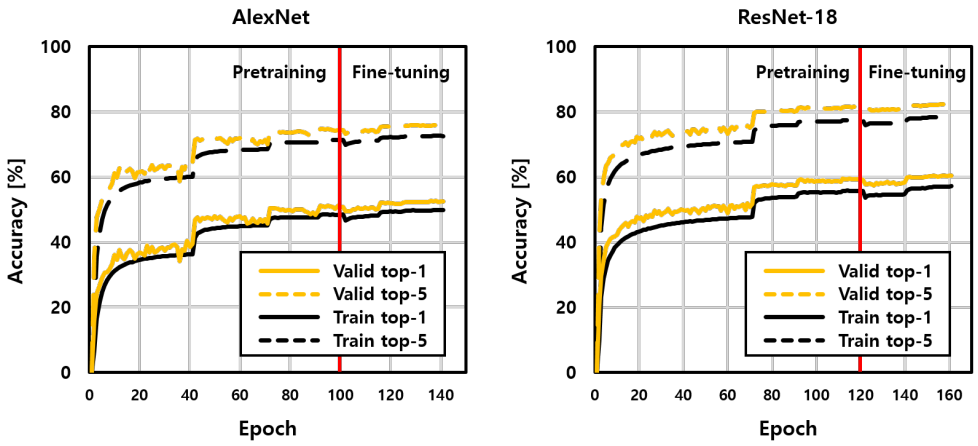

Figure 10: Training curves for AlexNet and ResNet-18 with BinaryDuo training scheme.

E COUPLED MODEL WITH HIGHER PRECISION

Based on the observation that the binary activation function suffers more from gradient mismatch problem than ternary activation function, we used a model with ternary activation function for coupled pretrained model. However, the proposed training scheme can also use the higher precision models as the pretrained model in the same manner. For example, a model with 2-bit activation function, which can be mimicked by three binary activations, can be used as the pretrained model. In this case, however, the model size of the 2-bit pretrained model should be even smaller than that of the ternary pretrained model since decoupling the 2-bit activation function increases the number of weights $3\times$. We evaluated such case on the same benchmark which was also used in Section 6.1.

Figure 11 shows the training results of the coupled 2-bit model and the corresponding decoupled binary model. Compared to the coupled ternary model reported in Section 6.1, the coupled 2-bit model achieved similar test accuracy (89.61%) which is also higher than baseline result. However, the performance of the binary model which was decoupled from the 2-bit model could not achieve a comparable performance to the value of the network decoupled from the ternary network which was reported in Section 6.1. We further trained with even lower learning rate and weight decay but we could not achieve better performance. This result was interesting because decoupling 2-bit model increases the number of weights in a model 3 times while decoupling ternary model doubles the number of weights. We suspect that the cause of this phenomenon is that the local minimum value that was found by the pretrained model becomes less useful when the model is more deformed through decoupling.

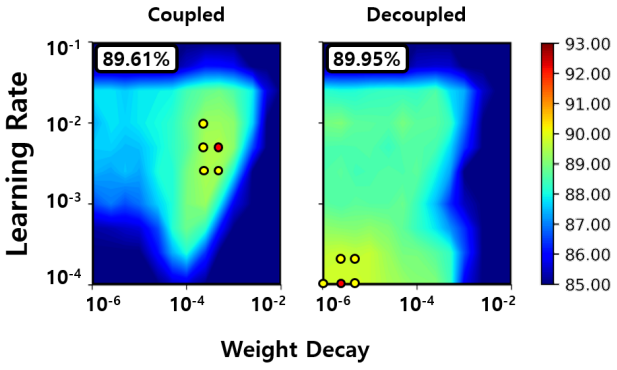

Figure 11: Training result of coupled 2-bit model and the corresponding decoupled model.

F BINARYDUO WITHOUT MODEL MODIFICATION

In this section, we introduce a variation of BinaryDuo that can keep the baseline model architecture even after the decoupling. While the proposed method in Section 5 uses the half-sized ternary model as the pretrained coupled model, this approach uses quarter-sized ternary model as the pretrained model. Let us call this approach as BinaryDuo-Q. BinaryDuo-Q quadruples the number of weights of a model when decoupling as shown in Figure 12. Suppose that the original baseline network has 4x4 configuration. BinaryDuo-Q reduces the width of all layers by half to make coupled ternary model. In the example case, the coupled model has 2x2 configuration as shown in the left side of Figure 12. When decoupling, one weight from the coupled model produces four weights in the decoupled model. While the decoupled model of BinaryDuo-Q has the same architecture as the original baseline model, the coupled ternary model has half number of parameters of the coupled ternary model used for BinaryDuo in Section 5. Figure 13 shows the training results of BinaryDuo-Q. As expected, the coupled ternary model with the quarter size of baseline model showed worse performance than the half-sized coupled model due to the less number of parameters. As a result, even though decoupling the quarter-size model improves the performance by 1.67%, the best test accuracy of the decoupled model using BinaryDuo-Q was 89.72% which is lower than that using BinaryDuo. Nevertheless, note that the result of BinaryDuo-Q is still better than baseline result.

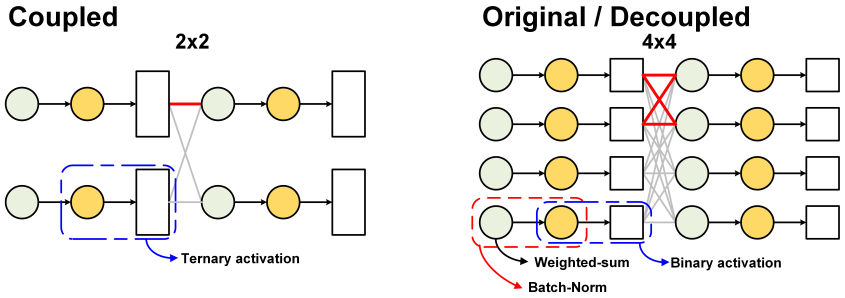

Figure 12: Example model architectures of coupled ternary and decoupled binary model when using BinaryDuo-Q. Coupled ternary model with 2x2 configuration is decoupled to binary model with 4x4 configuration.

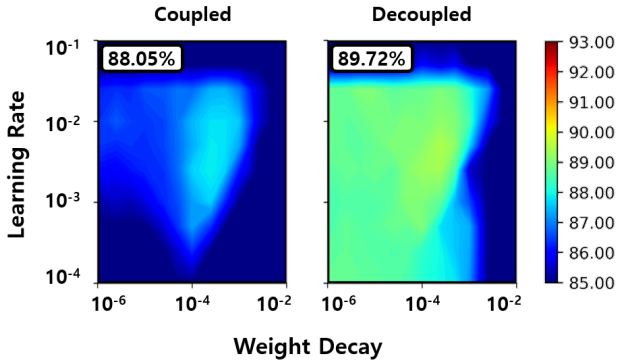

Figure 13: Training results of coupled ternary model and decoupled binary model when using BinaryDuo-Q.

G BETTER MULTI-BIT TRAINING

Recently, several works proposed to learn the interval and range of quantized activation function for higher accuracy (Jung et al., 2019; Choi et al., 2018; 2019; Zhang et al., 2018). Unfortunately, these techniques were not effective on binary activation networks. However, we use the multi-bit network as a pretrained model for binary activation network, and hence there is a chance of accuracy

improvement using such interval learning methods. In the future, we plan to explore such techniques to improve the performance of BinaryDuo.