RECURSIVE BINARY NEURAL NETWORK LEARNING MODEL WITH 2-BIT/WEIGHT STORAGE REQUIREMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper presents a storage-efficient learning model titled Recursive Binary Neural Networks for embedded and mobile devices having a limited amount of on-chip data storage such as hundreds of kilo-Bytes. The main idea of the proposed model is to recursively recycle data storage of synaptic weights (parameters) during training. This enables a device with a given storage constraint to train and instantiate a neural network classifier with a larger number of weights on a chip, achieving better classification accuracy. Such efficient use of on-chip storage reduces off-chip storage accesses, improving energy-efficiency and speed of training. We verified the proposed training model with deep neural network classifiers and the permutation-invariant MNIST benchmark. Our model achieves data storage requirement of as low as 2 bits/weight while the conventional binary neural network learning models require data storage of 8 to 16 bits/weight. With same amount of data storage, our model can train a bigger network having more weights, achieving 1% better classification accuracy than the conventional binary neural network learning model. To achieve the similar classification error, the conventional binary neural network model requires 3-4$\times$ more data storage for weights than our proposed model.

1 INTRODUCTION

Deep Neural Networks (DNN) have demonstrated the state-of-the-art results in a wide range of cognitive workloads such as computer version Krizhevsky et al. (2012) and speech recognition (Hinton et al. (2012)), achieving better-than-human performance for the tasks often considered too complex for machines. The success of DNN has indeed motivated scientists and engineers to implement a DNN in mobile and embedded devices, dubbed as Internet of *Smart* Things (Kortuem et al. (2010)). The recent works in this area however, mostly implement inference function of DNN, without function of training DNN, while training DNN is done in cloud computers and post-training weights are downloaded to mobile and embedded devices (Lane et al. (2016)).

On-device learning, however, becomes increasingly important for the mobile and embedded devices for the following three reasons. First, intelligent device benefits to have the model that is custom-built for the device and its user and environment. This is because the model tends to be more accurate and effective if constructed with the consideration of those factors. Second, the training data from mobile and embedded sensing devices can contain security-sensitive information, e.g., personal health data from wearable medical devices. At the risk of being leaked, users typically do not want to upload such data onto cloud computers. Finally, in the era of Internet of Things (IoT), we anticipate a drastic increase in the number of deployed devices, which can proportionally increase the number of learning tasks to be done in the cloud. Coupled with the complexity of learning, even for powerful cloud computers, this can be a computationally challenging task.

On-device learning, however entails various challenges, in algorithms, data, and systems (Roschelle (2003); Vogel et al. (2009)). The most eminent challenge regarding computing systems is high energy consumption caused by dense computation and data access of DNN system which is prohibited to limited resources of embedded devices. The high overhead of data access is caused by fetching DNN weights from DRAM (or FLASH) external to a computing chip on an embedded device. Since the data storage size is limited for such computing chip, the parameters of a DNN have to be stored in external DRAM and FLASH during training. For example, ARM Cortex M3 processor, a

processor widely used in commercial wearable devices such as FitBit, has only 64 kilo-Byte (kB) on-chip data storage. This can only fit in very small size of DNN especially if each weight is 32-bit float point number. Compared to accessing on-chip SRAM, accessing off-chip DRAM incurs 3 to 4 orders of magnitudes more energy and delay overhead. Therefore, fetching weights every time for each data makes training prohibitive to be implemented on a mobile and embedded device (Han et al. (2015)).

Recently several techniques on pruning, distilling, and binarizing weights have been proposed to compress the parameters of a DNN. This makes it more feasible to fit weights in on-chip SRAM (Han et al. (2015); Courbariaux et al. (2015; 2016); Rastegari et al. (2016); Hinton et al. (2015)). These techniques can also reduce computation overhead. However, these works focused on weight size compression ***after training is finished***. The data storage requirement during training remains the same.

Similarly, several learning models, which belong to so-called binary neural networks, have been proposed (Courbariaux et al. (2015; 2016); Rastegari et al. (2016)). These model uses sign bits (or binary information) of weights in several parts of the learning model notably the part of multiplying and accumulating weights with inputs/activations. Although this greatly reduces computational complexity, each weight still needs to be represented in high precision number with multiple bits during the end-to-end training process since it has to be iteratively fine-tuned in the weight update part. Therefore, this so-called binary neural network models have not demonstrated to scale storage requirement below 32 bits/weight.

Our goal is, therefore, to efficiently use the limited amount of on-chip data storage during training. We also aim to scale computational complexity. Toward this goal, we propose a new learning model, *Recursive Binary Neural Network (RBNN)*. This model is based on the process of training of a neural network, weight binarization, recycling storage of non-sign-bit portion of weights to add more weights to enlarge the neural network for performance improvement. We recursively perform this process until either accuracy stop improving or we use up all the storage on a chip.

We verified the proposed RBNN model on a multi-layer perceptron (MLP)-like classifier and the MNIST benchmark. And also convolutional neural networks (CNN) and voice activity detection (VAD) tasks. We considered typical storage constraints of embedded sensing devices in the order of hundreds of kB. The experiment in MLP-like classifier confirms that the proposed model (i) demonstrates 1% classification accuracy improvement over the conventional BNN learning model specifically following Courbariaux et al. (2015) for the same storage constraints or (ii) scale on-chip data storage requirement by $4\times$ for the same classification test error ($\sim$2%), marking the storage requirement of 2 bits/weight. The conventional BNN model Courbariaux et al. (2015) but also Courbariaux et al. (2016); Rastegari et al. (2016) exhibits a significantly larger storage requirement of 8 to 32 bits/weight. The experiment of CNN classifying MNIST proves as high as $6\times$ saving of on-chip data storage requirement and 2.4% accuracy improvement. For VAD tasks, the proposed RBNN proves $9\times$ data storage saving.

The remainder of the paper is as follow. In Sec. 2 we will introduce the works related to this paper, including comparison to existing works on DNN compression, BNN, and low-precision DNN. In Sec. 3 we will describe the intuitive and details of our proposed model. Sec. 4 will present the experimental results and comparisons to the conventional BNN model. Finally, in Sec. 5, we will conclude the paper.

2 Related Work

2.1 Distillation and Compression of DNN parameters

Knowledge distillation Hinton et al. (2015) is a technique to compress knowledge of an ensemble of DNNs into one small DNN while maintaining the accuracy. Although this technique can scale the number of weights for deployment systems post-training, it cannot scale data storage requirement for training. Specifically, during training, each of weights is represented in high-precision number, which needs to be stored in multi-bit data storage.

Another technique is to compress the data size of weights by exploiting redundancies in them. In Han et al. (2015), the authors combine four sub-techniques, namely weight pruning, quantization,

sharing, and compression coding to reduce the data size of weights. Similar to the knowledge distillation, this technique can be applied to the weights that are already trained, and cannot scale data storage requirement of weights during training.

2.2 BINARY NEURAL NETWORK (BNN)

Recent works proposed to use binary information of weights Courbariaux et al. (2015), activations Courbariaux et al. (2016); Rastegari et al. (2016), and even inputs Rastegari et al. (2016) in some parts of learning and post-learning operations. The use of binary information of weights notably in multiply-and-accumulate (MAC) operation can drastically reduce computational complexity. However, those BNN techniques still cannot scale the storage requirement of weights during training. In these works, each weight is represented in 32 bits. This is because mainstream training models such as stochastic gradient decent requires fine-grained update of weights.

2.3 LOW-PRECISION FIX-POINT WEIGHT REPRESENTATION

Several studies have demonstrated moderately lowering the precision of weights (i.e., quantization) has a tolerable impact on training and post-training operations of DNN (Gupta et al. (2015); Courbariaux et al. (2014)). In Gupta et al. (2015), the authors trained a DNN having 16-bit fixed-point weights with the proposed stochastic rounding technique, and demonstrated little to no degradation in classification accuracy. In Courbariaux et al. (2014), the authors proposed the dynamic fixed-point representation (i.e., dynamically changing the position of decimal point over computation sequences) to further reduce the precision requirement down to 10 bits per synapse. Using fix-point representation help to reduce storage requirement and fixed-point arithmetic is more hardware friendly (Han et al. (2015)).

3 RECURSIVE BINARY NEURAL NETWORK (RBNN) MODEL

3.1 KEY IDEA

Table 1 shows which information of weights are used in each step of training in both conventional BNN Courbariaux et al. (2015; 2016); Rastegari et al. (2016) and our proposed RBNN. The conventional BNN works Courbariaux et al. (2015; 2016); Rastegari et al. (2016) use sign bits of weights during multiply-and-accumulate (MAC) operation in feed-forward and back propagation. However, the weight update has to be done in high precision. This mandates to store multi-bit (32 bit if using float point number) weights in data storage during learning, resulting in no savings in weight storage requirement.

However, it has been studied that in the trained neural networks we can use only the sign bits of weights to perform inference (Courbariaux et al. (2015; 2016); Rastegari et al. (2016)). This vast different requirements of weight precision between learning and post-learning inspires us to create our RBNN model. As shown in the third column in Table 1, we also use only the sign bits for MAC operations to reduce computational complexity. The main difference is that after training and binarization of weights (keeping only sign bit), we recycle the data storages that are used to store non-sign bits of weights to add more multi-bit trainable weights to the neural network. We then trained this new network having both the binarized non-trainable weights and the newly-added trainable weights. We perform this process recursively, which makes the neural networks larger and more accurate but using the same amount of data storage for weights.

Figure 1 depicts the process of our proposed RBNN learning model with an example of multi-layer fully-connected neural network. In the beginning the neural network has one input, two set of two hidden, and one output neurons and eight weights each of which has n bits. We first train this $1 \times 2 \times 2 \times 1$ network using the conventional back-propagation training algorithm. After that, we discard all bits except the sign bit in each weight (binarization), resulting in a $1 \times 2 \times 2 \times 1$ trained network having binary weights (*trained_BNN*). Then we continue the second iteration of training (the second subfigure of Figure 1). Specifically, we recycle the storage that is used to store the n-1 non-sign bits of weights in the $1 \times 2 \times 2 \times 1$ network. Using this data storage, we add eight additional weights (W_{21} to W_{28}) to the *trained_BNN*, expanding the network to a $1 \times 4 \times 4 \times 1$. In this *enlarged_BNN*, each of the newly-added weights is $n - 1$ bits. In other words, the *enlarged_BNN*

Table 1: Comparisons of weight information usage in BNNs and RBNN

Steps	BNN	Proposed RBNN
MAC in forward prop.	Sign bits of weights	Sign bits of weights
MAC in back prop.	Sign bits of weights	Sign bits of weights
Weight update	All bits of weights	All bits of weights
Recursive recycling	N/A	Keep sign bits and recycle storages of the other bits for more plastic weights

comprises of one *trained_BNN* that has eight weight (W^b_{11} to W^b_{18}) that are trained (binary, non-plastic, marked as solid lines in Figure 1) and one *incremental_BNN* with eight weights (W_{21} to W_{28}) that are under training (n-1 bits, plastic, marked as dash lines in 1). The *incremental_BNN* is trained together with the *trained_BNN* but only the weights of *incremental_BNN* are updated.

We repeat the same process of binarization and recycling. In every iteration, the *enlarged_BNN* integrates 8 more weights, and the bit-width of newly-added plastic weights in the *incremental_BNN* is reduced by one. At the k iterations, the *trained_BNN* has $8 \cdot (k-1)$ neurons and the plastic weights have $(n-k+1)$ bit-width. After the k-th training is finished, as shown in the rightmost in Figure 1 the neural network becomes a $1 \times 2k \times 2k \times 1$ with $8 \cdot k$ binary weights. This network has k times more weights than the first $1 \times 2 \times 2 \times 1$ network. However, the data storage of weights remains the same, scaling the storage requirement per weight to n/k $(= 4 \cdot n/4 \cdot k)$, which is k times smaller than that of the first network. Thus the proposed RBNN can either achieve better classification accuracy - enabled by the more number of weights - with the same amount of weight storage, or reduce weight storage requirement for the same classification accuracy level.

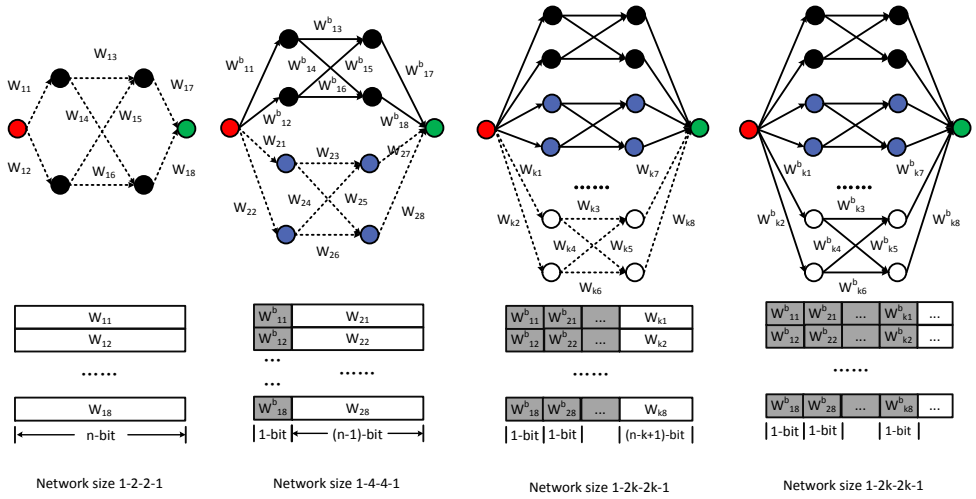

Figure 1: RBNN learning model with an example neural network. The recursive operation increases the number of weights in the neural network (top) while using the same amount of storage for weights (bottom).

3.2 Model details

Figure 2 depicts the details of the proposed RBNN model. The left part of Figure 2 depicts the flow chart of RBNN model, while right part explains the function of each stage. In the beginning of the training procedure, conventional BNN training algorithm *BNN_Training* is used to train a BNN. After training, we got a *trained_BNN* with binary synaptic weights. The synaptic bit-width

is reduced by 1. And then we use the rest synaptic bits as weights of *incremental_BNN*. After training the *incremental_BNN* with algorithm *Incremental_BNN_Training*, the performance of the *enlarged_BNN* is tested. If the performance doesn't stop improving and there are still available synaptic bits after weight binarization, the rest synaptic bits will be reused to further enlarge current *trained_BNN*. Notice that in each iteration of the proposed RBNN model, it requires to only update current plausible synaptic bits while keep the trained synaptic bits untouched. The implementation of this sub-word operation of synaptic weights are illustrated in A

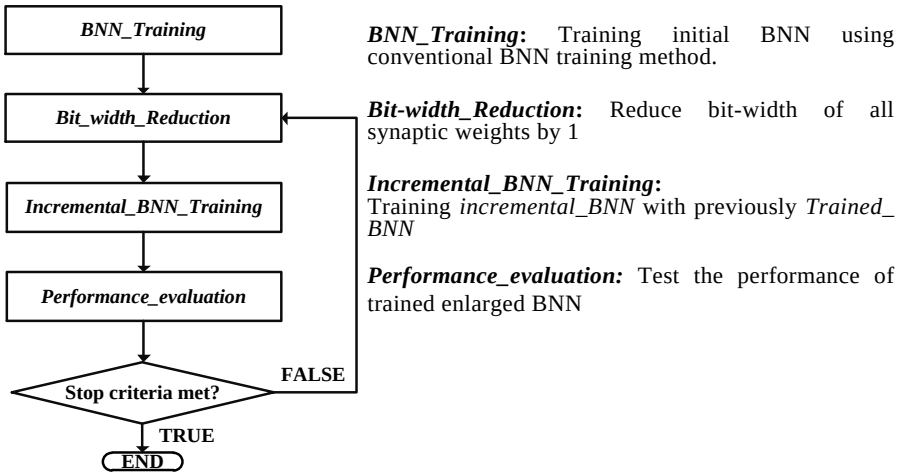

Figure 2: Detail of training method for RBNN model.

The method *Incremental_BNN_Training* is designed to train the *incremental_BNN* to improve performance of *enlarged_BNN*. To meet this goal, the conventional BNN training method is adjusted as shown in Algorithm 1. The main idea of this training method is: both *trained_BNN* and *incremental_BNN* are used to calculate output of the *enlarged_network* in feedforward stage. While during back-propagation and parameter-update, only plastic weights in *incremental_BNN* are updated. Similar to the conventional BNN training algorithm Zhou et al. (2002), binary weights are used in both feed forward and back propagation in *Incremental_BNN_Training*, to reduce computational overhead. Since weights in *trained_BNN* are binary, the multiplication related to weights can all be simplified as shift.

4 Experiment Setup

In this and next section, the detailed experiment setup and results for MNIST classification with MLP-like DNN systems using the proposed RBNN model are discussed. Furthermore, the proposed RBNN model is also applied to Convolutional Neural Network (CNN) classifying MNIST and MLP-like DNN doing Voice Activity Detection tasks. The setup and results are discussed in B and C, respectively.

4.1 Permutation-invariant MNIST benchmark

We used the permutation-invariant MNIST to test the performance of the proposed RBNN model. We use the original training set of 60,000 28-by-28 pixel gray-scale images and the original test set of 10,000 images. The training and testing data are normalized to be within the interval [-1, 1] and exhibits zero mean. Following the common practices, we use the last 10,000 images of the training set as a validation set for early stopping and model selection. As we use the permutation-invariant MINST, i.e., ignoring the 2-dimentional image structure of the image, we did not consider convolutional computation. We also did not consider data augmentation, pre-processing, and unsupervised pre-training during our experiment.

Algorithm 1 *Incremental_BNN_Training*. C is the cost function for mini-batch, η the learning rate and L the number of layers. The function *Binarize()* specifies how to binarize the weights. *Act_hid()* and *Act_out()* are activation function of hidden layers and output layer, respectively.

Require: a minibatch of inputs and targets (a_0, a^*), previous weights of incremental BNN $W(I)$, weights of *trained_BNN* $W(T)$

Ensure: updated weights of incremental BNN $W(I)^{(t+1)}$

1. Forward Propagation

1.1 Computing outputs of hidden layers in *trained_BNN* and *incremental_BNN*

for k = 1 to L-1 **do**

$a(T)_k = Act_hid(W(T)_k \cdot a(T)_{(k-1)})$

$W(I)^b_k \leftarrow Binarize(W(I)^b_k)$

$a(I)_k = Act_hid(W(I)^b_k \cdot a(I)_{(k-1)})$

end for

1.2 Computing outputs of enlarged BNN

$a_L = Act_out(W(T)_L \cdot a(T)_{(L-1)} + W(I)_L \cdot a(I)_{(L-1)})$

2. Back propagation

{Please note that only gradients of *incremental_BNN* are computed.}

Compute $g_{aL} = \frac{\partial C}{\partial a_L}$ knowinga_L and a^*

for k = L to 1 **do**

$g_{W(I)^b_k} \leftarrow (g_{a(I)_k} \circ a^{'}(I)_k) \cdot (W(I)^b_k) \cdot a(I)_{k-1}$

end for

3. Parameter Update

Please note that only weights of incremental_BNN are updated.

for k = L to 1 **do**

$W(I)^{T+1}_k \leftarrow W(I)^t_k + \eta \cdot g_{WI^b_k}$

end for

4.2 Neural network configuration and data format

We considered the storage constraints of mainly hundreds of kB based on the typical embedded system designs (Shiue & Chakrabarti (1999)). We considered a feed-forward fully-connected neural network with one or two hidden layers. We considered several different numbers of neuron neuron units in the hidden layer ranging from 200 to 800. The numbers of the input and output units are 784 and 10, respectively. We used the *tanh_opt()* for the activation function of the hidden layer and the *softmax()* or linear output for that of the output layer. We use the classical Stochastic Gradient Descent (SGD) algorithm for cross-entropy or hinge loss minimization without momentum. We use a small size of batch (1,000) and a single static learning rate which is optimized for each BNN. Any other advanced techniques such as dropout, Maxout, and ADAM, and etc are not used for both the proposed and the baseline learning models. We recorded the best training and test errors associated with the best validation error after up to 1,000 epochs. The results are averages of results from 20 independent experiments for each case.

We used the fixed-point arithmetic for all the computation and data load and access. The fix-point intermediate computations, such as gradient calculation have, also use fixed-point arithmetic with sufficient precision. The translation from wide fixed-point number to narrow fixed-point and binary number is performed with simple decimation without using advanced techniques such as stochastic rounding (Courbariaux et al. (2014)). We saturated values in the event of overflow in weight update. The dynamic range of fix-point representation is optimized to achieve better accuracy performance.

5 Results and Discussion

5.1 Accuracy improvement

Figure 3 depicts the classification errors of the proposed learning model across three recursive iterations. The initial bit-width of weights is 8 bits. In each series of data points in Figure 3, the leftmost point is the initial neural network, i.e., with 2 layers of 200 hidden units and 198,800 synaptic

weights ($= 784 \cdot 200 + 200 \cdot 200 + 200 \cdot 10$). At this point, the synaptic bit requirement, defined as the ratio of total storage bits to the number of weights, is 8-bit/weight. At this point the network is equivalent to one trained with the conventional BNN model specifically following (Courbariaux et al. (2015)). In the second leftmost data point in the series is the neural network after the first recursive iteration. The network size is enlarged by twice, resulting in the $784\times400\times400\times10$ network. This reduces synaptic bit requirement to 4 bits/weight. Compared to original BNN, the enlarged BNN respectively achieves $\sim$0.7% and $\sim$0.4% reduction in training and test error rate. Finally, after three recursive iterations, the neural network implements BNN with size $784\times800\times800\times10$ (555,800 synaptic weights). It requires only 198.8 kB data storage for weights, marking the weight storage requirement of 2b/weight and the test error of 2.17%. This accuracy is as good as fully-connected network with $4\times$ times of synaptic data storage in (Courbariaux et al. (2015)). In this experiment the structure of the generated network is as shown in Figure 1, i.e. the hidden neurons are not fully connected. The performance of this structure is as good as fully-connected version generated by the proposed RBNN model. The experiment will be discussed in D.

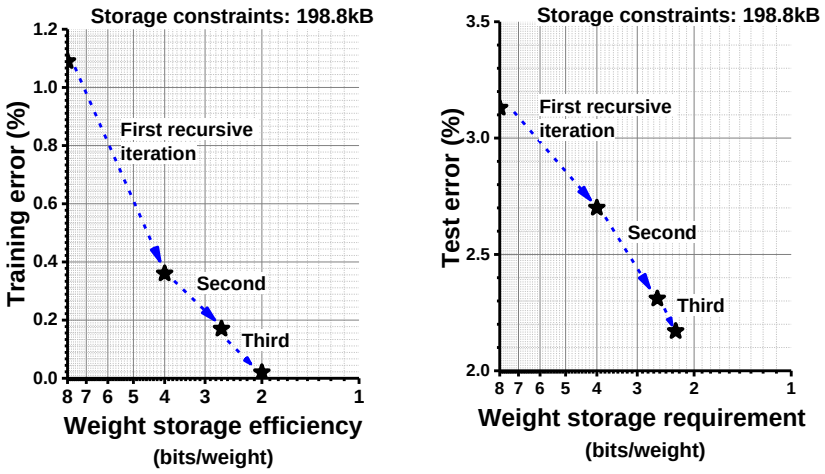

Figure 3: (left) Training error and (right) testing error across recursive iterations in the proposed RBNN model. The total weight storage used is 198.8 kB.

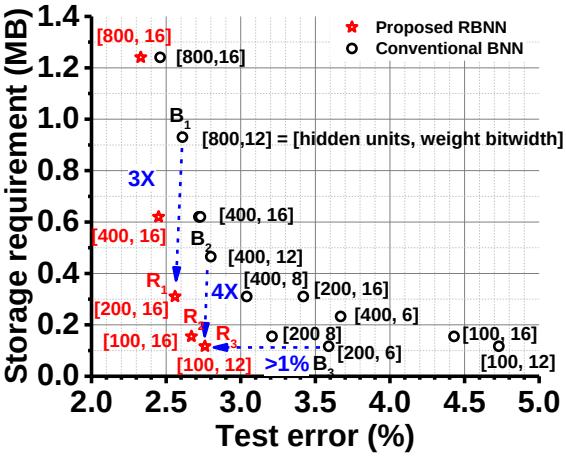

Figure 4: The storage requirement and test error trade-offs achieved by the proposed RBNN model and the conventional BNN model. The proposed model achieves $3\times$ data storage savings for the same test error and $> 1\%$ lower error for the same data storage.

Table 2: Detail comparisons of RBNNs and BNNs

	R_1	R_2	R_3	B_1	B_2	B_3
Initial hidden neurons	200	100	100	800	400	200
Final hidden neurons	800	700	400	800	400	200
Final synaptic weights	635,200	555,800	317,600	635,200	317,600	155,600
Initial weight bit-width	16	16	12	12	12	16
Storage requirement	**4**	2.28	**3**	**12**	**12**	16
Test error (%)	2.56	2.65	2.76	2.61	2.80	3.60
Arithm., training	2,223,200	2,779,000	1,111,600	1,270,400	635,200	317,600
Shift/Multiply/Add	635,200	555,800	317,600	635,200	317,600	158,800
	2,223,200	2,779,000	1,111,600	1,270,400	635,200	317,600
Arithm., inference	635,200	555,800	317,600	635,200	317,600	158,800
Shift,Add	635,200	555,800	317,600	635,200	317,600	158,800
Storage for weights	310kB	155kB	116kB	930kB	465kB	114kB
Total Train Energy (nJ)	**1,540.36**	1,377.59	**564.24**	**152,623.67**	**76,311.84**	50,883.49
Arithm.	**365.24**	402.95	**123.56**	**175.67**	**87.84**	67.49
Data Access	**1,175.12**	1,028.23	**440.67**	**152,44.80**	**76,224.00**	50,816.00

Table 3: Energy Table for 45nm CMOS Process

Operation (int)	12-bit ADD/ SHIFT	12-bit MULT	12-bit SRAM	12-bit DRAM	16-bit ADD/ SHIFT	16-bit MULT	16-bit SRAM	16-bit DRAM
Energy [pJ]	0.0375	0.126	1.387	240	0.05	0.225	1.85	320
Relative Cost	1	3.4	37	6400	1.3	6	49.3	8533

5.2 STORAGE SAVINGS

We trained DNNs of a range of configuration using our RBNN and the conventional BNN learning model (Courbariaux et al. (2015)). We used fix-point number for weights. For the conventional model, we considered BNN containing from 100 to 800 hidden neurons combined with 16-b to 6-b synaptic weights. For the proposed model, we considered from 100 to 800 initial hidden neurons and 12 to 16-bit initial weight precisions. Those DNNs require 116 kB to 1.2 MB data storage for weights. Figure 4 shows the results of this experiment: the proposed model can achieve 1% lower test error than the conventional model using the similar amount of data storage. To achieve the similar test error, the proposed RBNN model requires 4× less data storage than the conventional BNN model. It is noticed that the accuracy improvement is not as significant when hidden neurons are as many as 800. We observe that at this point, the training data has been modeled well (0 training error and very small cost function value), so we think at this point, regularization technique is needed to further improve performance, which is beyond the scope of this paper. Table 2 shows the detail comparisons of six neural networks, three from the proposed RBNN model (R_1, R_2, R_3) and three from the conventional BNN model (B_1, B_2, B_3) (Courbariaux et al. (2015)). The arithmetic complexity for learning and inferring is also listed. For conventional BNN, the amount of arithmetic of learning one data sample for one epoch is shown. While for RBNN, we add up arithmetic amount for all binarization-recycle iterations of training. R_1 and B_1, R_3 and B_2 achieve similar test error, but R_1 outperforms B1 in the storage requirement by 3× and R_3 outperforms B_2 by 4×, respectively. The downside of R_1 and R_3 is the increase in computations during training: it requires more shift and add operations, but it still needs the same number of multiplications which is much more complex than add and shift, as compared to B_1 and B_2. On the other hand, the computation complexity for inference operation are the same.

5.3 ENERGY CONSUMPTION SAVING

The last three rows of Table 2 compare the training energy with respect to both computation and data access for RBNN model and conventional BNN. The training energy is defined as the sum of energy cost of arithmetic and data storage access for synaptic weights. The energy consumption is calculated based on training arithmetic complexity and energy cost of basic arithmetic and memory operations in a 45nm CMOS process shown in Table 3. The original value of energy cost from Han et al. (2016)is scaled squarely with bit-width to calculate energy cost of multiplication and linearly to calculate other arithmetic and storage access operations.

The results in Table 2 show that to achieve same performance level, the proposed RBNN saves around 100$\times$ total energy for training (R_1 and B_1, R_3 and B_2), even though it requires slightly more computation. This is because RBNN reduces the data storage requirement for on-chip learning, so the data can be easily fit in on-chip SRAM of chips for low-cost application. On the other hand, for conventional BNN, the weights have to store and fetched from off-chip DRAM during training (Altera), which costs 2-order of magnitudes more energy than SRAM access and basic arithmetic according to Table 3.

6 CONCLUSION AND FUTURE WORK

This paper presents a learning model regarding local on-device learning with limited data on-chip storage. The proposed RBNN model efficiently uses limited on-chip data storage resources by recycling data storage that would have been wasted, to add and train more synaptic weights to a neural network classifier. We verified the proposed model with the neural network classifier and the permutation-invariant MNIST benchmark under the typical embedded device storage constraints. The results show that the proposed model achieves 2b/weight storage requirement while achieving 1% better classification error as compared to the conventional binary-weight learning model for the same storage constraint. Our proposed model also achieves 3-4$\times$ less data storage than the conventional model for the same classification error. We expect the future work that extends the application of the learning model to other neural network topologies and datasets, such as BNN with binary activation function Courbariaux et al. (2016), and also more complicated model like CNN. We also expect to apply the RBNN model to the ensembles of neural networks (Zhou et al. (2002)), the mixture of experts (Shazeer et al. (2017)), and the incremental learning (Xiao et al. (2014)).

REFERENCES

Altera. Altera cyclone iv overview. `https://www.altera.com/products/fpga/cyclone-series/cyclone-iv/overview.html`. Accessed December 4, 2017.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

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*, pp. 3123–3131, 2015.

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.

Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1737–1746, 2015.

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.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pp. 243–254. IEEE Press, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Gerd Kortuem, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2010.

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.

Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, pp. 1–12. IEEE, 2016.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

David Pearce and J Picone. Aurora working group: Dsr front end lvcsr evaluation au/384/02. *Inst. for Signal & Inform. Process., Mississippi State Univ., Tech. Rep*, 2002.

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.

Jeremy Roschelle. Keynote paper: Unlocking the learning value of wireless mobile devices. *Journal of computer assisted learning*, 19(3):260–272, 2003.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Wen-Tsong Shiue and Chaitali Chakrabarti. Memory exploration for low power, embedded systems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pp. 140–145. ACM, 1999.

Doug Vogel, David Kennedy, and Ron Chi-Wai Kwok. Does using mobile device applications lead to learning? *Journal of Interactive Learning Research*, 20(4):469, 2009.

Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 177–186. ACM, 2014.

Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.

A Implementation of Sub-word Operation of Synaptic Weights

In the proposed RBNN model, during training only plausible synaptic bits can be updated, while fixed (trained) synaptic bits need to maintain their value. Figure 5 illustrates the implementation of this sub-word operation of synaptic bits. An 8-bit synaptic weight is used in Figure 5 as an example, of which 3 bits are fixed (trained) synaptic bits and 5 bits are plastic bits that are under training in current iteration. At the beginning of training procedure, one complete weight *synp_in* is fetched from weight storage to computation circuits. And then a signal *synp_mask* which has 1 at the position of fixed weight bits and 0 at the position of plastic synaptic bits (11100000 in this example) and its reverse type (00011111) are *AND*ed to the weight to separate fixed weight *synp_fix* and plastic

weight *synp_plsb*, respectively. During the training procedure, *synp_plsb* is updated while *synp_fix* maintains the same value. Once training is finished, the updated *synp_plsb* and initial *synp_fix* are concatenated through *XOR* operation. Compared with conventional BNN, the proposed RBNN only requires one *AND* operation at the beginning and one *XOR* operation at the end of each training epoch, the cost of which is much smaller than cost of arithmetic and data access of training process.

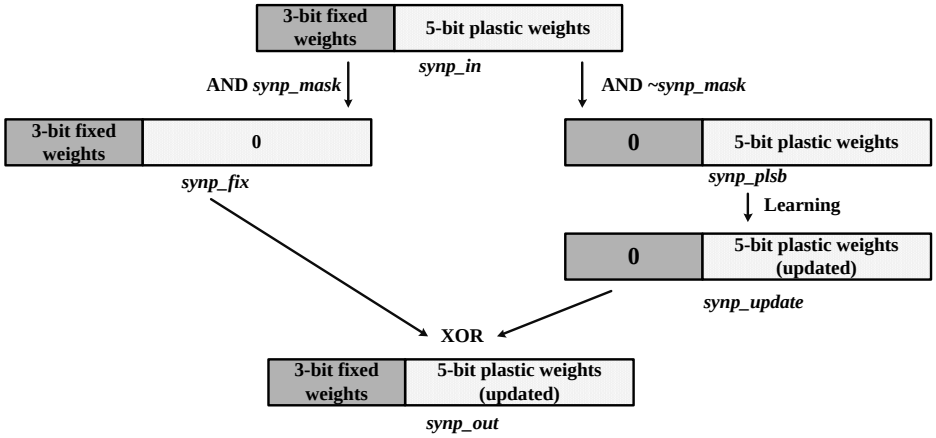

Figure 5: Detail of sub-word operation of synaptic weights.

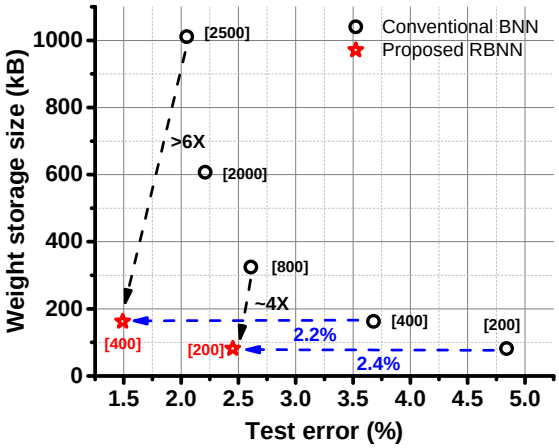

Figure 6: The weight storage size and test error for CNNs using the proposed RBNN and conventional BNN model. The proposed model achieves as high as 6X data storage saving and 2.4% accuracy improvement

B Application of RBNN to Convolutional Neural Network

B.1 Experiment Setup

We apply the proposed RBNN model to LeNet5 Convolutional Neural Network (LeCun et al. (1998)) with two convolution layers, having 6 and 12 of 5 by 5 feature maps, respectively. Both of them are followed by a 4x dowsampling average pooling layer. The fully-connected (FC) layers consist of one input, one hidden and one output layer. As in LeCun et al. (1998), binary weights are used for all convolutional and FC layers during forward and backward propagation and fix-point

Table 4: Accuracy and data storage size comparison of RBNN and conventional BNN on VAD tasks

	RBNN			BNN			
scenario	synaptic bit-width	hidden neurons	final accuracy(%)	synaptic bit-width	hidden neurons	accuracy(%)	data storage saving
bus	16	100	5.27	12	900	5.9	6.7X
cafe		100	8.8		1100	8.71	8.25X
park		100	7.94		1200	8.21	9X
river		100	8.15		1000	8.12	7.5X
traffic		100	8.05		900	8.07	6.75X

weights are used for weight updating to reduce computation overhead of training. The proposed RBNN model is applied to increase the size of hidden layer of the FC layers.

B.2 Results Analysis

We trained CNNs of a range of configuration for FC layers to classify MNIST dataset using the proposed RBNN model. And compare the accuracy with the CNN trained with conventional BNN method (Courbariaux et al. (2015)). For the proposed model, we considered from 200 to 800 *initial* hidden neurons and 16-bit *initial* weight precision. For the conventional model, we considered CNNs with FC layers containing from 200 to 2500 hidden neurons with 16-bit synaptic weights. The CNNs require 81kB to 1.01MB data storage for weights of all convolutional and FC layers. Figure 6 compares the accuracy performance and the according data storage size for CNNs with various configuration with both the proposed RBNN model and conventional BNN model. The results prove that the proposed RBNN model can achieve 2.4% lower test error than the conventional model using the same amount of data storage. To achieve the similar test error, the proposed RBNN model requires 7X less data storage than the conventional BNN model.

C Application of RBNN to Voice Activity Detection Tasks

To further validate the proposed RBNN model, we apply it to MLP-like DNNs to do Voice Activity Detection tasks. The structure of DNN used in this experiment is the same as the one in 4.2. The experiments were conducted on Aurora 4 (Pearce & Picone (2002)), which comprising 7133 utterances from 83 speakers. The basic feature used for the DNN to detect is 5 frames of 16-dimensional band-pass filterbank output. The signal-to-noise ratio of data in all scenarios used in the experiment is 10dB. Table 4 shows the topology and performance of DNNs using both RBNN and conventional BNN model for VAD tasks in various scenarios and proves as high as 9X data storage saving to achieve same level of detection accuracy.

D RBNN in fully-connected DNN systems

In this section we demonstrate one way of using the proposed RBNN to generate a fully-connected DNN and show the performance improvement compared to conventional BNN and tilted topology shown in Section 3.1. Figure 7 illustrates how the fully-connected DNN is built step by step with an example of DNN with initial size being 1-2-2-1 and n-bit. Firstly, new hidden neurons are added and the available weight storage is used for connection between input, newly added hidden, and output neurons. As shown in the second sub-figure, after k iterations, the topology becomes 1-2k-2k-1. Notice that the hidden neurons generated in different iterations are not connected with each other at this point. And then in the following iterations, the available weight storage is used to connect hidden layers generated in different iterations, adding m new weights each time, until all the neurons in the DNN system are connected with each other. The value of k and m is determined by size of available data storage, and the performance improvement of current iteration.

To evaluate the performance of fully-connected DNN generated with the proposed RBNN model, Figure 8 shows the classification performance on MNIST dataset of a 784-800-800-10 fully connect-

ed DNN generated from a 784-200-200-10 DNN with 8-bit initial synaptic bit-width. The test error rate at each iteration is shown, as well as the topology and plausible synaptic bit-width. According to Figure 8, the performance increases as more hidden neurons are added, while almost remains the same when more weights are added to connect existing hidden neurons. The possible explanation is to make sure the bit-width is wide enough, only 120k weights are added to connect hidden neurons in each iteration, which is much smaller than 198.8k in the first 3 iterations. The exploration of optimal way to generate fully-connected DNN or DNN with optimal topology is out of scope of this paper.

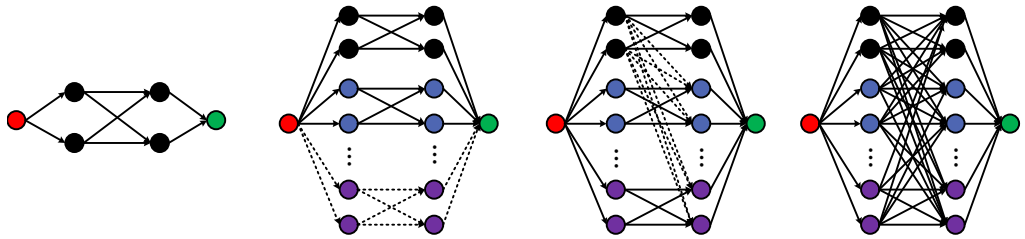

Figure 7: The way of building fully-connected DNN with the proposed RBNN model. Initially (first subplot), the DNN with size 1-2-2-1 has n-bit synaptic weights. After k times of iteration (second subplot), the network size is enlarged to 1-2k-2k-1 with hidden neurons generated in different iterations disconnected to each other. In the iteration afterwards (third subplot), the available synaptic bits are used for connecting hidden neurons, until the DNN becomes fully connected (forth subplot)
.

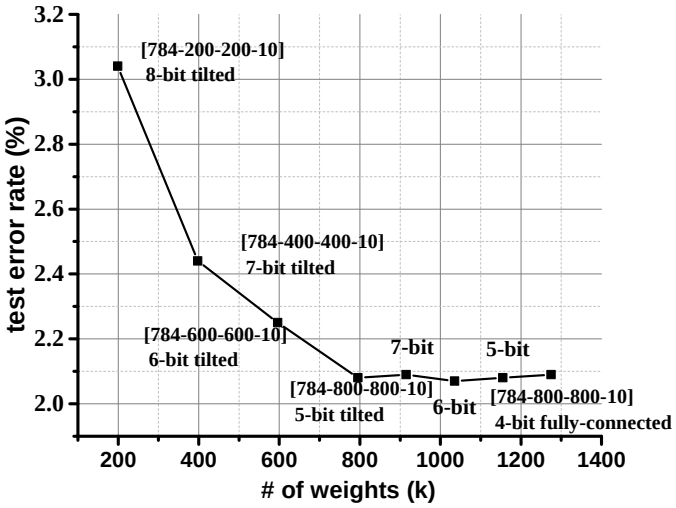

Figure 8: Performance of fully-connected DNN generated by the proposed RBNN model
.