BNN Pruning: Pruning Binary Neural Network Guided by Weight Flipping Frequency

Yixing Li, Fengbo Ren
Arizona State University, Tempe, AZ USA
E-mail: yixingli, renfengbo@asu.edu

Abstract— A binary neural network (BNN) is a compact form of neural network. Both the weights and activations in BNNs can be binary values, which leads to a significant reduction in both parameter size and computational complexity compared to their full-precision counterparts. Such reductions can directly translate into reduced memory footprint and computation cost in hardware, making BNNs highly suitable for a wide range of hardware accelerators. However, it is unclear whether and how a BNN can be further pruned for ultimate compactness. As both 0s and 1s are non-trivial in BNNs, it is not proper to adopt any existing pruning method of full-precision networks that interprets 0s as trivial. In this paper, we present a pruning method tailored to BNNs and illustrate that BNNs can be further pruned by using weight flipping frequency as an indicator of sensitivity to accuracy. The experiments performed on the binary versions of a 9-layer Network-in-Network (NIN) and the AlexNet with the CIFAR-10 dataset show that the proposed BNN-pruning method can achieve 20-40% reduction in binary operations with 0.5-1.0% accuracy drop, which leads to a 15-40% runtime speedup on a TitanX GPU.

Keywords— Neural network, binary, pruning

I. Introduction

In the rapid evolution of deep learning, neural network models have been growing from several to over a hundred layers for handling more complex tasks. Network models are often trained with powerful GPUs in the cloud or on stand-alone servers, and the trained models are then deployed to certain hardware platforms for performing inference. For cloud applications where neural network models are both trained and deployed in the cloud, computational complexity is typically a secondary concern as there is little gap in computing resources between the training and the inference stages. However, in many emerging Internet-of-things (IoT) applications where neural network models must be deployed onto resource-constrained edge devices [2] for performing real-time inference, the computational complexity of neural network models has become a major concern. Therefore, it is important to not only investigate how to build more compact neural network models that are friendly to hardware implementations but also rethink how to further compress compact neural network models for the efficient hardware implementations on resource-constrained edge devices.

The most compact form of deep neural networks are binary neural networks (BNNs) [3, 7, 18, 21]. BNNs are an extreme case of a quantized neural network, which adopts binarized representations of weights and exclusive NOR (XNOR) operations as binary convolution. By applying binary constraints on deep neural networks, the existing studies have shown up to 12x, 5x, 16x speedup on a CPU, GPU, and field-programmable gate array (FPGA) implementation [6, 7, 11], respectively, compared with its 16/32-bit floating-point counterpart. Another popular method to compress neural networks is pruning. Pruning methods help to remove insensitive weights and/or connections of a network. Since quantization and pruning are independent on each other, they could be jointly applied. Existing work has already shown promising results of combining a 4-bit quantization with pruning for a deep compression [5]. However, there has been no study that explores the pruning of BNNs. The major challenges of pruning BNNs are twofold. First, as both 0s and 1s are non-trivial in BNNs, it is not proper to adopt any existing pruning method of full-precision networks that interprets 0s as trivial. A new indicator is needed to identify the weights insensitive to network accuracy. Second, unstructured pruning can hardly result in any saving for BNNs as one has to introduce memory overhead to label the prunable weights that only have 1 bit. Therefore, we need a solution that avoids unstructured pruning.

In this paper, we propose a BNN-pruning method that uses the weight flipping frequency as an indicator to analyze the sensitivity of the binary weights to accuracy. Through experiments, we validate that the weights with a high weight flipping frequency, when the training is sufficiently close to convergence, are less sensitive to accuracy. To avoid unstructured pruning, we propose to shrink the number of channel in each layer by the same percentage of the insensitive weights to reduce the effective size of the BNN for further fine tuning. The experiments performed on the binary versions of a 9-layer Network-in-Network (NIN) [12] and the AlexNet [10] with the CIFAR-10 dataset [9] show that the proposed BNN-pruning method can achieve 20-40% reduction in binary operations with 0.5-1.0% accuracy drop, which leads to a 15-40% runtime speedup on a TitanX GPU. The source code is available on GitHub [1].

The key contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work that demonstrates the flexibility of and propose a generic method for pruning BNNs.

[1]https://github.com/PSCLab-ASU/BNNPruning

- We draw the insight that weight flipping frequency is an effective indicator of the sensitivity to accuracy in BNNs and validate its effectiveness to guide the binary weight pruning.

II. Related work

In this session, we discuss the related work of BNNs and pruning methods. In addition, we explain why the existing pruning methods of full-precision networks cannot be applied to BNNs.

A. Binary neural networks (BNNs)

Quantization is one of the most popular techniques that can be directly applied to compress a full-precision neural network model. Existing work has shown that with learned quantization, the precision of convolutional neural networks (CNNs) can be reduced to the level of 4-16 bits with merely no accuracy drop [20, 8]. Furthermore, [15] shows the feasibility of using a half-precision (FP16) data format to perform training, which can also accelerate the training process.

In the extreme case of reduced-precision neural networks, the weights and activations can be reduced to two states (1 bit), referred to as BNNs [3, 7, 18, 21]. The values of weights and activations in BNNs are constrained to (0, 1) or (-1, 1) depending on the preferred encoding scheme. In binary CNNs, a convolution operation can be simplified to the Boolean operation of XNOR and bit-count operations. As a result, BNNs have two direct impacts on hardware implementation. First, the use of extremely low-precision weights and activations enables the significant reduction of memory footprint for storing model parameters and intermediate results. Second, the use of Boolean operations significantly reduces the computational complexity, as well as the hardware implementation cost, thus greatly improves the energy efficiency of inference. These benefits are ideal for FPGA or application-specific integrated circuit (ASIC) implementations of BNNs, where customized hardware can be designed to perform the 1-bit operations efficiently [11]. On the other hand, CPUs and GPUs can also perform batches of binary convolutions in a bit-wise fashion with much improved efficiency [7, 6]. Overall, BNNs are highly suitable for efficient hardware mapping on CPUs, GPUs, FPGAs, or ASICs, which can bring significant savings in both memory and computation resources on edge devices.

BinaryConnect [3] is a partially binarized network with binarized weights but non-binarized activations. Binarized weights alleviate the memory bandwidth issue and allow BinaryConnect to substitute full-precision multiply-accumulate operations with full-precision additions only.

BinarizedNN [7] is a fully-binarized CNN with both with binarized weights and activations. The binarized weights and activations of BinarizedNN enable XNOR-based binary convolution operations. As a result, BinarizedNN is highly efficient to be mapped on CPUs, GPUs, FPGAs, and ASICs for inference.

XNOR-Net [18] is a more relaxed version of BNN. In addition to its binarized weights and activations, XNOR-Net also introduces full-precision coefficient matrices to enhance the capacity of the network, which improves the classification performance on the ImageNet dataset [4].

DoReFa-Net [21] adopts a mixture of 1-bit weights and 2-bit activations. The binarized weights in DoReFa-Net can result in the same amount of memory saving as the other BNNs. However, compared with BinarizedNN and XNOR-Net, DoReFa-Net doubles the computational complexity of the convolution operations with its 1-bit weights and 2-bit activations. Although, its convolution can still be decomposed to 1-bit operations, the total number of operations almost doubles, which results in a limited acceleration performance on CPUs and GPUs.

B. Pruning

In this paper, we refer to the pruning methods that involve multiple rounds of pruning and retraining in a iterative fashion as iterative pruning. We refer to the other pruning methods that prune a network based on optimization techniques without the need of retraining as optimization-based pruning.

Iterative pruning. The iterative pruning methods for full-precision neural networks [5, 14, 16] generally follow a three-step training pipeline, as shown in Fig. 1. If not starting from a pre-trained floating-point network, training one is the first step. The second step is to prune out redundant connections determined by certain threshold values. Intuitively, if the absolute values of weights are sufficiently small, their influence on the output values is fairly limited. The third step is to fine-tune the network to re-train the inherited weights. The second and third steps are repeated to further prune and fine-tune the network in an iterative fashion until satisfactory performance is achieved. Unfortunately, it is impossible to directly apply the iterative pruning method developed for full-precision networks to BNNs. Since 0s and 1s are both non-trivial in BNNs, the absolute values of the binarized weights are no longer a valid indicator of their sensitivity to accuracy thus cannot be used to guide the pruning of BNNs. Therefore, it is improper to adopt any iterative pruning methods for full-precision networks that interpret 0s as trivial in BNN pruning.

Optimization-based pruning. For the same reason, the optimization-based pruning methods [17, 19] that adopt a pruning objective function based upon the exact values of network weights cannot be applied to pruning BNNs [17] either. In case that the pruning objective function is based upon the separated masking arrays of weights [19], the pruned network models will have unstructured weights. As a result, their hardware implementations will require extra memory to store the flags of prunable weights as well as extra logic for manipulating the memory pointers to skip the corresponding computation, which creates difficulty in achieving actual runtime speedup on CPUs and GPUs.

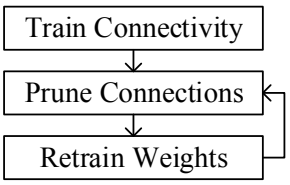

Fig. 1. The overview of the general pruning flow.

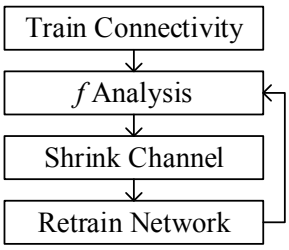

Fig. 3. The overview of the BNN pruning flow.

III. Methodology

To tackle the above-mentioned problems, we propose a novel BNN-pruning method that adopts weight flip frequency (f) as an indicator of the sensitivity of binary weights to accuracy to guide the pruning of BNNs. We empirically validate that a high and low f indicates a low and high sensitivity of binary weights to accuracy, respectively. Therefore, f is an effective indicator for identifying the binary weights that have a high criticality to pruning and can be pruned with a well-bounded accuracy drop.

A. Weight flip frequency (f)

In this paper, we define f as the flipping count of a weight during a specified last stage of training (from $epoch_{start}$ to $epoch_{end}$). In BNNs, weight flipping means that a weight value switches from 0/-1 to 1 or 1 to 0/-1. During the specified training interval, f is increased by 1 whenever the weight value flips. We hypothesize that, toward the end of the training, the last few percents of accuracy gain stems from the update of the weights with a high weight flipping frequency, and these weights have little influence on the accuracy that has already been established. Therefore, f can be used as the indicator of the sensitivity of binary weights to accuracy as well as their criticality to pruning . An f being equal to 0 or 1 means that the weights belong to this f group remain stable or become stable toward the end of the training, which implies that, if one flips their values, it is most likely to hurt the accuracy. On the contrary, an f being high or hitting the upper bound (the weight flips in every iteration) means that the values of this group of "noisy weights" most likely has little influence on the accuracy gain. Such behavior can be interpreted as that the training is trying fine-tune these "noisy weights" to further shape the decision boundary of classification but failed as they turned out to have little impact on the accuracy gain.

We conducted the experiments to validate our hypothesis. The experiments are conducted on the CIFAR-10 [9] dataset with a 9-layer binarized NIN [12]. We use the same network architecture as described in [12]. The training interval for f analysis is set as the epoch range that contribute to the last 1% of accuracy gain toward the end of training. We group the weights into three groups. The 1^{st} and 2^{nd} group of weights satisfies f=0/1 and f=2, respectively. The 3^{rd} group of weights contain the same amount of weights as the 1^{st} group, but are randomly selected from the entire weight set. Fig. 2 shows the accuracy comparison when we randomize different groups of weights during inference. Note that only weight randomization is performed in the experiments, and no fine-tuning of the weights is performed after the randomization. The results are averaged over ten trials. The experiment results show a 1.1% gap between the baseline model and the one with the 1^{st} group of weights randomized. Also, randomizing the 1^{st} group of weights achieves even lower accuracy than randomizing the same amount of randomly selected weights (the 3^{rd} group).

These results indicate that the 1^{st} group of weights, where f is equal to 0 or 1, is the most sensitive to the final accuracy rate, while the 2^{nd} group of weights with a high f has little impact. This validates our hypothesis and suggests that $f \geq 2$ is a viable threshold for identifying the prunable binary weights.

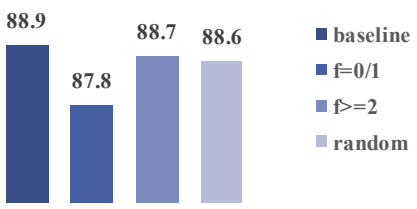

Fig. 2. Accuracy comparison for randomizing different groups of weights.

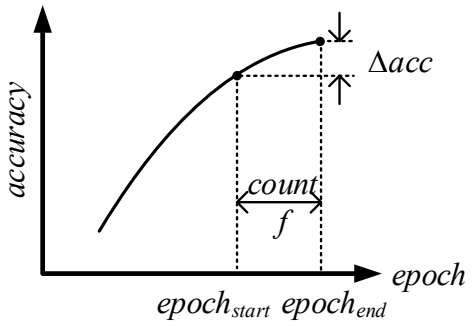

Fig. 4. The illustration of training interval for f analysis.

TABLE I

THE LAYER-WISE BNN-PRUNING RESULTS OF THE BINARIZED NIN AT EACH ITERATION.

layer	filter	N^{th} iteration								
		baseline	1st		2nd		3rd		4th	
			p_L/%	channels	p_L/%	channels	p_L/%	channels	p_L/%	channels
conv2d	5x5	192	N/A	192	N/A	192	N/A	192	N/A	192
binconv2d	1x1	160	5.0	152	5.9	143	0.7	142	3.5	137
binconv2d	1x1	96	5.2	91	5.5	86	1.2	85	4.7	81
pooling										
binconv2d	5x5	192	1.5	189	1.1	187	0.5	187	1.0	185
binconv2d	1x1	192	9.9	173	5.8	163	3.1	158	6.3	148
binconv2d	1x1	192	3.1	186	2.7	181	0.6	180	2.2	176
pooling										
binconv2d	3x3	192	1.6	189	1.1	187	0.5	187	1.1	185
binconv2d	1x1	192	4.2	182	4.9	173	1.2	171	3.5	165
conv2d	1x1	10	N/A	10	N/A	10	N/A	10	N/A	10
pooling										
GOPs (M)	220		206		193		190		180	

B. **BNN compression flow.**

The overall flow of BNN pruning is shown in Fig. 3, and Fig. 4 illustrates the training interval during which f analysis shall be performed. Δacc is the accuracy drop tolerance defined by the user. $epoch_{end}$ is the final epoch of the training phase. $epoch_{start}$ is the starting point of f analysis, where the accuracy at $epoch_{start}$ is Δacc lower than the final accuracy at $epoch_{end}$.

The first step in the BNN-pruning method is to train a BNN from scratch. In the case of a pre-trained BNN model is available, the last stage of training (from $epoch_{start}$ to $epoch_{end}$) needs to be performed again for the purpose of f analysis.

The second step to analyze the weight flip frequency – f. One shall log the statistics of f for each weight during the training interval from $epoch_{start}$ to $epoch_{end}$. Subsequently, one shall calculate the portion of insensitive weights ($p_L\%$) in the L^{th} layer that satisfy $f \geq 2$ for each layer.

The third step is to reduce the size of the BNN by shrinking the number of channels in the L^{th} layer by $p_L\%$, and the forth step is to retrain the network at the reduced size. One shall repeat the second, third, fourth steps in a iterative fashion until the maximum percentage of insensitive weights ($p_L\%$) is close to 0, which indicates that there is no room for further compression.

Preserving the pruned architecture and fine-tune the inherited weights in BNN pruning is not recommended for two main reasons. First, the insensitive weights are found to be sparsely distributed and unstructured in our experiments. Unstructured pruning can hardly result in any saving for BNNs as one has to introduce memory overhead to label the prunable weights that only have 1 bit as well as extra logic for manipulating the memory points to skip the corresponding computation. Second, prior work shows that retraining inherited weights in conventional pruning methods typically does not produce better performance than simply training a smaller model from scratch [13]. Therefore, we recommend reducing the size of the BNN network at each layer guided by the f analysis and $p_L\%$ and retraining the smaller BNN model in BNN pruning. Comparing to the intuitive approach of exploring the appropriate network size through binary search, the proposed BNN-pruning method can quantitatively analyze the layer-wise redundancy to effectively reduce the BNN size, which greatly reduces the search space and time.

IV. **Experiment**

We tested the proposed BNN-pruning method with the CIFAR-10 [9] dataset on two BNNs: the binary versions of a 9-layer binarized NIN [12] and the AlexNet[10]. The binarization method used in this paper is introduced by the XNOR-net [18]. The source code we build is based on [1].

The first set of experiments is conducted on the binarized NIN. The network architecture used in our experiment is shown in Table I. Table I also shows the layer-wise pruning results in four iterations of BNN pruning. Note that the 1^{st} and the last layer are floating-point layers, and we do not apply any pruning on them. In each iteration, our target Δacc is set to 0.5%. After four iterations of BNN pruning, the final network model has a 20% reduction in GOPs (Giga operations). The accuracy of the baseline model is 86.5%, and the accuracy of the final network model is subject to the target accuracy drop of 0.5%. Compared with directly performing a binary search for the appropriate network size, the proposed BNN-pruning method can provide a layer-wise, quantitative guideline to effectively shrink the BNN size. For the runtime evaluation, we tested the layer-wise runtime of the binarized NIN using an optimized binary kernel on an NVIDIA TitanX GPU. The final network model after BNN pruning leads to a 15% runtime speedup as comparing to the baseline model.

TABLE II

EXPERIMENT RESULTS OF BNN PRUNING.

Arch.	GOPs reduction	Speedup	Δacc
NIN	20%	15%	0.5%
AlexNet	40%	25%	1.0%

In addition, we tested the proposed BNN-pruning method on the binarized AlexNet with the CIFAR-10 dataset [9]. In this set of experiments, we set the target Δacc to 1.0%. After three iterations of BNN pruning, the final network model results in a 40% GOPs reduction and a 25% runtime speedup on an NVIDIA TitanX GPU, subject to the 1.0% target accuracy drop. The BNN pruning results and their impact on runtime speedup are summarized in Table II. Overall, the experiment results show that using the weight flipping frequency as the indicator of weight sensitivity to guide BNN pruning can lead to 20-40% GOPs reduction and 15-25% runtime speedup at the limited cost of 0.5-1.0% accuracy drop.

V. Conclusion

In this paper, we present a study that explores the weight redundancy in BNNs and propose a generic solution for BNN pruning. The weight flipping frequency f is an effective indicator of the sensitivity of binary weights to accuracy and their criticality to pruning. Reducing the BNN size by shrinking the number of channels in the L^{th} layer by a factor of p_L% is the key to effectively removing redundancy in BNN pruning.

VI. Acknowledgement

This work is supported by an NSF grant (IIS/CPS-1652038) and an unrestricted gift (CG#1319167) from Cisco Research Center. The computing infrastructure used in this work is supported by an NFS grant (CNS-1629888). The four GPUs used for this research was donated by the NVIDIA Corporation.

References

[1] `https://github.com/jiecaoyu/XNOR-Net-PyTorch`.

[2] Saman Biookaghazadeh, Ming Zhao, and Fengbo Ren. "Are FPGAs suitable for edge computing?" In: *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*. 2018.

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

[4] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[5] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *International Conference on Learning Representations* (2016).

[6] Yuwei Hu et al. "Bitflow: Exploiting vector parallelism for binary neural networks on cpu". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2018, pp. 244–253.

[7] Itay Hubara et al. "Binarized neural networks". In: *Advances in neural information processing systems*. 2016, pp. 4107–4115.

[8] Raghuraman Krishnamoorthi. "Quantizing deep convolutional networks for efficient inference: A whitepaper". In: *arXiv preprint arXiv:1806.08342* (2018).

[9] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

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

[11] Shuang Liang et al. "FP-BNN: Binarized neural network on FPGA". In: *Neurocomputing* 275 (2018), pp. 1072–1086.

[12] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in network". In: *arXiv preprint arXiv:1312.4400* (2013).

[13] Zhuang Liu et al. "Rethinking the value of network pruning". In: *arXiv preprint arXiv:1810.05270* (2018).

[14] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. "Thinet: A filter level pruning method for deep neural network compression". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5058–5066.

[15] Paulius Micikevicius et al. "Mixed precision training". In: *International Conference on Learning Representations* (2018).

[16] Pavlo Molchanov et al. "Importance estimation for neural network pruning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11264–11272.

[17] Pavlo Molchanov et al. "Importance estimation for neural network pruning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11264–11272.

[18] Mohammad Rastegari et al. "Xnor-net: Imagenet classification using binary convolutional neural networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 525–542.

[19] Ruichi Yu et al. "Nisp: Pruning networks using neuron importance score propagation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9194–9203.

[20] Dongqing Zhang et al. "Lq-nets: Learned quantization for highly accurate and compact deep neural networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 365–382.

[21] Shuchang Zhou et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients". In: *arXiv preprint arXiv:1606.06160* (2016).