Simulate-the-hardware: Training Accurate Binarized Neural Networks for Low-Precision Neural Accelerators

Jiajun Li*†, Ying Wang*, Bosheng Liu*†, Yinhe Han*, and Xiaowei Li*†
*Cyber Computing Laboratory, State Key Laboratory of Computer Architecture,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
†University of Chinese Academy of Sciences, Beijing, P.R. China
{lijiajun01,wangying2009,liubosheng,yinhes,lxw}@ict.ac.cn

ABSTRACT

This work investigates how to effectively train binarized neural networks (BNNs) for the specialized low-precision neural accelerators. When mapping BNNs onto the specialized neural accelerators that adopt fixed-point feature data representation and binary parameters, due to the operation overflow caused by short fixed-point coding, the BNN inference results from the deep learning frameworks on CPU/GPU will be inconsistent with those from the accelerators. This issue leads to a large deviation between the training environment and the inference implementation, and causes potential model accuracy losses when deployed on the accelerators. Therefore, we present a series of methods to contain the overflow phenomenon, and enable typical deep learning frameworks like Tensorflow to effectively train BNNs that could work with high accuracy and convergence speed on the specialized neural accelerators.

CCS CONCEPTS

• **Computing methodologies → Speech recognition**; • **Computer systems organization →** *Neural networks*;

KEYWORDS

Binarized Neural Networks, Overflow, Containing, Simulating

ACM Reference Format:
Jiajun Li*†, Ying Wang*, Bosheng Liu*†, Yinhe Han*, and Xiaowei Li*†. 2019. Simulate-the-hardware: Training Accurate Binarized Neural Networks for Low-Precision Neural Accelerators. In *24th Asia and South Pacific Design Automation Conference (ASPDAC '19), January 21–24, 2019, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3287624.3287628

1 INTRODUCTION

Attributed to the rise of the edge devices, there is a growing amount of literature that recognizes the importance of training Binarized Neural Networks (BNNs) for the specialized accelerators. Low power devices benefit from the advantages of BNNs including low storage requirement and multiply-free computation. Concerning the issues of low power and low memory, most researchers aim to simplify the architecture of the specialized accelerator with BNNs that induces only very low bit-width operations. However, training BNNs and minimizing the accuracy losses caused by the low data precision is not that easy especially for the specialized neural accelerators. There are many prior works that study how to effectively train highly accurate BNNs for general purpose devices like CPUs and GPUs. Courbariaux et al. [3] proposed the straight-through estimator (STE) to calculate the gradient of BNNs; Rastegari et al. [11] proposed XNOR-Net where both the weights and the activation values are binary; Zhou et al. [19] used low bit-width gradients in the backward propagation; Zhou et al. [18] proposed iterative and incremental training method; Hu et al. [7] adopt hash-searching to train BNNs. However, straightforwardly applying these works or the open-source codes to specialized neural accelerators will lead to a considerable drop of model accuracy. The reason is the inconsistency between the training environment (typically on GPU or CPU with full-precision 32-bit or 64-bit representation) and the inference of BNNs that runs on specialized neural accelerators with precision-limited fixed-point operators. In the BNN accelerators, the low bit-width operators tend to cause frequently severe overflow phenomenon because the numerous matrix multiplications in BNNs usually require a large number of additions. In this situation, overflow and operation rounding due to limited precision will cause a large deviation while deploying a BNN in the specialized accelerator of BNNs, even leading the inference to be incorrect on the accelerator. As shown in Fig. 1, this deviation is produced between the inference of specialized accelerator and the inference of the simulator due to a lack of simulating the phenomenon of overflow and operation rounding in the training environment, e.g. Tensorflow running on GPUs. According to our observation in the experiments, this deviation is able to be solved by containing and simulating the overflow and rounding the activation value during the training stage. Therefore, training effective BNNs for specialized accelerator involves two difficult problems, namely the overflow containment, and the overflow/rounding simulating and the training after the overflow containment. To our best knowledge, though using binary weight representation, the existing specialized accelerators usually use surplus

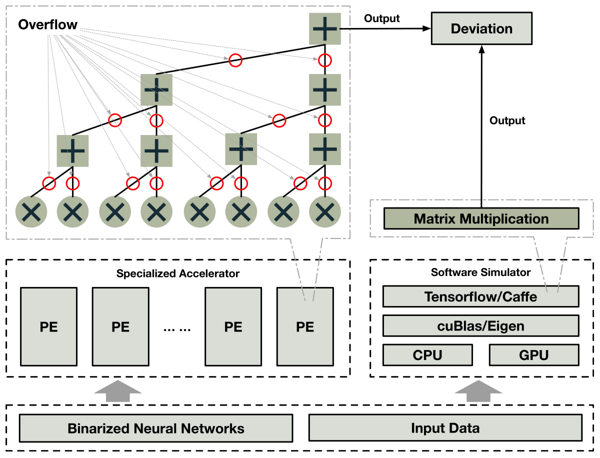

Figure 1: The deviation is caused by the overflow between the low-precision neural accelerators and the simulator.

bit-width to encode the activation result or the partial results during the inference so that the overflow containment is unnecessary, but the surplus bit-width will increase the overhead of energy, storage, and computation. Additionally, many existing tools including cuBlas and Eigen cannot directly simulate the overflow phenomenon in the inference, so that we are unable to use these tools to train BNNs directly. Tensorflow has a module for quantizing the parameters of neural networks into short fixed-point code without accuracy loss, and enable accurate BNN inference on CPU or GPU or even neural accelerators with high-precision adders (16-bit or 32-bit). However, it cannot simulate the overflow/rounding of matrix operations caused by the low-precision operators during the network inference, and the overflow is proved to have poor accuracy in BNN accelerators in our experiments. Our work is to solve the problem of inconsistent training for the specialized BNN accelerator that adopts low precision adders and activation format.

Our main contributions are the following.

- Overflow containment: we propose a series of methods for the specialized low-precision accelerator, including the designs of 1) a hardware-friendly normalization layer that is at least 1.5X faster than Batch Normalization; 2) a novel Small-Pipeline Rule (SPR) that makes the loss of the BNNs as Low as 3% in the accuracy; 3) an aggregated convolutional operation that utilizes the features of the normalization layer to reduce the overflow in the convolutional operation.
- Overflow/rounding simulation: we design a lossless method to simulate the overflow/rounding with GPU that is about 100X faster than the vanilla method, and only 80.75% slower than the original method. Section 4.4 will describe vanilla method and original method. Then we propose a new regularization term that the accuracy is 12% higher than the method in [15].

2 CONTAINING

This section has three parts to realize respectively the overflow containing. The normalization layer normalize the activation values to reduce the waste of the bit-width. Further, the Small-pipeline Rule constrains the overflow in the procedure of computation. Finally, an aggregated convolutional operation contains the overflow with the aid of the normalization layer.

2.1 Normalization Layer Design

A suitable normalization layer is necessary for training a deep neural network, such as Batch Normalization [8] (BN) that is able to improve the generalized ability and avoid the overfitting. More importantly, the normalization layer is able to reduce the overflow and we will give the reason in the next paragraph. However, deploying the BN layer in accelerators increases the computational burden and the design cost for the specialized low-precision accelerator without multiplier and divider to conduct standard BN operations. Thus, we adopt a simpler normalization that has only two parameters includes a divisor $\sigma' = 2^n$ where n is natural number and a subtraction μ that represents mean. In fact, we replace approximately the standard deviation with the $\sigma' = 2^n$. This normalization function is deterministic as below:

$$Norm(X) = \frac{X - \mu}{\sigma'} \tag{1}$$

The subtraction μ moves the inputs to be distributed around 0 to better utilize the operation bit-width. For example, if the range of data representation is $[-128, 127)$ and the inputs are (128, 256, 200), the overflow will happen. But if the inputs minus 128 to shift around 0, the overflow will be contained. In Eq. (1), the parameters of $Norm(X)$ for each normalization layer are different fixed-point values. We replace approximately the standard deviation with the $\sigma' = 2^n$ because the dividers can be replaced by the shift operators in the accelerator. This step of normalization layers only reduces a part of the overflow so that the overflow still exists during the inference. The next subsection, we propose a novel method that reduces the overflow.

2.2 Small-pipeline Rule

The small-pipeline rules (SPR) guides us to design the structure of BNNs with less frequent overflow. BNNs already restrict the weights and the activation values to +1 or -1. Therefore, the overflow is related closely to the kernel size and the number of feature maps in the inference of BNNs. The SPR aims to restrict these three factors, as shown in Eq. (2), where w, h, c, $bitwidth$, and η denote the width of kernels, the height of kernels, the number of feature maps, the bit-width of adders, and the tolerance factor of the overflow, respectively. Please see Fig. 2, it visualizes Eq. (2) helping to understand this issue.

$$w \cdot h \cdot c \leq (1 + \eta) \cdot 2^{bitwidth} \tag{2}$$

Currently, popular DNNs (Deep Neural Networks) [5, 12, 14] tend to go deeper so that the small kernel is prevalent. For example, imposing a regularization term on the 7×7

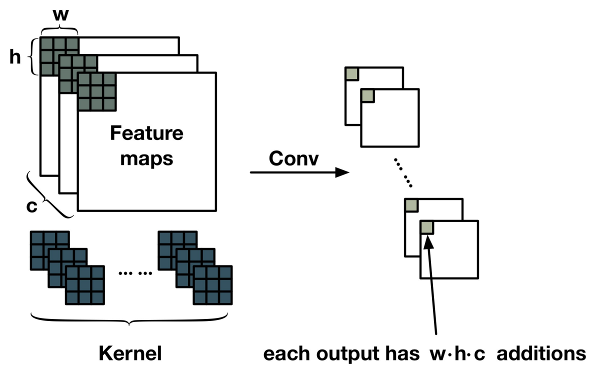

Figure 2: Additions for each output pixel in BNNs.

convolutional filter equals three stacked 3×3 convolutional filters. That is to say, BNNs merely use the small kernel to reach the effects of the big kernel. Using the small kernel gives more room for feature maps to bring more the capability for BNNs, leading to a higher accuracy. If following completely SPR as Eq. (2) to restrict the kernel size and the number of feature maps, the overflow will almost not happen.

2.3 Aggregated Convolutional Operation

In this subsection, we propose an aggregated convolutional operation to reduce the overflow. The operation aggregates one activation layer and one normalization layer into one convolutional layer as a new layer. The operation can contain the activation values because the normalization layer has a division operation. As shown in Eq. (6), due to the linear property of HardTanh [3] and ReLU [4], we can apply Distributive Law of Multiplication to distribute the division operation of the normalization layer to the convolutional layer, where $W_{ij'}$ denotes the grouped weights, $x_{ij'}$ denotes the grouped inputs, and G is the number of the groups. There is an example to explain the computational process of aggregated convolutional operation. Suppose that $\sigma' = 2^4$, $\mu = 0$, $G = 4$ and the convolutional layer uses an 8-bit adder that the output range is $[-128, 127]$. For floating-point operation, $Conv(X) = 512$, $ReLU(512) = 512$, and $Norm(512) = 128$; for the original convolutional operation, we have $Conv(X) = 127$, $ReLU(127) = 127$, and $Norm(127) = 31$; for the aggregated convolutional operation, we have $AC_{ReLU}(X) = 127/4 + 127/4 + 127/4 + 127/4 = 31 + 31 + 31 + 31 = 124$ because of $G = 4$, where the aggregated convolutional operation denotes as $AC_{ReLU}(x) = Norm(ReLU(Conv(x)))$, $Conv(X)$ represents the convolution operation, $ReLU(X)$ is a activation function, $Norm(X)$ is the mentioned normalization layer of Section 2, and GP is a grouped function that . As shown in the example, our method still induces a few acceptable deviations, but our method reduces the most overflow. The form of HardTanh is given directly in Eq. (7). The two activation functions are given in Eq. (4) and Eq. (3).

$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3}$$

$$HardTanh(x) = \begin{cases} 1, & x \geq 1 \\ x, & -1 \leq x < 1 \\ -1, & x < -1 \end{cases} \tag{4}$$

$$GP = \sum_{g}^{G} \frac{1}{\sigma'} \sum_{i} \sum_{j} W'_{ij} x'_{ij} \tag{5}$$

$$AC_{ReLU}(x) = \begin{cases} GP - \frac{\mu}{\sigma'}, & GP \geq 0 \\ 0, & otherwise \end{cases} \tag{6}$$

$$AC_{HT}(x) = \begin{cases} \frac{1}{\sigma'} - \frac{\mu}{\sigma'}, & GP \geq \frac{1}{\sigma'} \\ GP - \frac{\mu}{\sigma'}, & -\frac{1}{\sigma'} \leq GP < \frac{1}{\sigma'} \\ -\frac{1}{\sigma'} - \frac{\mu}{\sigma'}, & \frac{1}{\sigma'} < GP \end{cases} \tag{7}$$

3 SIMULATING

The overflow cannot be completely erased. That is to say, due to the inconsistency between the training environment and the inference substrate, we still need to simulate the overflow. The inference is greatly affected by the overflow containment so that the training and the regularization term have some adjustments.

3.1 Simulating Overflow/Rounding

When the overflow is inevitable, simulating overflow ensures the computational consistency between the hardware and the software. Existing software tools, such as cuBlas, Eigen, Caffe, and Tensorflow optimize significantly the computational speed of matrix operations. However, these tools cannot completely simulate the overflow/rounding for the specialized low-precision accelerator. Thus, we propose a method to utilize these tools as much as possible. As shown in Fig. 3, A and B are the operating matrices of $W_a \times H_a$ and $W_b \times H_b$, respectively. Hadamard product is known as Element-wise product of matrices. Adder tree is a generic procedure. The specific operation of Adder tree is determined by the specific low-precision accelerator. Our method uses a series of hand-crafted matrix operators to replace the original matrix multiplication to simulate the overflow/rounding. Because the overflow/rounding appear inside the matrix multiplication, the importance of simulation is to expose the results of every multiplication of the matrix product so that we are able to simulate the overflow/rounding for every multiplication operation in training. As shown in Fig. 3, we divide equally the matrix multiplication into two main parts that are implemented as the Hadamard product and the Adder tree. And these two main parts still make up of matrix operations so that we can simulate with existing tools. Besides, our method is general in some degree because we can design a specific Adder tree to simulate the overflow/rounding according to the specific low-precision accelerators

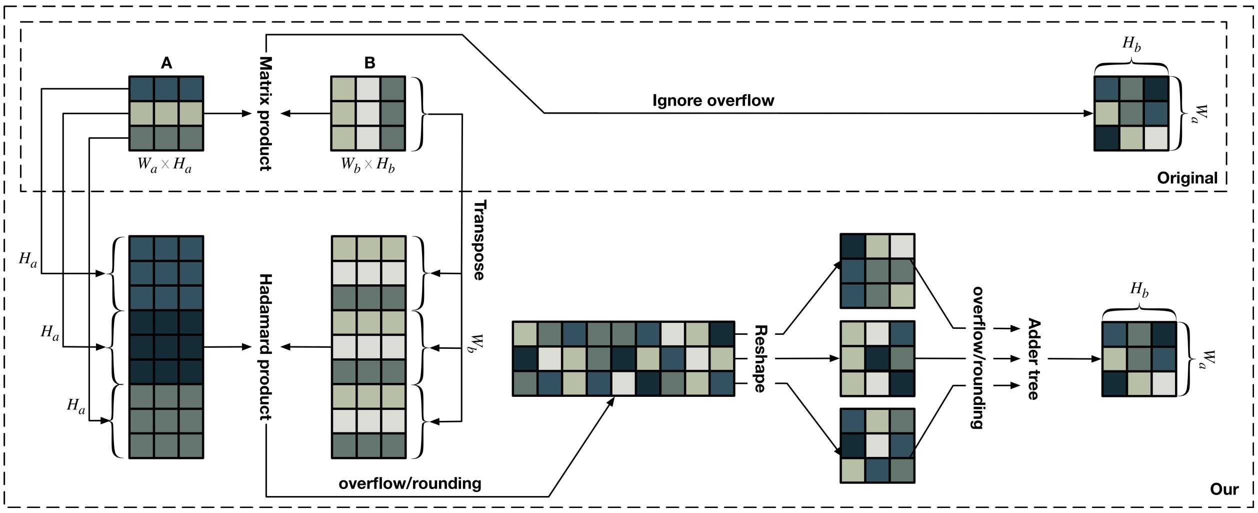

Figure 3: Simulating the overflow/rounding with GPU.

3.2 Training BNNs

The effective method of training BNNs [3] is that binarized function applies the sign function to the inference computation and the gradient is calculated by the straight-through estimator (STE). The methods [2, 11, 19] are basically the same as [3]. The methods proposed in [7, 18] are not easy to reproduce. Thus, we determine to use the method in [3]. However, after containing the overflow/rounding, the loss of training BNNs fail to converge on a minimum point. We reason that the failure may be caused by the overflow/rounding, which brings in the deviation affecting the training.

3.3 Regularization Term

The regularization term helps to provide the prior knowledge of BNNs that binarized weights are {+1,-1} for the training. Wei Tang et al. proposed a regularization term [15] for training BNNs. The term is defined in Eq. (8), where $Loss(W, b)$ is the Loss function, L is the total number of layers, $W_{ij}^{(l)}$ is the weight, b is the offset, and λ is the regularization coefficient.

However, as shown in Fig. 4, this regularization term has two disadvantages. The first disadvantage is that it cannot confine the weights to {+1, -1} as expected. The second disadvantage is that the weights greater than 1 or less than -1 will make the loss function to output probably a negative value. Thus, we propose a new regularization term based on the cosine function. Our regularization term obtains higher accuracy than the method in [15]. The new regularization term is defined in Eq. (9), where $cos(x)$ is the cosine function.

$$J(W,b) = Loss(W,b) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{N_l} \sum_{j=1}^{M_l} (1 - (W_{ij}^{(l)})^2) \quad (8)$$

$$J(W,b) = Loss(W,b) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{N_l} \sum_{j=1}^{M_l} (cos(\pi W_{ij}^{(l)}) + 1) \quad (9)$$

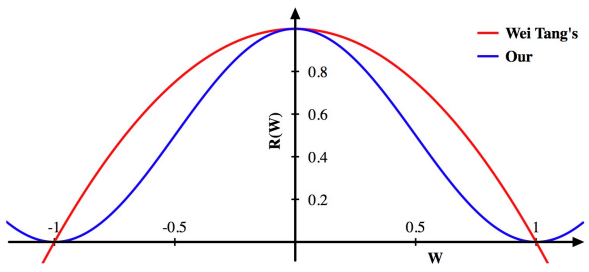

Figure 4: The comparison between the regularization term of Wei Tang et al. and our regularization term.

4 EXPERIMENTS

If not stated, all of the experiments adopt Cifar10 [9] as the dataset, train the models on GTX1080, and use LSTM with two layers of 128 hidden cells each. The weights of BLSTM adopts 1 bit and the activations are represented with 8 bits. The accumulator and adder adopt 16 bits. Drawing on the experience of this work and [10, 13, 16] to research and development, we have done 1.0 version BNNs accelerator. And we will present that work of accelerator in the near future.

4.1 Normalization Layer

Table 1 shows the experimental results are as the expectation of the design. Our normalization layer compromises a bit of accuracy, but it improves a lot of computational speed. This improvement benefits from hardware-friendly shift operation and the simple normalization operation. Losing 10% accuracy is able to increase 50% speed, which also can reduce the chip area, and cut the design costs due to the simpler architecture design.

Table 1: Quantized Batch Normalization vs. Batch Normalization.

	Batch Norm.	Our
Accuracy	82.32%	72.30%
FPS	4060.45	6097.35

4.2 Regularization Term

This experiment adopts AlexNet of BNNs, where the learning rate is 0.001. We use the mentioned normalization layer of Section 2. During the training, Our method also convergence fast than the method in [15]. The accuracy of the method in [15] stops early to increase, which is probably due to the fact that $J(W, b)$ less than zero cause this phenomenon. As shown in Fig. 4, the $R(W)$ is approximate to negative values if the weights are within $(-\infty, -1] \cap [+1, +\infty)$. Table 2 shows two regularization terms.

Table 2: The comparison of two regularization terms.

	Wei Tang et al.	Our
Accuracy	64.55%	72.30%

4.3 SPR Performance

Following SPR (η is 0, *bitwidth* is 8 without fraction part, *lr* is 1.0e-07), the BNNs is designed as $K3S1C16 - K3S1C16 - K3S1C16 - K3S1C16 - K3S1C16 - K3S1C16 - K3S1C16 - K3S1C16 - Fc10$, where K is for the kernel size, S is for the stride and C is for the output channels. The model is designed by following SPR, which is compared with AlexNet without SPR. If let $\eta = 0.25$, the block of the BNN is designed as K3S1C20. If let $\eta = 0.5$, the block is designed as K3S1C24. As shown in the Table 3, with SPR, the accuracy loss is significantly reduced after data representation transformation. But if the value of η is not appropriate, it will cause too many overflows leading the accuracy to drop. Thus, the tolerance coefficient η is able to be slightly greater than zero. It means SPR is able to ignore a small amount of the overflow because all the elements of the matrix in the Hadamard product are +1 or -1 with a very low probability.

Table 3: Performance of SPR in the transformation of model data representation.

			SPR	
	AlexNet	$\eta = 0.25$	$\eta = 0$	$\eta = 0.5$
Float point	75.35%	69.13%	68.85%	69.95%
Fixed point	42.03%	67.22%	66.98%	64.30%

4.4 Simulating Overflow/Rounding

Undoubtedly, simulating the overflow/rounding in matrix multiplication drastically increases the training time. As shown in Table 4, where batch size is 128, our method is 80.75% slower than the original method, but the original almost cannot simulate overflow/rounding. Beside, our method is about 100X faster than the vanilla method. The vanilla method can simulate overflow/rounding, but it cannot accelerate with GPU.

Table 4: The time-consuming of matrix product.

	Original	Vanilla	Our
Time consuming (s/batch)	1.6	840.5	8.3

4.5 Overall Evaluation

In this section, we practice an application in Keywords Spotting with LSTM [6] with the method of the overflow containing and the simulating. LSTM should be binarized, then we have a binarized LSTM named "BLSTM". Our BLSTM framework based on [1] has two LSTM layers and one fully connection layer. We give three experiments to evaluate the model with our methods. In the experiments, we use an open dataset [17] from Google AI, which provides 30 short words and 6 one-minute noise of different scenarios. The dataset has total 65,000 one-second long utterances in total.

As shown in Fig.5, the BLSTM loses much accuracy compared with the floating-point neural network. However, this loss can be compensated by increasing the hidden cells. The result also shows that the number of the hidden cells and the accuracy are significantly positive-correlated. In this experiment, we use ten keywords as the positive samples and the remaining keywords as the negative samples. The LSTM has two layers of {10, 20, ..., 110, 128} hidden cells, 100 time-steps, and 40 features of Mel-frequency cepstral coefficients.

For observing the influence of keyword number, we add an "unknown" category that holds the remaining keywords of except the positive samples. The order of the keywords is **A**: 'bird', 'dog', 'eight', 'four', 'happy'; **B**: 'seven', 'six', 'three', 'bed', 'cat'; **C**: 'down', 'five', 'house', 'nine'; **D**: 'sheila', 'tree', 'up', 'wow', 'zero'; **E**: 'marvin','go', 'stop', 'yes', 'no'; **F**: 'one', 'two', 'on', 'off', 'left', 'right'. As shown in Fig. 6, the BLSTM of 128 hidden cells for each layer can handle tasks of different difficulty level very well. Besides, when the number of the keywords equals {25, 30}, the trend of the accuracy changes. This phenomenon hints that different keywords have different levels of difficulty for the spotting.

This part investigates the tolerance ability of LSTM to environment noises. At first, we augment the dataset that extra includes the "unknown" and "silence" categories. Firstly,

Table 5: The effect of the augmented dataset and original dataset.

Dataset	Floating-point LSTM	BLSTM
Original	99.80%	89.20%
Augmented	96.34%	85.77%

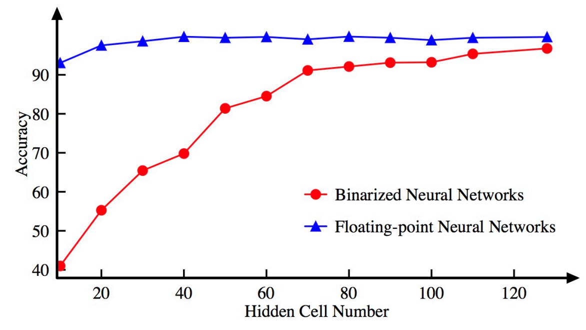

Figure 5: The affect of hidden cell number.

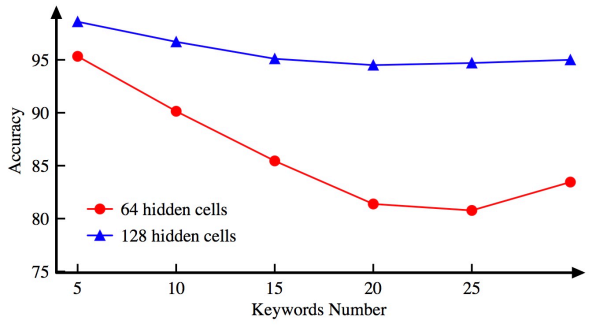

Figure 6: The effect of the keyword number.

6000 samples were sampled randomly from the six one-minute background noise to compose the "silence" category. Then, the 6000 noise samples cover randomly on all the original data with the different signal-noise ratios 10, 100, and also the noiseless. Finally, the ten keywords are still used as the positive samples, and the remaining 20 keywords compose the "unknown" category. The augmented dataset has 65000*3+6000 samples in total. As shown in Table 5, the BLSTM and floating-point LSTM have almost the same loss of accuracy, when the BLSTM and the floating-point LSTM respectively have 64 hidden cells. This experimental result is surprising because it indicates that the BLSTM is competent in scenarios with the noise. Because, by observing the Table 5, we can know that the floating-point LSTM loses about 3.5% accuracy and the BLSTM loses about 3.4% accuracy when processing the augmented dataset. Thus, it can be seen that BLSTM still has good anti-noise ability for this task.

5 SUMMARY AND CONCLUSIONS

The overflow is the biggest issue when the most methods are used directly in the specialized low-precision accelerators. To reduce the overflow, we focus on the overflow containment, the overflow/rounding simulating, and the BNNs training hyper-parameters including the design of BN layer and regularization term. Therefore, in this work, we propose a series of the effective methods to contain the overflow, simulate the overflow/rounding, and train accurate BNNs for low-precision neural accelerators. We hope our work can inspire the intelligent specialized accelerators to achieve better performance.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. The work was supported in part by National Natural Science Foundation of China (NSFC) under grant No. (61522406, 61834006, 61532017, 61521092, 61874124, 61504153), and Beijing Municipal Science & Technology Commission (Z171100000117019, Z181100008918006), Strategic Priority Research Program of the Chinese Academy of Sciences (XDPB12). The corresponding authors are Yinhe Han and Ying Wang.

REFERENCES

[1] Guoguo Chen, Carolina Parada, and Tara N Sainath. 2015. Query-by-example keyword spotting using long short-term memory networks. In *ICASSP*. IEEE, 5236–5240.
[2] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.
[3] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).
[4] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *AISTATS*. 315–323.
[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.
[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[7] Qinghao Hu, Peisong Wang, and Jian Cheng. 2018. From hashing to CNNs: Training BinaryWeight networks via hashing. *arXiv preprint arXiv:1802.02733* (2018).
[8] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).
[9] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. (2014).
[10] Shiqi Lian, Yinhe Han, Xiaoming Chen, Ying Wang, and Hang Xiao. 2018. Dadu-P: a scalable accelerator for robot motion planning in a dynamic environment. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.
[11] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*. Springer, 525–542.
[12] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[13] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei Li. 2016. C-brain: a deep learning accelerator that tames the diversity of CNNs through adaptive data-level parallelization. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE*. IEEE, 1–6.
[14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, et al. 2015. Going deeper with convolutions. In *CVPR*.
[15] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy?. In *AAAI*. 2625–2631.
[16] Ying Wang, Jie Xu, Yinhe Han, Huawei Li, and Xiaowei Li. 2016. DeepBurning: automatic generation of FPGA-based learning accelerators for the neural network family. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 110.
[17] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv preprint arXiv:1804.03209* (2018).
[18] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).
[19] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).