どこでもAIに向けた機械学習システムのビットエネルギー効率向上

知能情報システム工学専攻

席　家禎

Improving Bit Energy Efficiency of Machine Learning Systems Towards an Era of AI-Everywhere

Intelligent Information System Engineering

Xi, Jiazhen

どこでも AI に向けた機械学習システムの
ビットエネルギー効率向上

知能情報システム工学専攻

セキ　カテイ

要旨

近年のディープラーニング型ニューラルネットワーク(NN)の機械学習システムはより複雑な画像の認識精度を畳み込み層をより深くして向上させている。結果、パラメータ数や乗加算計算回数、計算対象のデータの総移動距離/時間の増加に伴う急激な消費電力増加の課題が顕在化している。さらに今後、クラウドだけでなくセンサー付近にも AI を埋め込む AI-IoT（どこでも AI）のシステムの実現を考えた場合、今とは桁違いにエネルギー効率の高い機械学習システムの実現なくしてはその夢はかなえられない。本論文ではその目標の深刻な障害となる課題を解決する技術の提案と効果、残された課題について、以下の 2 つの提案技術から論ずる：

1)　Memory 回路にデータ記憶機能だけでなく乗加算計算（MAC）の機能も Dual Role として兼務させる In-Memory Computing 技術でデータ移動に起因するエネルギー削減手法を論じ、メモリアレーのコラム数削減手法の提案とその効果(16%削減)、課題を論ずる。

さらに、

2)　MAC 計算対象データの表現ビット数を従来の 32bit 浮動小数点でなく大胆に 1bit (Binary Neural Network: BNN) に削減することで、Bit 演算のエネルギー効率を桁で向上させることを目的とした低精度表現 NN 機械学習モデルの精度や安定性の問題を解決する手法の提案とその効果(CIFAR-10 データセットでは 15%精度向上と 54%のばらつき削減)、と課題を論ずる。

第 1 章では 1bit を含む低精度表現型 NN 機械学習モデル、BNN 機械学習の概念と目的を説明し、従来の BNN が抱える課題を明らかにし本論文の背景と動機を明らかにする。

第 2 章では従来の In-Memory 技術が微細化や低電圧動作に伴う MAC 演算回路特性の非線形性のために生ずるモデル誤差の解決のために導入していたアンサンブル学習の副作用である面積、消費電力課題を解決する手法を提案する。

第 3 章では従来の低精度表現型 NN 機械学習モデルが抱えるビット表現力の低下による精度劣化と不安定な学習状態を解決する手法として、注目する層にのみバギング学習を適用し、BNN での表現力低下を抑制し、従来の BNN 学習において課題であっ

たトレーニング曲線の大きなふらつきを抑制する手法を提案する。又、その効果を汎用的に評価するために広範囲に異なる NN 機械学習モデルのネットワーク構成に適用した場合、あるいは CIFAR-10 以外のデータセットに適用した場合の詳細な実験結果を取得し、その効果を汎用的に議論する。

第 4 章では、BNN 学習の不安定性の課題をさらに掘り下げ、いきなり 1 ビット化せずに段階的に徐々に 1 ビット化するなどの緩和プロセスを提案し、別の視点から提案されているバッチ正規化プロセスや Adam などのオプティマイザーに対する依存性などを明らかにしながら提案技術の効果を汎用的に議論する。

第 5 章では本論文の主要な結果を要約する。

西暦 2021 年 5 月 31 日

Improving Bit Energy Efficiency of Machine Learning Systems Towards an Era of AI-Everywhere

Doctoral Course of Intelligent Information System Engineering

Xi, Jiazhen

Abstract

The power consumption of neural networks has intolerably increased with the ever-increasing amount of number of MAC (Multiply Accumulate) operations and data movements for computing the machine learning models. Thus, the computing performance for the models can't be increased anymore under the power constraint systems until a much higher energy-efficient model could be developed. To address such a challenge, some trials with using in-memory computing and binary neural networks (BNN), have drawn much attention in the power-constraint fields like the internet of things (IoT). However, the 1-Bit training process of conventional methods also needs much training memory/time at a cost of the unstable training process and the loss of accuracy. This work proposed (1) A column reduction technique for the in-memory machine learning classifier. The proposed method can achieve similar accuracy as original full precision with MNIST dataset and with much lower computing memory (with 16% columns reduced), compared to conventional work. (2) Layer-wise ensemble technique for BNN to improve the performance of low precision networks via employing the ensemble learning technique which reduces the error and its standard deviation by 15% and 54% on the CIFAR-10 dataset, respectively, compared to the BNN serving as a baseline. (3) Training with the relaxation of both weights and activations for binary neural networks to alleviate the variance by the conventional BNN training process (reduced by 1.71%) with the experiments of various cases including different optimizers and with or without batch normalization.

The outline of this thesis is as follows. Chapter 1 presents the concept of low precision machine learning and the purpose of this work. Chapter 2 introduces the conventional in-memory boosting classifier and then proposes a column reduction technique to improve the bit energy efficiency of the in-memory boosting classifier followed by the comparison to the conventional technique. Chapter 3 presents the low precision neural network and its instability issue of conventional works and then proposes a layer-wise ensemble method for the binary neural network (BNN), followed by the experiments to compare the conventional methods and the proposed one the metrics of the stability and accuracy of the network. Chapter 4 presents the conventional training process of BNN with relaxation technique and proposes a new training procedure that employs relaxations to both low precision weights and activations, followed by the comparisons between the proposed method and conventional training process in various cases of the different optimizers and with trained or fixed batch normalization. Chapter 5 summarizes this thesis.

May 31, 2021

Index

Chapter 1 Introduction

1.1 Backgrounds

With the various sensors being developed and applied into reality, the target signals in the embedding sensing systems are being more and more complex, such that traditional fixed processing and rule-based algorithms are becoming more difficult to design for each target signal separately. To address such a problem, the data-driven models, enabling training models with various kinds of data, are playing an important role in the applications with signals too complex to analyze, especially at the computers of centralized cloud servers. With the recent developments of artificial intelligence (AI) algorithms as the core of data-driven models, the accuracy performance of the system has made great progress in almost everywhere of modern life applications like face recognition, speech recognition, self-driving cars, and various Internet of Things (IoT) systems (see Fig. 1.1).

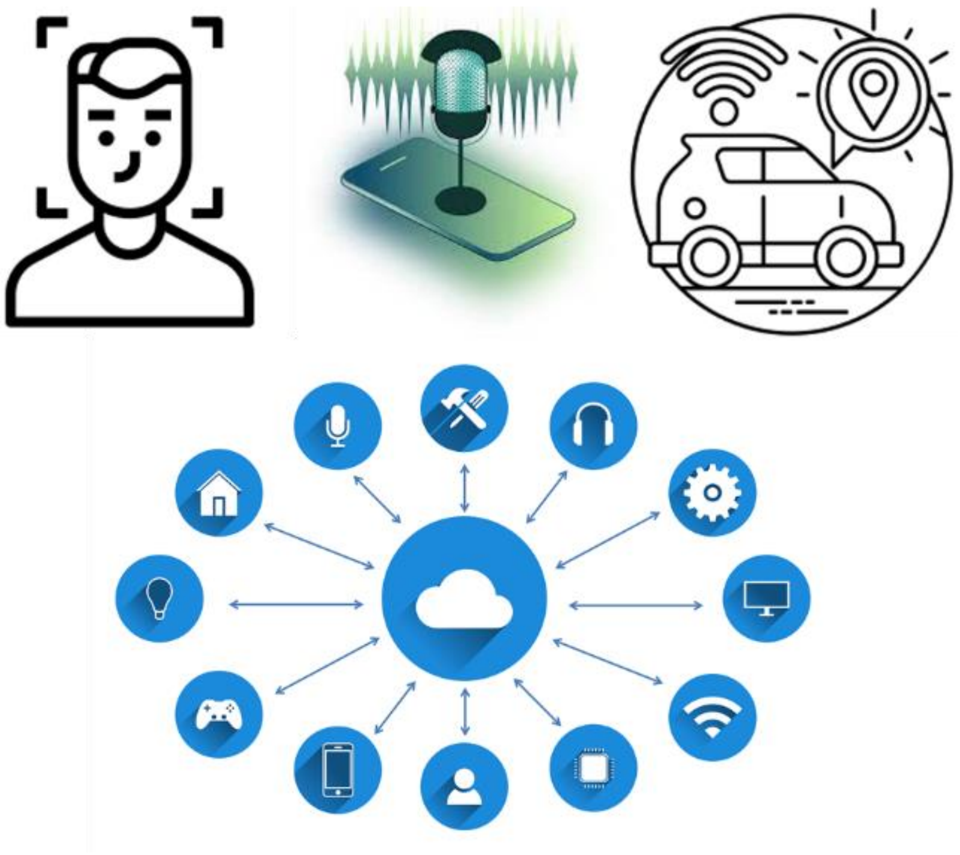


Fig. 1.1 The applications powered by AI algorithms

However, various sensors widen the features of systems, which conflicts with another important need for the low energy cost of such applications. What's more, with more complex AI-based algorithms being developed, the problem of energy cost and system power is becoming much serious. This somehow restricts the employment of data-driven models and algorithms in such embedding systems whose energy/resource is limited. The complex data generated by various sensors of the devices combined with the extreme energy limitations in the device side yields the deployment of AI not only at the cloud side but also the near-device (edge) side, which is exactly the concept of 'AI-everywhere'.

Faced with the conflict of energy cost and system performance, various embedding hardware architectures are being developed and improved. One of the most focused trends is the concept of 'Edge Computing', instead of one centralized cloud (see following Fig. 1.2 which shows the positions of 'edge' and 'cloud' in the era of AI-everywhere).

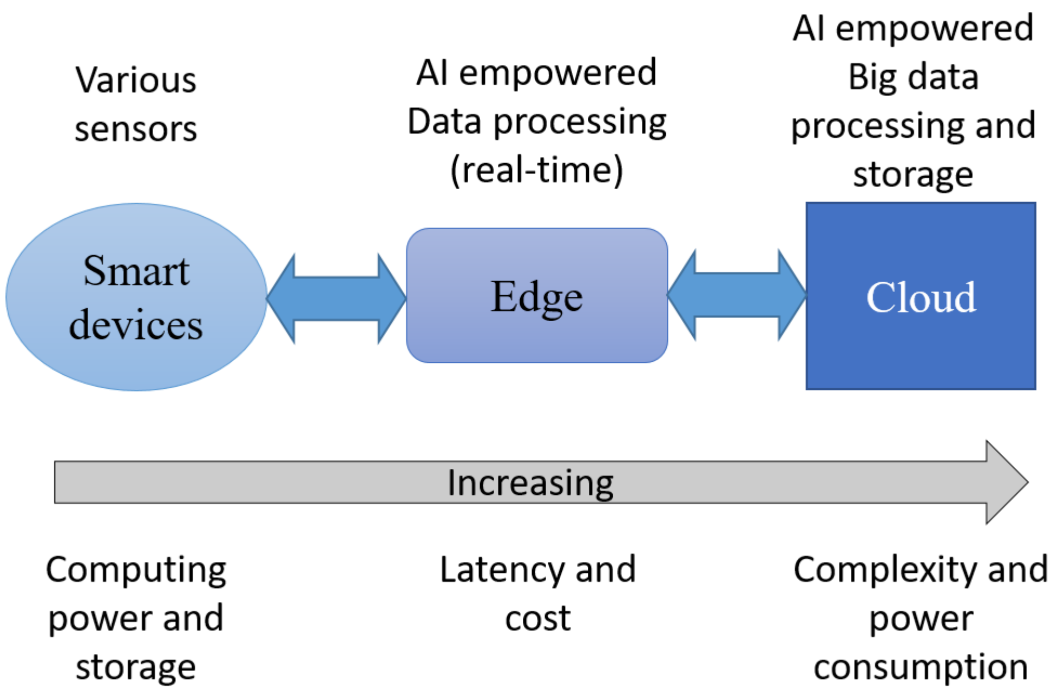

Various sensors
AI empowered Data processing (real-time)
AI empowered Big data processing and storage
Smart devices
Edge
Cloud
Increasing
Computing power and storage
Latency and cost
Complexity and power consumption

Fig. 1.2 The positions of Edge computing and Cloud computing in the era of AI-everywhere

The conventional systems with separate data storage and calculation module are still with energy cost at least, which is the cost of data accesses and communications. The following Table 1 shows the energy of operations in a 45nm CMOS at 0.9V. From the table, it is shown that the energy of the same kind of operation (add/multiply/read) among the numbers in different bit-width varies in a great range. Thus, the key to reducing the energy cost is to use the 'cheap' operations (integer add/multiply) instead of 'expensive' operations (float add/multiply), which is exactly to improve the bit-energy efficiency.

To improve the bit-energy efficiency, an ‘in-memory computing’ system architecture, which integrates the computing model and memory storage together into memory cells, was highlighted recently (M. Kang, 2014)[1]. By such a computing/memory combined structure, with the number of data accesses reduced largely, the whole energy cost is also reduced. In addition, with parallel processing enabled, such in-memory systems can achieve higher throughput compared to conventional structures, leading to a better processing speed.

Table 1 The energy of operations in a 45nm CMOS at 0.9V

Operation	Energy(pJ)
Integer ADD (8b)	0.03
Integer ADD (16b)	0.05
Integer ADD (32b)	0.1
Float ADD (16b)	0.4
Float ADD (32b)	0.9
Integer MULT (8b)	0.2
Integer MULT (32b)	3.1
Float MULT (16b)	1.1
Float MULT (32b)	3.7
8KB SRAM READ (32b)	5
32KB DRAM READ (32b)	640

With the development of deep learning algorithms, there are also increasing demands of reducing the energy of such kinds of models with millions or even billions of parameters. To address the above issue, the network quantization techniques were proposed to meet the requirements for saving energy and memory cost that is critical in mobile devices and embedded systems. The quantization technique allows relaxing the precision requirements of the network parameters from the float32 to lower integer precision like 8-bit, 4-bit, 2-bit, and even binary (1-bit) while suppressing the accuracy degradation to an allowable level. It can contribute to reducing the required computing resources and the memory footprint to complete the training within a practical time.

To summary, the variety of sensors and complex signals call for the employing of data-driven models mostly supported by AI algorithms. the scaling complexity of models and algorithms achieves better performance at the cloud side but also raises the problems of relatively high necessary system power and energy costs for the edge side. Such a problem yields the deployment of AI algorithms in both the cloud and edge side, which is 'AI-everywhere'. New structures like in-memory computing systems employed combined memory and computing models; new techniques like network quantization employed quantized parameters in low numerical bit-width. Such techniques largely reduce the energy cost but also make the computing accuracy degraded and model training more difficult[2]-[4]. The future target of the research to realize 'AI-everywhere' is to improve the bit-energy efficiency which is to hold and improve the performance of computing model and reduce the energy cost at the same time.

1.2 In-memory computing technique

In-memory computing was discussed initially early in the 1990s. Restricted by the development of memory and computing hardware at that time, however, the in-memory computing technique did not make much progress, but numerous novel system architectures had inspired the later research as well as developments in related areas. Nowadays, within the big data processing area, the concept of in-memory is that by saving data into RAM for professional servers rather than hard disk whose read/write speed is relatively limited and altering the whole server-side software architecture and the computing model to support parallel processes, leading to the breakthrough of the traditional hard disk I/O performance and data-processing-online. Thus, more efficient usage of data is achieved[5].

Distinguished from the concept of in-memory computing in the database and big data processing area, employing less or even no explicit data accesses so as to reduce energy cost and increase the system throughput by in-memory computing technique in the area like embedding systems and sensing signal processing is being widely applied. To make it clear, the in-memory computing technique is specially referred to as the technique used in the embedding system and signal processing areas in this paper's further discussion.

In such an area, because of limited resources and strict demand on the energy cost, except for the traditional target of high accuracy, the design for data processing and inference system is also directing to the exploration of low energy/power cost system design and architectures[6].

The following Fig. shows the concepts of the in-memory computation and conventional compute-out-of-memory (out-memory) process. The goal of the process in Fig. is to recognize images in 128 × 128 pixels. The conventional out-memory process first serially accesses data from the memory and then performs computation for recognition. During this process, the frequent data access makes it cost lots of energy, up to 100 nJ, to recognize one image. However, the computation of in-memory processes is performed within the memory (SRAM cells), which can avoid frequent access and only costs 1 nJ per image inference.

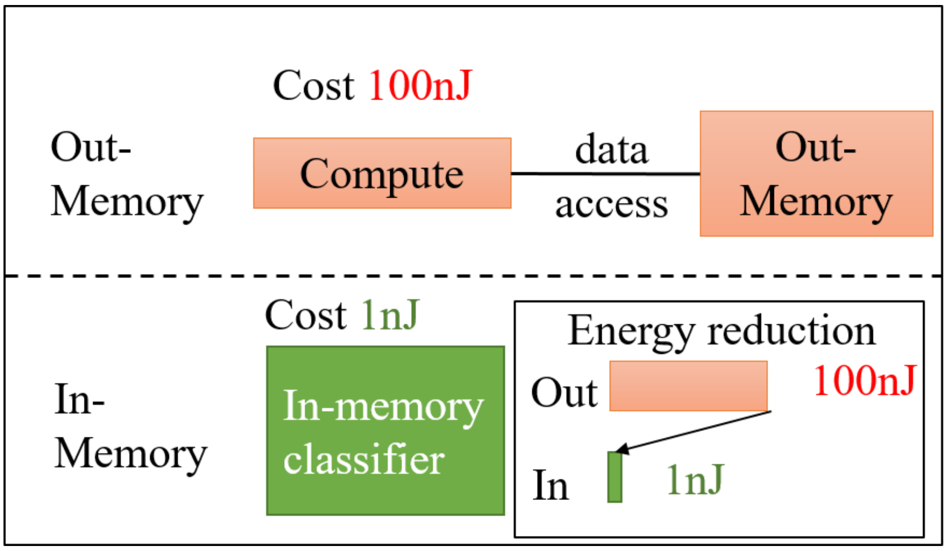


Fig. 1.3 Comparison between the out-memory process and in-memory process

In 1996, Berkeley IRAM project[7] carried on a series of initial experiments for the design of general computing system integrating processing unit and DRAM memory on single chip and proposed the concept of Intelligent RAM (IRAM), attempting to improve the metrics including delay, throughput, power, energy cost and necessary space of the systems by such kind of system architecture which can be employed to embedding, portable, desktop or distributed computer systems. At the same period of time, Computational RAM(CRAM)[8] and Smart memories[9] were also proposed in succession.

In 2002, H. J. Mattausch et al.[10] proposed the structure of associative memory and achieved searching 1-dimension minimum Hamming distance through parallel bit computation. In succession, Y. Oike et al.[11] proposed the structure of associative co-processor employing hierarchical searching architecture. In 2003, R. Genov et al.[12] proposed the concept of Kerneltron, combining the model of support vector machine (SVM) and DRAM arrays, achieving a large throughput and high efficiency with the relatively strong generalizing ability of SVM. In the same year, based on the demand of data-level parallelism (DIP) processing (in contrast of instruction-level parallelism), exited in the area of multimedia and telecommunication, C. E. Kozyrakis et al.[13] proposed a new architecture called ‘Vector RAM’ (VRAM) with low energy cost and simply designed structure, applicable for the DIP processing in the multimedia and telecommunication area.

The earlier research about in/near-memory-computing included variety of novel designs of system architecture and made many signs of progress in the respects like parallel processing and reducing system energy cost. In the recent years, with the mass development of integrated circuits (IC) especially very large-scale IC (VLSI), the effects of traditional circuit redundant designs ensuring hardware resilience and robustness to hardware variability as well as circuits non-linear noises are much harder to achieve. Thus, people pay more attention to the fact that the hardware variability make the performance of the hardware degraded[14], especially the in-memory computing systems employing a lot of analogy commutating characteristics. And to solve such problems, many kinds of solutions are proposed.

In 2010, V. K. Chippa et al.[15] proposed the 'scalable effort hardware design' converting the algorithm resilience to efficient hardware realization, achieving a lower energy cost and the robustness to the hardware nonlinear noises to some extents. The main idea of the proposed design method is that identify the mechanisms that can be approximated and generalized to the correct results through calculation in the abstract hardware designs, then extract them to expose them as the adjustable knobs in the realization and optimize them through algorithms, reaching an approximately available system realization, which is also always nearly correct. During the optimization, the cross-level optimizing is critical. Based on this concept of design, they realized a low power embedding SVM classifier, achieving 1.2~2.2× lower energy cost with no accuracy lost and 2.2~4.1× lower energy cost with least accuracy lost from experiment results. The comparison experiments of with/without cross-level optimization showed the ability of balancing energy cost and holding accuracy performance with cross-level optimization.

Traditional hardware implementation always includes redundant design in case of hardware error. However, with the scaling of memory technology, the space in the memory chip for such redundant design is becoming less and harder, which sometimes becomes an instinct limit to the performance of hardware systems. To solve such problem, a methodology of building inference models adapted with errors and nonlinear noises is proposed and becomes an alternative choice to achieve the hardware resilience of system[16].

In 2012, N. Verma et al.[17] proposed the concept of 'Data-Driven Hardware Resilience'(DDHR), which explores the method of embedding the inference ability of data-driven models in the hardware facilities to achieve the system-level resilience of hardware platform. With the techniques of pattern recognition, data mining and knowledge discovery widely being employed, there are more and more demands on advanced inference ability for modern embedding electronic systems. While latest embedding systems analyze business data through various kinds of data science methods, these methods should also be used to model the data including errors as well as the nonlinear noises coming from the hardware system itself. They conveyed several experiments mainly concerned about biomedical signal processing systems, showing that the system performance degraded by artificial insert stuck error and nonlinear noises can restore to a normal level through DDHR method.

Biomedical signal processing and analysis is always a challenging problem in the signal processing area. One of the difficulties is the necessity of extracting the target from the various background signals. For example, the EEG signal of patient suffering epilepsy could be very similar and difficult to distinguish compared to the signal of normal sleep from the same patient. Therefore, some high ordered signal inference models are needed. The other difficult point is that even from the same patient, the pathological signals represented during different period of time could also be different, leading to a demand of customized detecting system but quite expensive with traditional biomedical detecting facilities.

In 2013, K. H. Lee et al.[18] proposed a processor integrated CPU and configurable accelerator for many machine-learning functions to analyze different biomedical sensing signals. The CPU mainly realized feature extraction of different kinds of biomedical signals. The accelerator embedded with implemented SVM classifiers of many kernels including linear, polynomial, and radical basis functional kernels enabling the customized adaption of different kinds of pathological signals from different patients, reducing the efforts of human experts in the initial diagnosis phases. With two EEG/ECG based biomedical sensing applications, the writers employed the data-driven modeling framework to analyze the sensing data, restraining the errors and hardware variabilities and achieved a relatively high accuracy. The prototype of the experiments employed 130nm CMOS circuit and the implemented system achieved the necessary energy cost of 273μJ and 124μJ per decision by different applications, reducing the cost by 62.4× and 144.7× compared to the traditional single CPU implemented system. The other arrhythmia detection application[19] showed the customized analysis cost could be reduced by 20× through the proposed method.

In 2014, M. Kang et al. proposed the concept of ‘Compute memory’, enabling parallel multi-row accesses and analogy signal processing with the computing model embedded into SRAM arrays. The application in the pattern recognition of compute memory relaxed the constraints of demands of system on the respective of computational accuracy and linearity. With a demonstrated pattern detection system with 256x256 target images using 'compute memory', the writers found that with several certain errors included in the analogy signal processing circuits, even employing analogy calculations where the accuracy is limited, the performance of whole system was not being affected extremely. The results of simulations showed that the demonstrated system could reduce about 63% energy cost and probability of detection did not decrease when the PSNR>12dB.

N. Verma et al.[20] proposed ‘Hardware-Driven-Kernel Learning’ (HDKL) and discussed and compared it with ‘Data-Driven Hardware Resilience’ (DDHR) in concerned of implementing a machine learning inference model within computational resources limited embedding hardware systems. The writers thought DDHR used a top-down method, fitting the target data and also fitting the variety of nonlinear noises and errors. In contrast, HDKL employed a bottom-up method with the target of training a model enabling normalization of new kernel functions driven by the strategy of hardware implementation. With regard to the application of machine learning inference models in the embedding systems, the writers illustrated several opinions. First, the instinct resilience of data-driven models to the calculation errors makes the error tolerances possible. Second, with top-down fitting the statistical inference model with error data, enough ability of error tolerance can be achieved. Finally, bottom-up artificial designed inference kernel functions based on the hardware implementation can explicitly reduce the complexity of the implemented hardware system.

In 2015, Z. Wang et al.[21] proposed an ‘Error Adaptive Classifier Boosting’ (EACB) algorithm. Within the FPGA emulation, a ratio of >2% of all the circuit nodes inserted by the random errors and small ratio <1% of circuits nodes being error protected, the system could restore the normal performance level compared to non-error inserted state, with the model trained through the proposed EACB algorithm. Therefore, the writers thought that with the hardware errors aware and through the data-driven training with them, the inference model could restrain the nonlinear noises and errors from hardware circuits and its affects to the calculation accuracy. In addition, proposed EACB algorithm applied the small batch training methodology to traditional AdaBoost algorithm framework in order to fit in the resources limited embedding applications’ demands. Compared to traditional Boosting algorithms, the proposed algorithm achieved 65× memory demand as well as 10× energy cost. In the other paper of the writers[22], the availably and performance of EACB is validated through FPGA based system experiments.

M. Kang et al.[23] proposed a memory-intense, high-throughput and low-energy-cost VLSI architecture applicable for the convolutional neural networks with the compute model embedded into the memory array cells. The system level experiments of MNIST[24] classification application showed the errors and nonlinear noises could be complemented well by the strong fitting and error-tolerance ability of convolutional neural networks, reaching a final classification error rate of 0.87%. The writers also gave the behavioral model of the nonlinear noises as well as the estimated energy by the employed 45nm SOI CMOS circuits. The demonstrated system achieved the reduction of 24.5× energy delay product, 5× energy cost and a boost of 5× throughput with a relatively high classification accuracy, compared to the traditional implemented systems.

Generally, the processing and analysis for the sensing signals are after A/D converting in the traditional embedding sensing systems. In many of such applications, the target of analysis for the sensing signals is to make inferences from the sensing signals but people have no concerns about the original high dimensional signals data. However, the original high-dimensional signals sometimes could be too complex to model, or the computational/storage costs are too high to implement. J. Zhang et al.[25] proposed a ‘matrix multiply ADC’ (MMADC) with support for matrix multiplying. The writers proposed an algorithm combining the feature extraction and classification to a single model matrix enabling multiplication of input signals and programmable model matrix. Then the classification calculation is performed to achieve the final classification result. In the experiments of two applications including ECG signal-based arrhythmia detection and image-based gender classification, the results showed the classification performance of a MMADC approximates an ideal RBF kernel based SVM classifier, with 9.7×, 23× energy cost reduced in two demonstrated application experiments, respectively.

In 2016, W. Rieutort-Louis et al.[26] proposed a large-area image sensing and detection system, integrating sensors and thin-film transistor (TFT) circuit enabling detecting and classifying the images from the sensors. Modern large-area electronic (LAE) techniques implement over million numbers of sensors scaling and sensing images, while tremendous numbers of signals from sensors brings a big challenge to the post processing as well as the scalability of the systems. To solve such problems, the proposed system achieved redacting 3.5~9× of sensing signals number from original 36-way sensing signals through amorphous silicon circuit detecting the edges of images. The system employed EACB based models embedded into TFT circuits to classify the images with reduced features, the outputs also combined to reach stronger discriminate ability. Through EACB training algorithm, the system got over the hardware implementation defects and nonlinearities from the TFT circuits, reaching a better performance of classification compared to traditional AdaBoost implemented system. Within the experiment of 5-label classification of image edges, the demonstrated system reached a performance of true positive (TP)>85% and true negative (TN)>95%, approximately to an extent of ideal SVM classifier.

As the core of the data-driven models, the machine learning algorithms play an important role in the analysis and process of complex embedding sensing signals. There is a strict demand on 'always-on' detection, while the original complex high dimensional however information redundant signals' storage is not really needed in such applications. However, one of the main challenges of applying machine learning algorithms to such applications is that modern machine learning algorithms are sometimes really complex with large numbers of parameters or model weights. Thus, tremendous computational and storage/cache resources are needed, making their availability relatively limited in fact, where I/O accesses cost between the data storage and computing model contribute a lot.

To restrain I/O costs, S. Hanson et al.[27] proposed a mixed system structure including an 'always-on' based low-energy low-accuracy detector and a passive activated high accuracy detection node, reducing the total energy cost to an extent. However, all those conventional separate computing/storing structure systems still exist a lower limit of energy cost, the I/O accesses cost from computing model and data storage (mixed structure system restrains the cost but still has the separate structures). To resolve the problem, that is, to change the conventional computing/storing-separate structure fundamentally, some combined or embedded structures also proposed in succession[28][29].

From the literature we can find that in-memory computing has huge potential in areas like VLSI, IoT, low power/energy systems, and embedded sensing signal processing systems, while the data-driven models, as well as machine learning algorithms, play an important role in respective of improving signal analysis accuracy, restraining nonlinearity's effects on system performance, and enhancing system robustness and flexibility[30]. Therefore, improving system architecture and optimizing algorithms of the model to achieve better performance of the system and further reduced energy cost is a critical trend in recent research on in-memory computing.

1.3 Low precision deep learning with quantized parameters

In recent years, deep neural networks (DNNs) have been proved to be the state-of-the-art solutions in various complicated real-world applications, such as object recognition and detection, speech recognition and voice synthesis, and natural language processing. However, as the recent designs of DNNs have shown strong abilities even at the same level as human beings, a huge number of network parameters and more computational complexity are needed. The bandwidth also turns to be one of the limitations for the frequent data communications between computing units and memory. Therefore, the deployment of large DNNs on embedding systems and mobile applications is still quite difficult, which has extremely serious constraints on the resources[31]. The redundancy of the network designs made it possible to compress the DNNs with little or no loss of accuracy.

There are mainly three directions to compress or accelerate the network from the view of design redundancy[32]. (See following Table 2 for details).

(1) The structure of the networks.

(2) The optimization methods of the networks.

(3) The hardware accelerators for the networks.

Table 2 Network compression and acceleration approaches

Network Compression and Acceleration	Structure		Novel components, Architecture search, Knowledge distillation
	Optimization		Convolution optimization, Pruning, Quantization
	Hardware accelerators	Platform	CPU/GPU, ASIC, FPGA
		Optimization	Calculation table, Computation reuse, Memory optimization

There are mainly three parts in the approaches of network structures.

The novel components including separable convolution and residual blocks can reduce the number of connections through designing some efficient blocks. The network architecture search (NAS)[33] can automatically search for efficient network architectures from a large hyper-parameter space which is pre-defined. The knowledge distillation (KD)[34] can generate a compressed network which can reach the accuracy in the same level as a larger network, through training the student network to imitate the teacher network. Such student network should be simpler and have less computational consumption than the teacher network.

The network optimization includes the optimization techniques for convolution computations, network pruning, and network quantization. Thanks to taking 3D tensors as the input directly without flatting, there are fewer coefficients that the convolutional layers require compared to the fully-connected layers. Several techniques were proposed to further improve the computational efficiency of convolutional operations including Winograd convolution[35], fast Fourier transform (FFT) based convolution, and image to column approach[36]. Network pruning[37] focuses on pruning the network parameters that have no or less impact on the accuracy. Network quantization focuses on the conversion of the parameters' datatypes with lower numerical precision types like replacing 32-bit floating point with 8-bit integers.

The hardware accelerators[38] are mainly designed for the network acceleration with specialized platforms including central processing unit (CPU), graphic processing unit (GPU), and field programmable gate array (FPGA). Optimized with specified artificial intelligence instructions usually within single instruction multiple data (SIMD) units[39], CPUs are mainly used for inferences where the system do not have the specialized accelerators for inferencing. GPUs produced by different manufactures including NVidia[40], AMD[41], and ARM[42] can be used for both training and inference, which have specially optimized units for network acceleration. The ASICs are also designed for the training and inference acceleration, including Habana from Intel, Tensor processing unit (TPU)[43] from Google, Hanguang[44] from Alibaba, and Kunlun[45] from Baidu. FPGAs are also wide used for network acceleration and because of the gate level operations, FPGAs are usually used in low-bit-width networks and binary neural networks[46].

There are also some specific hardware optimizations for neural networks which are usually incorporated into some specialized ASICs. The lookup tables can be used for accelerations of activation functions. The partial products can be stored in special registers and then easy to be reused. The memory can be designed to reduce the computation cycles of a network through the memory access ordering with specialized hardware[47].

The network quantization is one of the most promising techniques to meet the severe constraints on memory footprint and computational power for the deployment of DNNs to the embedding/IoT related application scenes. We mainly focus the quantization techniques for the DNNs in this thesis.

1.4 Objective and main work of this research

This work mainly researched in-memory computing-based machine learning classifiers and the responding column reduction algorithms. Traditional machine learning models, trying to maximize the model performance namely accuracy or loss optima, usually do not quantize the model parameters. They only manage to quantize parameters to some extent when the number parameters are very large, like deep convolutional neural networks with up to billions of network parameters, in order to balance the efficiency of employing resources of computing and storing. Based on the scene of application in this work which is power/energy-restricted embedding system applications, as the carrier of computing model, memory arrays like SRAM cells are with a limited structure and such carried model can only employ linear model with limited bits (1-Bit for 6T SRAM cells) of the resolution, leading to a necessity of quantization for parameters. Meanwhile, the original classification objective becomes a new non-convex mixed integer programming problem. As a result, the traditional numerical optimization methods like stratified gradient descent or Newton's method cannot be used directly given such a constraint. In addition, the linear classifier, whose classification performance is relatively weak because it only generates linear decision boundary, becomes even weaker for the existed nonlinear noises and hardware errors in the circuits and memory cells. This research employed the Error Adaptive Classifier Boosting (EACB) algorithm based on the quantized linear models fit for training models applied in embedded hardware and explored different methods used for approximate 1-Bit quantization of linear models. Different from general full-resolution models applying digital calculations, in-memory-based models use analogy characteristics of memory circuits like SRAM cells, the real model's performance can be degraded in that the existence of cells' nonlinearity. Within the simulations in this research, in order to validate the performance of the in-memory classifier with different strengths of nonlinearity namely time-dependent variability, we generated and inserted noisy data in different strengths (standard deviations of normal distribution) to simulate the variability, achieving the error-aware model training in the afterward training phase.

A general classification task usually is a multi-label classification one. For example, the handwritten digit dataset MNIST used in this research is a 10-label (0~9) classification task and 'all-versus-all' (AVA) strategy is applied to make the final decision, where the original 10-label task is reformed to 45 binary classification tasks, like 0vs1, …, 0vs9; 1vs2, …, 1vs9; ...; 8vs9. For a single classification loop, 45 binary classifiers make the binary decision for each binary task first and the AVA voter makes the final prediction based on votes each class got. The conventional implemented in-memory-based classifier used a fixed number of weak binary linear classifiers and combined them to a strong binary classifier for each binary task. However, in this research, we considered reducing the redundant numbers of weak binary classifiers based on the fact that difficulties may vary among different binary tasks. In order to explore the possibility of redacting the number of memory cell arrays used in the in-memory machine learning classifiers, we proposed heuristic algorithms based on several different criteria, after exploring the relationship between final AVA accuracy and the number of weak classifiers used in different binary tasks.

Binary neural networks (BNNs) have drawn much attention because of the most promising techniques to meet the desired memory footprint and inference speed requirements. However, they still suffer from the severe intrinsic instability of the error convergence, resulting in an increase in prediction error and its standard deviation, which is mostly caused by the inherently poor representation with only two possible values of -1 and +1. In this work, we have proposed a cost-aware layer-wise ensemble method to address the above issue without incurring any excessive costs, which is characterized by 1) layer-wise bagging, 2) cost-aware layer selection for the bagging.

Furthermore, not only the inherently poor representation with only two possible values of -1 and +1, and the training process with quantization also is an important factor that causes them suffered from the severe intrinsic instability of the error convergence, resulting in an increase in prediction error and its standard deviation. In this work, we have proposed a new training procedure with relaxed quantization to address the above issue without incurring any excessive costs, which reveals the possibility of employing relaxed quantization on both activations and weights, and the corresponding experiments are conducted to evaluate the proposed technique.

1.5 Constitution of this thesis

The arrangement of chapters is as follows:

Chapter 1 is the introduction, which introduced the background of this research, including the concept, development, and application of in-memory computing. Several works of literature were summarized to illustrate the current state of research in the area of in-memory computing. Finally, the chapter arrangement was introduced briefly.

Chapter 2 mainly introduced the proposed memory array reduction algorithms. Since the boosted model is based on the quantized linear classifiers and the actual energy cost of such a system is in positive relation with the total area of the SRAM array, the key to restraining energy cost is to reduce the total area of the SRAM arrays, i.e., the number of columns (weak classifiers). This chapter researched several heuristic criteria-based algorithms enabling the reduction of weak classifier numbers employed to the trained boosted model.

Chapter 3 presents the low precision neural network and its instability issue of conventional works and then proposes a layer-wise ensemble method for the binary neural network (BNN), followed by the experiments to compare the conventional methods and the proposed one the metrics of the stability and accuracy of the network.

Chapter 4 presents the conventional training process of BNN with relaxation technique and proposes a new training procedure that employs relaxations to both low precision weights and activations, followed by the comparisons between the proposed method and conventional training process in various cases of different optimizers or with and without batch normalization.

Chapter 5 concludes this thesis with the main contributions summarized.

Chapter 2 Improving bit energy efficiency of in-memory boosting classifier

Abstract

Nowadays, artificial intelligence is widely being applied on the Internet of Things and sensor signal processing area. However, with the computing model in a higher complexity, there is an increasing need for ultra-low-power MAC (multiply and accumulate) operation for input feature vector and model parameters in order to control the system power. Meanwhile, with the memory technology scaling, the increasing impact of hardware variation also demanding higher system resilience for error-adaptive processes and fault tolerance. To get over such challenges, the direction of in-memory computing, where the computation is performed within memory (like SRAM bit-cell), is highlight recently. A column reduction technique for an in-memory machine-learning classifier in 6T SRAM cells is discussed in this work, based on an error-tolerant boosting algorithm (a.k.a., error-adaptive classifier boosting, EACB). The proposed technique is mainly applied to the in-memory machine-learning classifier system wherein the weight of the linear model is restricted to 1 bit applicable for standard 6T SRAM cells, employing the EACB algorithm to recognize down-sampled handwritten digits. First, the number of columns of the boosted classifier is pruned. Second, three methods: greedy search, fast version of greedy search, and worst-care optimization, are discussed and implemented. Finally, the reduction effects of the proposed methods are compared. The simulation results show that besides the 11.50% column reduction from pruning, the proposed methods can further reduce 3.23%, 5.14%, and 5.49% of the column number on average, respectively, with similar accuracy to ensure that the corresponding part of the model can be reduced to achieve better energy saving.

2.1 Introduction

In-memory computing is implemented generally through the analogy characteristics of the memory circuit in different conditions and modes and its peripheral circuits. This research mainly the in-memory computing machine learning classifiers implemented based on SRAM cells.

Within SRAM cells based in-memory computing system, the parameters of the classification model (1-Bit quantized linear classifier) are trained and stored in the cell array of SRAM. Input data to be classified is transferred into analogy low voltage signals (<400mV) and connected with the model/memory array via word-lines of SRAM. According to the model parameter stored in the cell (+1 or -1), the differential voltage is formed on BL/BLB and summarized current results in the final binary decision, +1 or -1 by the comparator, which enables the 'sign' function. All the binary decisions get to the final multiclass decision after the all-vs-all voter.

The feature of such a system is that employing word-line-parallel computation enabled by

SRAM structure, the efficiency of calculating as well as the speed of making inferences is improved and the throughput of the system is enlarged, compared to a computing/storage separate system. Thus, the total cost of energy of such a system is reduced. Such a feature of in-memory computing system fits the scene of embedded always-on detecting applications.

The basic theory of SRAM-based in-memory computing classifier and the comparison between conventional computing/storage separate systems would be introduced from different points of view in succession in the following sections.

2.1.1 Theory of SRAM based in-memory computing

Fig 2.1 shows one column of cell array of standard 6T SRAM.

Each column of SRAM cell array is composed of word-lines (WL) in the horizontal direction, bit-line/bit-line-bar (BL/BLB) pair in the vertical direction, and the bit cells with an equal number of WLs. The data, either +1 or -1, is stored in the bit cell by setting the state of transistors with different bit-line voltages through 'write' mode.

As normal memory storing data, there are mainly two modes of SRAM employed:

1) When using the read mode, the WL connected to the address of the target to read is pulled up with high voltages. Then the data is read from the selected bit-lines.
2) When using the write mode, the WL connected to the address of target to write is pulled up with high voltages, similarly. Then set the bit cell state to 1 or 0, employing the stronger driver of BLs, compared to the bit cell.

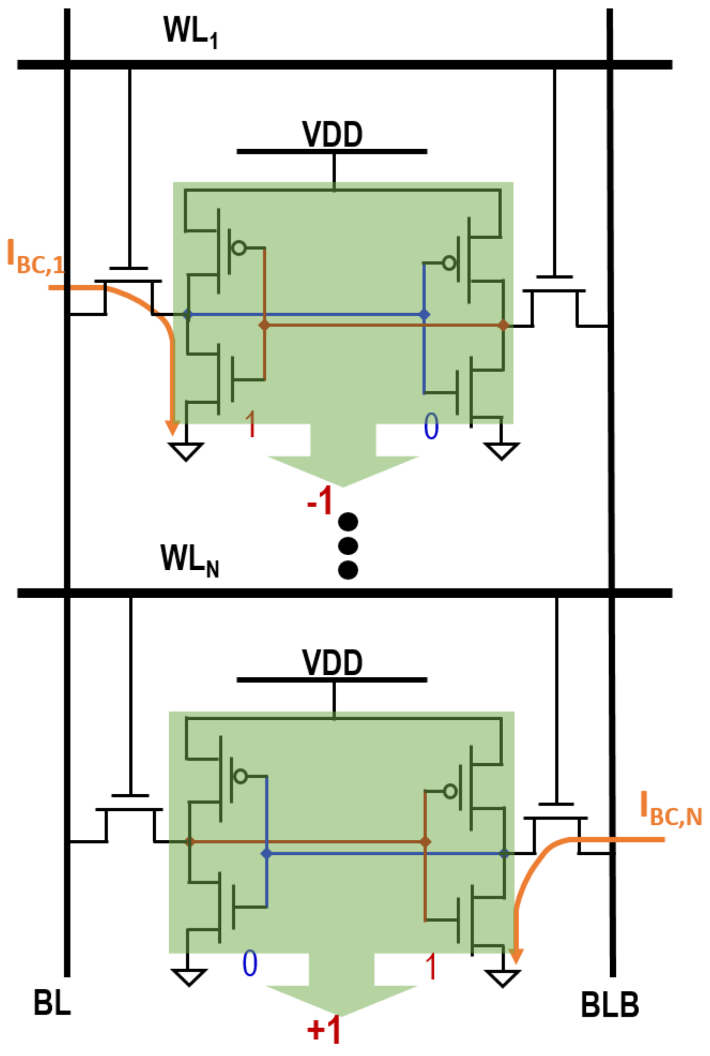


Fig 2.1 One column of cell array of standard 6T SRAM

When an SRAM array that stores the model parameters is treated as a binary linear classifier, the classification process per decision is as follows:

The BL/BLB is pre-charged first. Then the same analogy voltages, which are from the DAC converting the input data to the analogy signals, are applied to all the WLs. Because of the low voltage range (0~400mV), the state of the bit cells will not be changed. The responding bit cell current will be applied to either BL or BLB according to the data stored in the bit cell. Therefore, the voltage on the BL/BLB can be treated as a differential signal, which is equal to a multiplication between the input data and stored 1-Bit model parameters. Finally, after BL/BLB discharging, the accumulated BL current results in the final binary decision through a comparator implementing sign function[48].

There are several special points to distinguish classification process and normal modes:

Different from the high voltage applied on the BL in normal read/write mode, in order to hold the classifier's parameters unchanged, the low voltage signal is employed on BLs.

Since each column of SRAM bit cells forms a binary classifier which needs to be multiplied by all the input feature vectors of one input sample, all the WLs should be driven in each decision-making cycle. In addition, if the classifier is extended into a multi-label classifier employing AVA strategy, all WLs also should be driven at the same time and sending the analogy signal standing for the input feature vector. Thus, it is necessary that the input data is in the shape of a 1-dimension vector or should be reshaped into such a dimension.

2.1.2 Comparison of in-memory computing and separated system

As the parameters of a binary classifier stored in a column of bit cells need to be quantized to 1-Bit, the accuracy and performance of such a classifier are strongly degraded. Therefore, such a binary classifier can only achieve a somehow quite weak classification boundary, viewed as a 'weak' classifier. A solution to improve the accuracy of the classifier is to employ some ensemble learning algorithm like Boosting[49], combining several relatively weak classifiers' decisions with weights to an ensemble classifier with stronger performance. In order to achieve the binary classifier with better single classifier accuracy, Z. Wang et al.[50] proposed 'Constraint-Resolution Regression' algorithm employing mixed-integer-programming solver[51] to train the binary classifier, reaching a higher accuracy compared to the performance of 1-Bit quantized binary classifier directly. The writers employ such binary classifier as the base learner and apply a variation of Boosting algorithm to get the ensemble classifier. With the experiment on the classification of the MNIST dataset down-sampled from 28×28 to 9×9 on a standard 128×128 6T SRAM, the final 10-way classification accuracy of 90% is achieved with the energy cost of only 0.63nJ per decision. Compared to traditional storage/computing separately implemented system with a cost of 71.61nJ, the proposed system achieved 113x reduction of energy cost and 1525× reduction of EDP.

The detailed comparison between in-memory computing based and traditional storage/computing separately based system can be considered in the following respective[52]:

1) Bandwidth. Within the separated system, multiple accesses to the addresses of memory where the computing model is stored are necessary. Within the in-memory-based system, the in-memory computing enabling the computing model embedded into the memory cells, and all the feature vectors are available instantly. Thus, the bandwidth of the system is enlarged.
2) Energy. With regard to the energy of the two systems, both the discharge of BL/BLB is the main part. However, within the separated system, higher resolution means more BLs. Though the WLs are driven at the same time within the in-memory-based system, the low voltage employed contributes to rather lower energy of the whole system.
3) Delay. Within the traditional separated system, the total number of accesses and processing cycles increases as the input data scales, leading to the increase of system

delay. However, the massive parallel processing that existed in the in-memory computing system enables the relatively stable BL swing, contributing to the reduction of system delay.

4) SNR. Within the separated systems employing the low BL swing design, a rather high SNR can be reached. On the other hand, in the in-memory computing system with a variety of states of each column of bit cells according to the model's parameters, the SNR is changing in dynamic. In addition, the differential voltage decreases with the capacitance of BLs, leading to a relatively lower signal passed through the comparator finally. Thus, the SNR of the system is decreased.

After the previous comparisons, it can be found that compared to the traditionally separated system implementation, the in-memory computing system achieves relatively higher bandwidth, lower delay as well as lower energy, with the cost of the system to an extent. Thus, such a trade-off of performance can bring an alternative to some embedding hardware systems and applications with the scene and demands of low delay and low energy cost.

2.2 In-memory machine learning classifier

2.2.1 Structure of SRAM based in-memory classifier

The simulations of this research mainly focus on the classification algorithms applied in the in-memory computing classifier, extending to the responding memory array reduction methods. The employed system in this research is of the similar structure of the implemented MNIST classification system in the literature[52]. The diagram of the structure is shown in Fig 2.2.

The main part of the structure is a 6T SRAM memory array, where each bit cell $C_{i,j}$ stores a 1-Bit state standing for the responding model parameter +/-1 and each column of m bit cells consist of a 1-Bit 'weak' linear model, since it has only 1-Bit resolution parameters and can

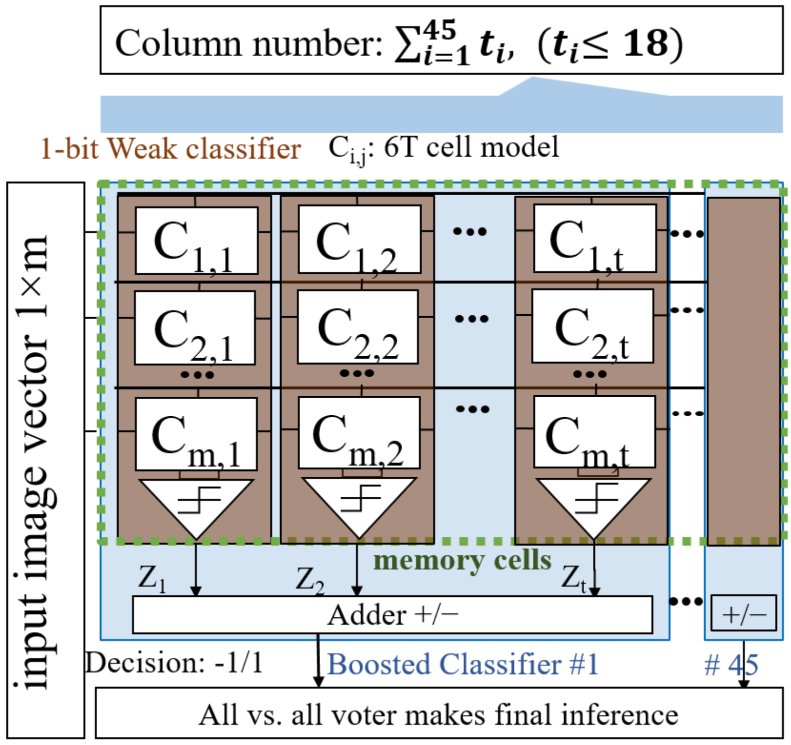


Fig 2.2 Structure of in-memory classifier

only generate linear decision boundary with limited classifying ability.

To improve the performance of such weak classifier, the Boosting algorithm (see Fig 2.3), which will be introduced in detail, is employed to train boosted strong classifiers combining several weak linear classifiers.

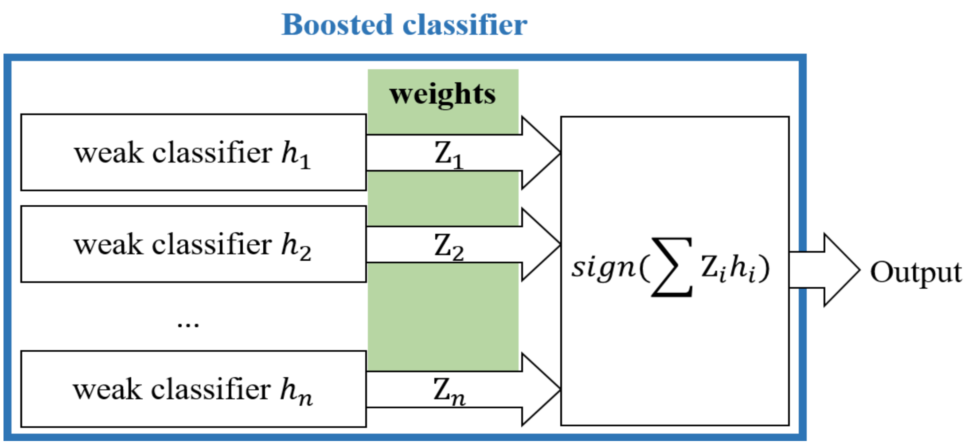


Fig 2.3 Diagram of Boosting classifier

With regard to the multi-label classification task, all-vs-all strategy is employed to generate the final classification decision from all 45 boosted binary classifiers.

There are some things which should be noticed that the implementation of such a hardware system needs not only a standard 6T SRAM but certain peripheral circuits like WLDAC which is between the memory arrays and input feature buffer, address decoder, and BL drivers, etc., in order to support all the features needed of the in-memory computing system. Since the main objective of this research is to simulate and research the algorithm part of the in-memory-based machine learning classifier, the detailed discussion for the peripheral circuits will not be included in the following part of this work.

2.2.2 Problems of conventional SRAM based in-memory classifier

Though in-memory classifier achieves quite great progress in the respective of reducing energy of system compared to the traditional separated systems, such a system requires still quite a lot of numbers of memory arrays, up to 810 columns for the MNIST task, in order to fit the input data as well as the hardware errors and nonlinear noises existed in the SRAM bit cells, contributing to about 76% of the total energy costs in each cycle by the classification processing.

In addition, through the EACB algorithm, the conventional implemented system corrects circuit non-linearity which is inborn from manufacturing. But it cannot correct acquired time-dependent variability for uncertain strength of variation. In the following chapters, this research explores the multiple classifiers which try to correct not only circuit non-linearity but also time-dependent variability within a range via employing several error-adaptive models pre-trained with different scaled variations.

Based on the theory of in-memory computing, it is necessary to transmit the converted input signals to all the bit cells through the WLs at the same time. Thus, the energy used for the classifying process is related to the number of all of the memory bit cells, which is equal to the size of the parameter matrix of the classification model. Thus, reducing the number of bit cell columns to improve the energy cost in the classifying process is a critical method to improve the total energy cost of the system. The detailed introduction will be illustrated in the following chapters.

2.3 Error Adaptive Classifier Boosting (EACB)

2.3.1 AdaBoost algorithm

Within the process of Boosting algorithm, the first base learner is trained using initial dataset, then, adjust the distribution of the dataset according to the performance of the trained base learners and train the base learners iteratively[53]. Detailed implementations include paying attention to the wrongly classified samples, increasing the weights of the wrongly classified samples and increasing the number of the wrongly classified samples, i.e., resampling, etc. Based on the adjusted dataset, iteratively repeat the training & reweighing process until some preset conditions satisfied, like maximum number of base learners and threshold of boosted accuracy. Finally, combine the trained base learners by weights and output the cumulative decision[54]. Among the various of boosting algorithms, AdaBoost[55][56] is a representative one and the ‘additive model’ based formation is simplified relatively[57], as follows

$$H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$$

(2.1)

From (2.1) we can see that this is an additive model by summing the decisions of the base learners by weights. The loss function is an exponential loss function (binary classification case), as (2.2), where D is distribution of the dataset

$$J_{exp}(H|D) = \mathrm{E}_{x\sim D}[e^{-f(x)H(x)}]$$

(2.2)

The loss is exponential expectation of errors given the dataset with some distribution.

The process of AdaBoost algorithm is as follows:

Given dataset $(x_1, y_1), \dots (x_m, y_m), x_i \in X, y_i \in Y = \{1, -1\}$, initialize the data distribution $D_1(i) = 1/m$. For iteration $t = 1, \dots, T$, repeat:

1) Train the base learner based on distribution D_t
2) Make a weak prediction $h_i: X \rightarrow \{1, -1\}$ by the trained base learner and calculate the error ϵ_t of prediction

$$\epsilon_t = \mathrm{Pr}_{i\sim D_t}[h_t(x_i) \neq y_i]$$

(2.3)

3) Calculate the weight of the trained base learner by ϵ_t

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$$

(2.4)

4) Update the distribution D_t by weight α_t, where Z_t is normalizing factor

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t}\exp(-\alpha_t y_i h_t(x_i))$$

(2.5)

Until iterations stop, the final prediction of ensemble classifier is

$$H(\boldsymbol{x}) = \mathrm{sign}(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}))$$

(2.6)

It should be noticed that a ‘base (weak) learner’ also needs somehow stronger than a prediction by random guesses which can be defined by a random predictor with an accuracy of about 0.5. Different from other ensemble algorithms, it is necessary that the base learners can be trained from the dataset with the same distribution. For learners who accept the weights for samples, the boosting process can be implemented by updating the sample weights. For those who cannot be trained with sample weights, the boosting process can be implemented by resampling the dataset according to the performance of the previous base learners, such as subsampling the samples correctly classified, oversampling the sample wrongly classified, etc. Compared to the implementation of reweighing, the advantage of resampling is that when the base learner cannot satisfy the basic condition of accuracy larger than 0.5, the iteration can get into the next round directly, achieving 'restart' to avoid weak performance of the ensemble classifier caused by under-training.

AdaBoost is widely applied in various prediction tasks for its simplified theory and quite good performance. In addition, it has many variation algorithms being proposed referring to different scenes of applications and demands. Within the area of in-memory embedding systems with limited computing/memory/power resources as well as hardware errors and nonlinearity such that normal AdaBoost algorithm is hard to be applied, a variation of AdaBoost, 'Error adaptive classifier boosting' (EACB) algorithm is proposed as a solution to such problem.

2.3.2 Error Adaptive Classifier Boosting algorithm

The strict resource/cost limits of embedding systems and instinct hardware errors, nonlinearity can lead to the degradation of inference models being applied in practice, which is always a big challenge in areas like embedding signal processing and machine learning[58]. With regard to research on such problems, the direction of constructing machine learning based data-driven models to analyze target data as well as the environment data, including hardware errors and nonlinear noises, makes some progresses, of which 'Error adaptive classifier boosting' (EACB) is a representative one.

EACB has some similarities with FilterBoost[59]. Original AdaBoost needs base learner can be trained with the entire dataset in some distribution in each iteration and update the weights for samples in the next iteration based on the error rate of current base learner. However, it needs huge computing/memory resources to process the entire dataset in each iteration. To solve this problem, FilterBoost and EACB convert the entire dataset to some samples in small batch of from the original dataset generated. Thus, the necessary computing and memory resources are largely reduced. Finally, a similar performance can be achieved by adjusting the number of samples in one training batch.

Within FilterBoost, the reweighing is implemented through the designed filter calculating sample weights and corresponding acceptance probabilities, where the accessibility to the entire dataset is necessary. In addition, the samples in the training batch are divided into two parts of which one is used to train the current base learner and the other is used to calculate the final classifier weight based on its accuracy.

Different with FilterBoost, it is considered that within the scene of embedding sensing systems, since the accessibility to the entire dataset is hard to achieve, it is usually applied that generating training batches from a data stream. Hence, with the procedures of the framework of AdaBoost algorithm adjusted, EACB employs the previous training batch to train the current base learner and uses current training batch to evaluate the performance of current learner, calculating the classifier weight and reweighing the training batch by ensemble classifier. The detailed procedures of EACB are as follows (case of binary classification):

Generate training batch $(X_t, y_t), y_t(i) \in \{1, -1\}, i \in \{1, \dots, N\}$ with batch size of N from dataset S, where $t = 0, \dots, T$ standing for number of iterations. Initialize ($t = 0$) the prediction of ensemble classifier for X_t as $F_0(X_0) = 0$.

For iteration $t = 1, \dots, T$, repeat:

1) Update sample weights of previous $(t-1)$ training batch based on the previous prediction of the ensemble classifier and normalize

$$D_{t-1}(i) = 1/(1 + \exp(y_{t-1}(i)F_{t-1}(X_{t-1}(i))))$$

$$D_{t-1} = D_{t-1}/Z_{t-1}$$

(2.7)

2) Train the current (t) base classifier with previous $(t-1)$ training batch and corresponding distribution

$$h_t = \mathrm{train}(X_{t-1}, y_{t-1}, D_{t-1})$$

(2.8)

3) Get current training batch (X_t, y_t) from S
4) Calculate error rate ϵ_t of current base classifier on current training batch
5) Calculate classifier weight of current base classifier

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

(2.9)

6) Update current ensemble classifier

$$F_t = F_{t-1} + \alpha_t h_t$$

(2.10)

Obtain the final prediction

$$H = \mathrm{sign}(F_t)$$

(2.11)

With the procedures of EACB, it is achieved that only one training batch is needed in an iteration, thus, the necessary resources are further reduced, and the ability of online learning makes the training models deployed within the embedding hardware and fitting the environment data.

2.4 Column reduction technique for in-memory boosting classifier

Through the previous chapters, it was introduced that the performance of 1-Bit Constrained-Resolution Regression (CRR) based classifier can be improved by boosting algorithm like EACB and to get over time-dependent variability, the proposed multiple classifiers can be applied to improve the performance of the model, bringing a stronger robust system towards the variability. In this chapter, from the perspective of reducing the column numbers of model, the methods to decrease the energy costs further will be explored.

2.4.1 Origin of the number of the memory column arrays

Within the 1-Bit quantized linear model-based in-memory boosting classifier, it is necessary to multiply the input feature vector and all 1-Bit linear models per decision, which is the main source of the computation costs. From the contents in the previous chapters, it is also critical that the final accuracy is in positive correlation to the number of EACB iterations. However, each iteration means one more column in the SRAM memory array correspondingly, increasing the necessary energy cost per classification decision. In addition, the nonlinearity and variability in the bit cells make the actual number of arrays of such in-memory classifiers scale to reach a satisfying performance of classification with more energy costs.

Therefore, it can be considered that improve the system cost performance by reducing the number of memory bit cells that are necessary within the computation. The number of bit cells for computing is mainly decided by the multiplications of its row number and column number, where the row number is the same as WLs number, i.e., number of features and the column number is the same as the 1-Bit linear weak classifiers. Since the number of features is varying referring on the target data and the application as well as the preprocessing methods, it is always a case-by-case value. This research does not limit the type of input data and the preprocessing methods, we only consider the reduction of memory arrays column number, i.e., reducing the number of weak classifiers, to reduce the total energy cost of the system.

However, reducing the number of weak classifiers means the less iterations of boosting, which is easily leading to under fitting of the model and loss of accuracy. Thus, holding the performance of the classifier, to reduce the total number of weak classifiers is the target of this research on the column reduction methods for in-memory classifier.

2.4.2 Redundancy of memory arrays

Consider the redundancy of the memory column arrays, that is the redundant weak classifiers. Since the process of boosting training is to fit the data with some distribution iteratively, which is a process of decreasing the residual to the re-distributed dataset gradually. However, when the residual gradually deceased to a small value, the newly trained weak classifiers contribute less, which can be somehow redundant for the boosted classifier. From the simulation result of the previous chapter, following Fig 2.4 shows the final accuracy with the number of iterations with the input data of 11×11 size without variability, employing the implementation of conventional work.

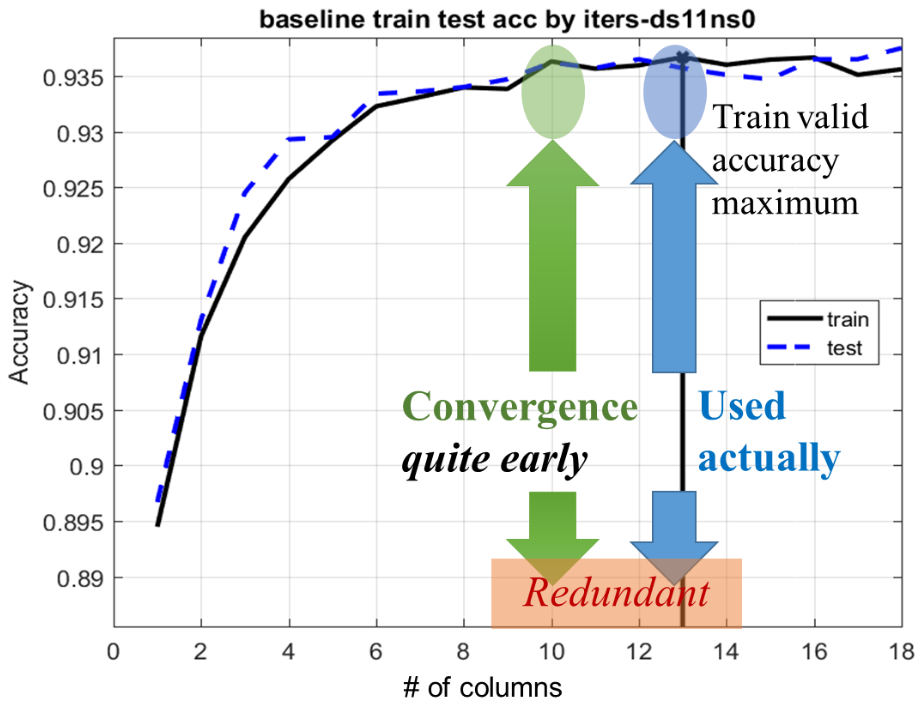


Fig 2.4 Redundant spaces of conventional work

From the figure, we can find that the model by the conventional implementation, the column number t, which is 13, for each boosted classifier is chosen by the accuracy maximum of train valid dataset. However, we can see that after about 10 iterations, it probably comes to convergence. So, there can be some redundant spaces for us to optimize, which is exactly the key point this research focus on.

2.4.3 Baseline of performance

Because of the target of this chapter, the performance of the classifier not only includes the classification accuracy but the total number of columns (weak classifiers) as well, so the baseline here also has 2 metrics: accuracy and total column number. To be simplified, note the dataset with a down-sampling size of 11×11 and variability strength of 0 as D(11,0)

For example, as D(11,0), each boosted classifier comes to the maximal accuracy 0.936 at iteration 13, meaning each boosted binary classifier has 13 weak classifiers in 13 columns. Since all-vs-all strategy is employed on 10-label task, the total column number of such classifier is 45×13=585. Therefore, for dataset D(11,0), the baseline performance of the model is accuracy=0.936 and number of columns=585 and the optimized model should compare with those metrics.

2.5 Heuristic column reduction algorithms

This section mainly researches the column reduction algorithms applicable for the in-memory Boosting ensemble classifiers with the 1-Bit linear model as the base classifier. In the previous section, we know that for D(11,0), 585 columns are needed. However, the difficulty is that the possible searching space for such a problem is quite large. What's more, the process of boosting is iterative, not independent which the current iteration needs the performance of the previous iteration to reweighing the samples, thus, the original order in the boosting process should be protected. This section explores the column reduction methods from the heuristic algorithms' perspective.

2.5.1 Pruning columns for different boosted classifiers

Within the all-vs-all (AVA) voting strategy, the original 10-class classification problem is divided into 45 binary classification tasks. However, the difficulties for different binary tasks are also different and the best EACB iteration for each boosted classifier also varies. For example, for some hard classification tasks, it may need nearly 10 or more iterations to reach the maximum accuracy, while 3 or fewer iterations can be enough for some simple classification tasks.

The first step to reduce the columns is by pruning the columns for different boosted classifiers, which is cutting down the columns after the boosted classifier reaching its maximum of accuracy. Fig 2.5 shows the comparison of columns of fixed same iterations and columns with

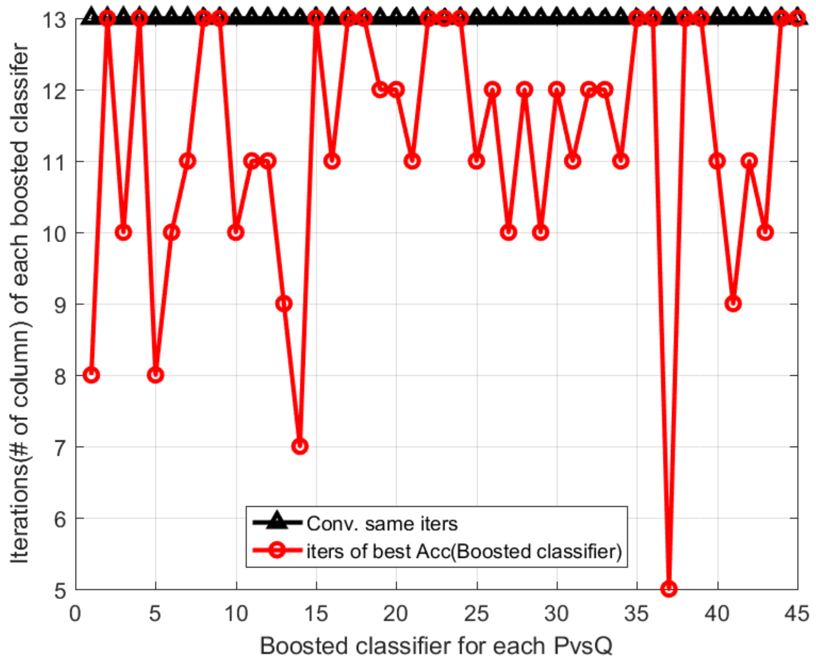


Fig 2.5 Conventional same iterations VS. iterations of best accuracy

pruning for D(11,0).

The criterion of pruning is simply selecting iterations of best train accuracy of each boosted classifier, rather than conventional same fixed EACB maximum iteration.

The comparison of accuracy of model with or without pruning by iteration is shown in Fig 2.6. Both models have an accuracy at the same level and model with pruning gets earlier convergence compared to model without pruning. For D(11,0), this reduces 206 columns from 585 to 379.

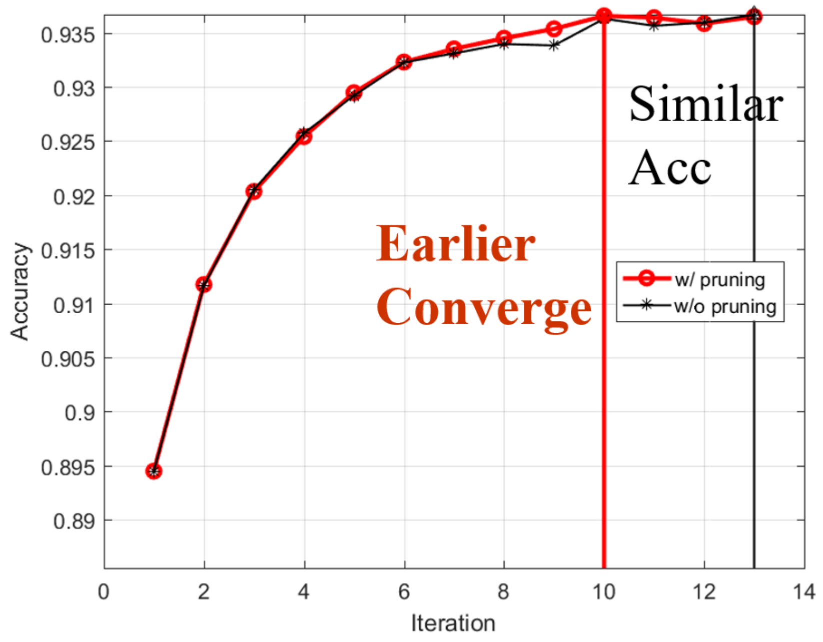


Fig 2.6 Accuracy with pruning VS. accuracy without pruning

From the Fig 2.6, it can be seen that through pruning process, the model converges in just 10 (maximal) columns, reaching the similar performance as the original model.

It should be noticed that the column numbers after pruning is the maximal number, compared to the fixed original column number, which stands for the maximum of the iterations(columns) of boosting process, because from the Fig 2.5 we can see that after pruning the boosted classifiers' column numbers are not fixed and vary in a range, but it is controlled that the maximum of the columns cannot surpass that of the original model.

2.5.2 Reducing columns by greedy search

In this and next section, two column reduction algorithms will be proposed based on different searching strategies, which can be applied to the model after being processed by the method of pruning proposed in the previous section.

First, consider the process of the columns growing, where for each binary task the boosted classifier will iteratively train $t(1 \leq t \leq T)$ weak classifiers in order. In this independent process, the boosted classifiers for other binary tasks are not considered, which is a searching process for the local optima. However, the final prediction is given by the all-vs-all voter, leading to the circumstance that many boosted classifiers have reached the maximum by 12 columns on their own binary task for example, but the final AVA accuracy is lower than that of the model with maximum columns of 10.

Therefore, with protecting the original training order of boosting, consider changing the column growing strategy for the boosted classifiers, according to the obtained history training accuracy records from the original training process. In each step of growing column, consider the contribution to the final AVA accuracy of growing the current column from the initial state, i.e., number of columns as $\sum_{1}^{45} t_i = 45, t_i = 1, i = 1, \ldots, 45$ to the final state. For example, for D(11,0), the final state is that the column number is $\sum_{1}^{45} t_i = 379$, where t_i is corresponding column number of boosted classifiers. The final accuracy for the state in each step will be saved and the target is to find one middle state with less columns, whose final accuracy is also higher than that of the final state.

The detailed strategy of selecting the boosted classifier to grow a column in each step, a greedy-search-based algorithm is proposed in this research. According to the gain to the final AVA accuracy of each boosted classifier that has not grown to the maximal column number, only the boosted classifier with the maximal gain to the final AVA accuracy will be selected to grow one column.

The procedures of the proposed algorithm are as following pseudocode in Fig 2.7.

```
// Greedy search
// index begins from 1
input: $col_{prn}$ // column number of pruned model
output: $col_{grd}$ // column number of greedy model
          $acc_{final}$ // accuracy of greedy search by iteration
// init
 1. for i from 1 to 45
 2.     $col_{grd}[i] = 1$
 3. endfor
 4. $acc_{final} = zeros(1, 1 + sum(col_{prn}) – sum(col_{grd}))$
 5. $iter = 1$
 6. $acc_{final}[iter] = calcAccAVA(col_{grd})$
// iter
 7. while $sum(col_{grd}) < sum(col_{prn})$
          // one step greedy
 8.     $iter = iter + 1$
 9.     $acc_{test} = zeros(1,45)$
10.     for i from 1 to 45
11.         if $col_{grd}[i] < col_{prn}[i]$
12.             $col_{grd}[i] = col_{grd}[i] + 1$
13.             $acc_{test}[i] = calcAccAVA(col_{grd})$
14.             $col_{grd}[i] = col_{grd}[i] – 1$
15.         endif
16.     endfor
17.     $i_{max} = argmax(acc_{test})$
18.     $col_{grd}[i_{max}] = col_{grd}[i_{max}] + 1$
19.     $acc_{final}[iter] = max(acc_{test})$
          // end of one step greedy
```

Fig 2.7 Greedy search column reduction method procedures

'calcAccAVA' means calculating the all-vs-all accuracy of the model. 'col' is an array to record the total column number in each iteration step, while 'acc_{final}' is also an array to record the final AVA accuracy. The footprint 'prn' is as the model after pruning and 'grd' is referred to proposed greedy search. 'acc_{test}' is the temporal array with 45 elements to record the responding AVA accuracy of boosted classifier if any of them grows one column.

The following Fig 2.8 shows the comparison of the pruned model and the model processed by greedy search algorithm.

The Fig 2.9 shows the result of column number of each boosted classifier by greedy search labeled as the same rule for D(11,0).

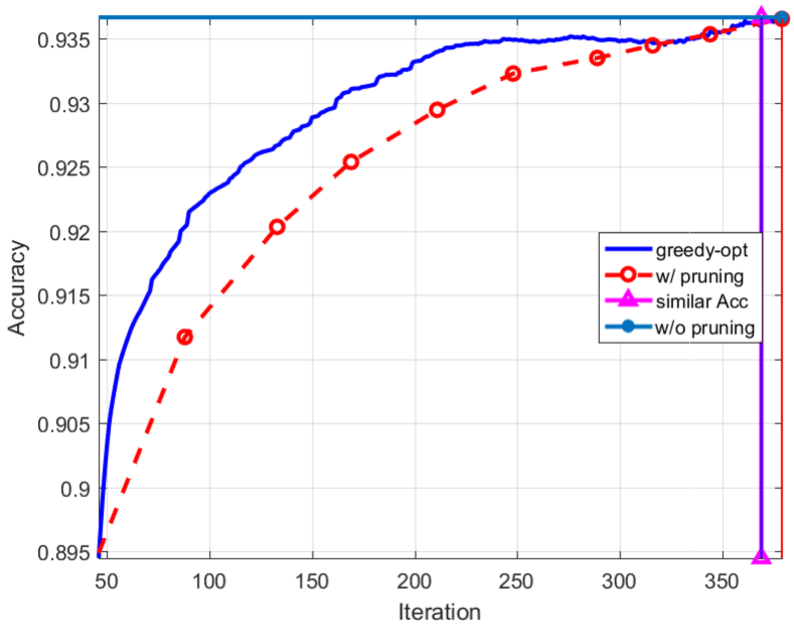


Fig 2.8 Accuracy by iteration of greedy search

The curve labeled with 'greedy-opt' represents the accuracy changing with the total column number (iteration of greedy search) and it reaches the same accuracy ('similar Acc') as the result with pruning before the end point which achieves the reduction for the columns. The curve labeled with 'w/ pruning' and 'w/o pruning' is the same as the result with and without pruning.

It shows that the accuracy path of greedy search is more near to the left-top corner within most of the iterations meaning that with same columns it finds column setting with higher accuracy than the pruned result by greedy search.

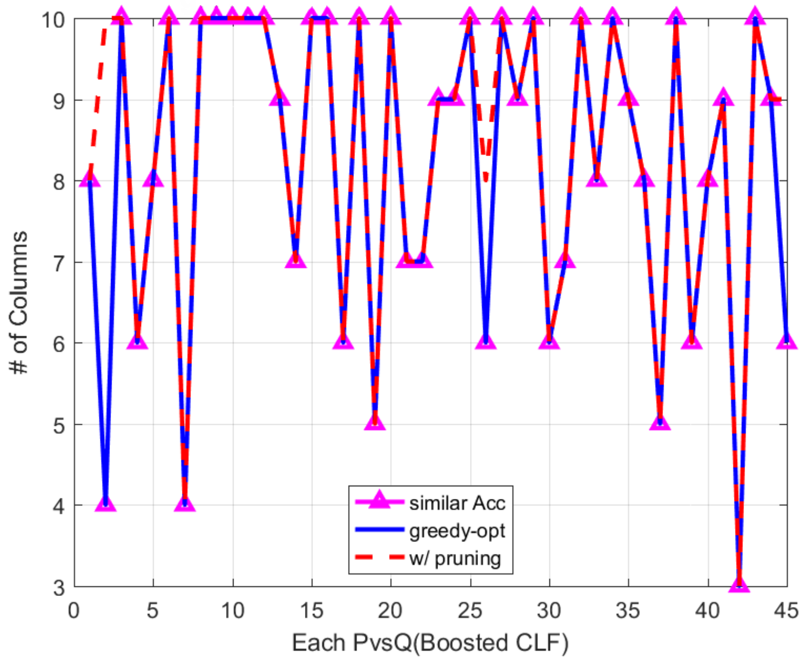


Fig 2.9 Column number for each boosted classifier by greedy search

2.5.3 Reducing columns by fast greedy search

However, only one column is grown in each step, leading to a huge cost of computation. To improve the algorithm, consider the column growing process first, which is shown as following Fig 2.10.

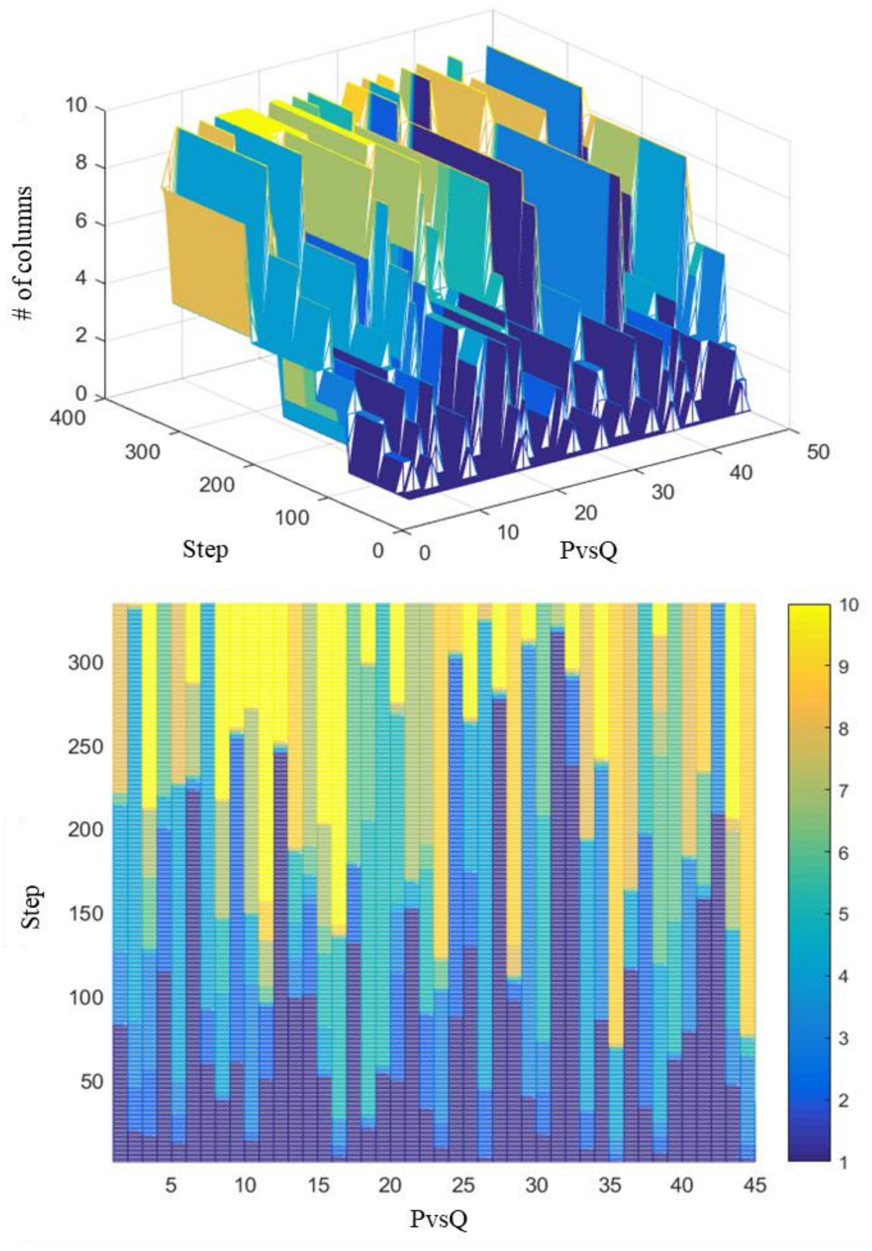


Fig 2.10 Column growing process (greedy)

It is shown that there is no strong trend of incensement of columns but rather sparse and existence that column number of the same boosted classifier is increased within some continuous steps. Thus, it is considered that the growing of columns in several steps can be integrated, rather than by decision in each step, which is quite resource consuming. We consider converting the original greedy column reduction algorithm to a fast version where an extra procedure is added that after growing one column to the selected boosted classifier in any step, repeat growing one more column to the boosted classifier until the final AVA accuracy is not increased. The detailed description is as following Fig 2.11

To be simplified, the initial process is as the function ‘init()’ and the procedures of original greedy process in each step are simplified as function ‘oneStepGrd()’.

```
// Fast greedy search
**input**: $col_{prn}$
**output**: $col_{grdfast}$, $acc_{final}$
// init
1.  $col_{grdfast}$, $acc_{final}$, iter = init($col_{prn}$)
2.  while sum($col_{grdfast}$) < sum($col_{prn}$)
    // one step greedy
3.      $col_{grdfast}$, $acc_{final}$, iter, $i_{max}$ = oneStepGrd()
        // fast greedy
4.      while $col_{grdfast}[i_{max}]$ < $col_{prn}[i_{max}]$
5.          $col_{grdfast}[i_{max}] = col_{grdfast}[i_{max}] + 1$
6.          $acc_{fast}$ = calcAccAVA($col_{grdfast}$)
7.          if $acc_{fast} > acc_{final}[iter]$
8.              iter = iter + 1
9.              $acc_{final}[iter] = acc_{fast}$
10.         else
11.             $col_{grdfast}[i_{max}] = col_{grdfast}[i_{max}] – 1$
12.             break
13.         endif
14.     endwhile
15. endwhile
```

Fig 2.11 Fast greedy search column reduction method procedures

At first, the procedures of initialization are the same as previous, obtaining the accuracy array, column number array and variable ‘iter’. The procedures in the first while-loop are also the same as normal greedy noted as function ‘oneStepGrd()’, finally reaching the index of selected boosted classifier ‘i_{max}’ and current iteration number ‘iter’.

Then entering the addition ‘fast’ part inner loop, repeat growing to the selected boosted classifier if it can bring gain to the final AVA accuracy, otherwise break the inner loop, and enter the next normal greedy step. The stop condition is the same as the normal version, whether the total number of columns is over that of the pruned model.

The similar accuracy result of fast greedy search algorithm is as following Fig 2.12, while the original greedy search is in shallow color and noted as 'Original'

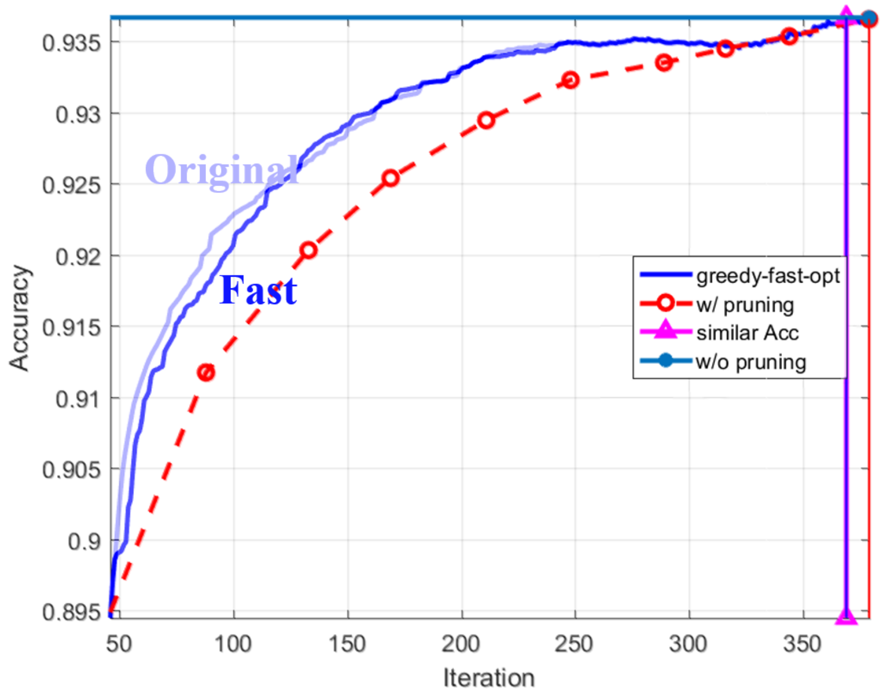


Fig 2.12 Accuracy by iteration of fast greedy search

Fig 2.13 shows the result of column number of each boosted classifier by fast greedy search. Through the fast greedy algorithm, the more efficient processing is achieved. According to

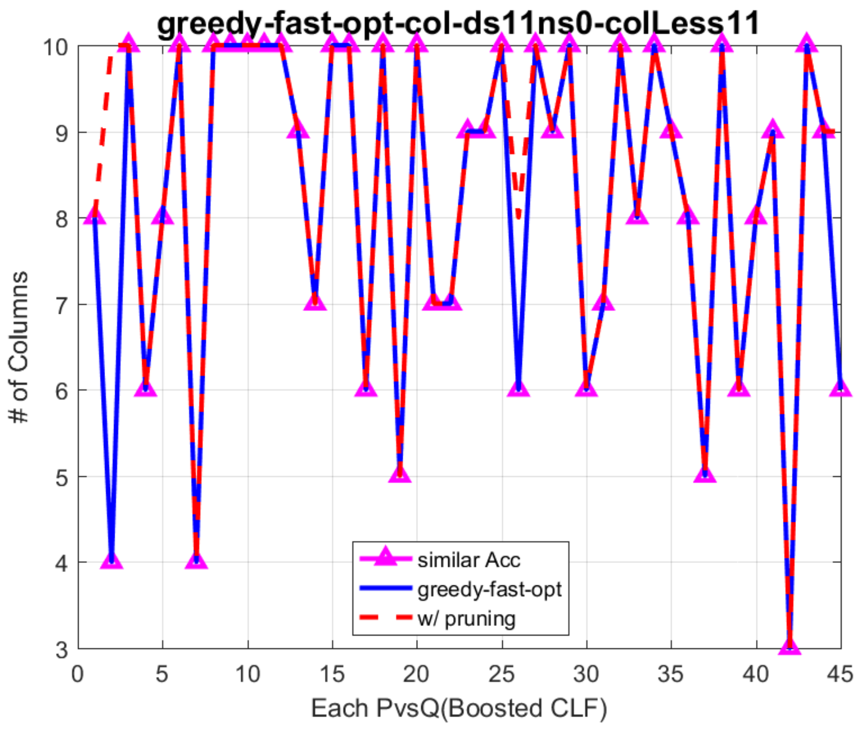


Fig 2.13 Column number for each boosted classifier by fast greedy search

the timer records, compared to the original greedy search, over 50% of the processing time can be saved. It reaches a similar result of 11 columns reduced with similar accuracy on D(11,0).

Similarly, the detailed column number growing process is given as following Fig 2.14.

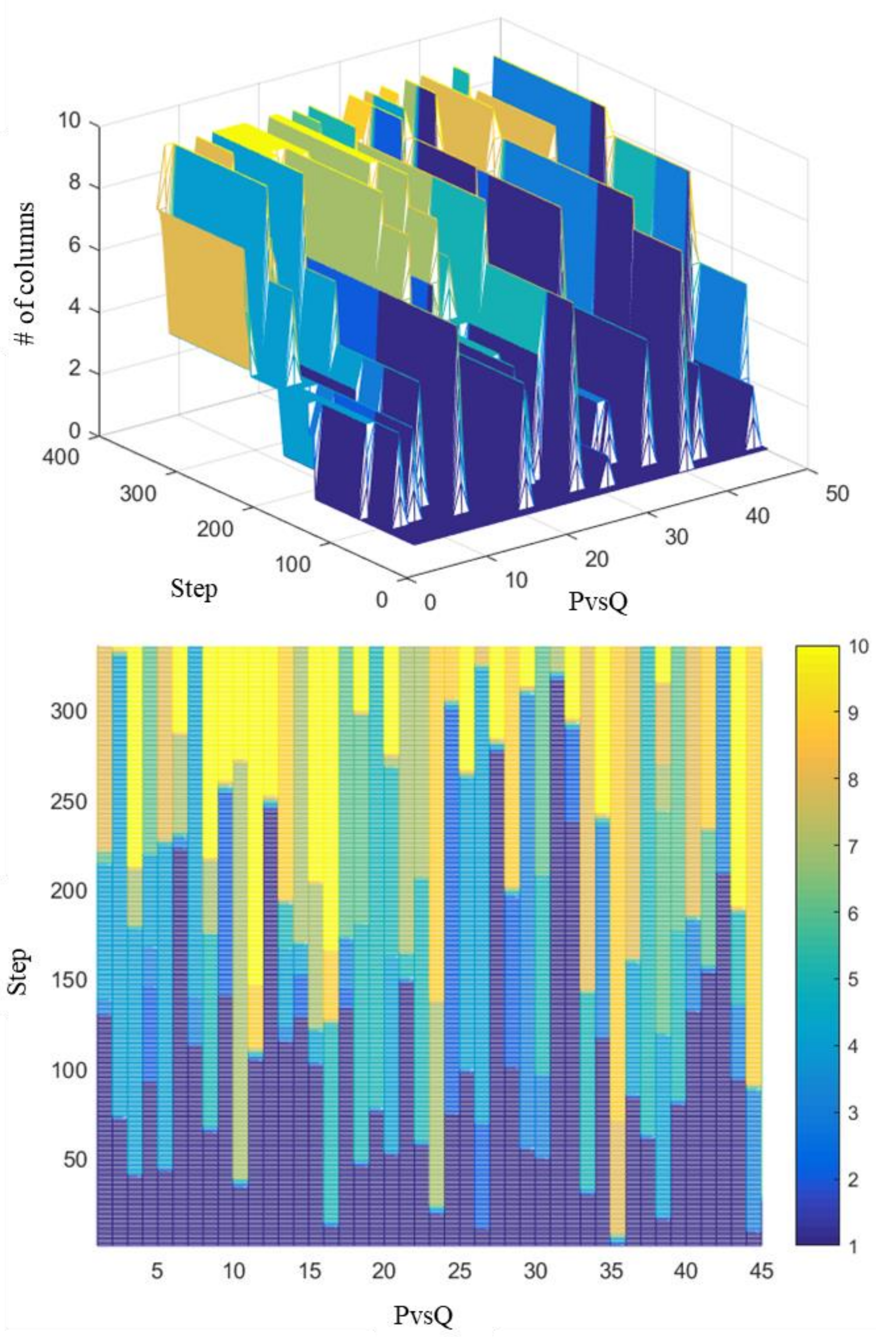


Fig 2.14 Column growing process (fast greedy)

As shown in the figure of column growing process, several columns are grown continuously to the same boosted classifier in continuous greedy step, rather than one column in one step, which saves the processing time greatly. By comparing the process of fast version with the normal version, it can be found that the process of fast version becomes sparser but is similar with the normal version.

2.5.4 Reducing columns by worst-care

In this section, another heuristic column reduction algorithm based on a special rule will be introduced.

Through greedy search algorithm which is based on the rule of maximal accuracy gain first, the local optima of training phase are converted to the maximum of final AVA accuracy in each search step, achieving the reducing the columns finally.

Again, consider the difference of difficulty of 45 binary classification tasks. Such difference leads to the great differences in performance of boosted classifiers on their binary task, which will have an impact on the performance in the final all-vs-all voting phase. From this point, there is a problem with the greedy search method, which is that the selected optimal state in each search step is not the global optimal state of all steps, for the current selection will affect the selection in the next step. If there is a local optimal state in the early phase selected by the algorithm and not manage to escape from it, the final performance may be worse.

To avoid solving the global maximum, which is relatively difficult to solve, we can consider another criterion from the point of different difficulties of subtasks. First, see Fig 2.15 as the comparison of the boxplot of binary classification accuracy on the 45 binary tasks and the final AVA accuracy in all 20 experiments.

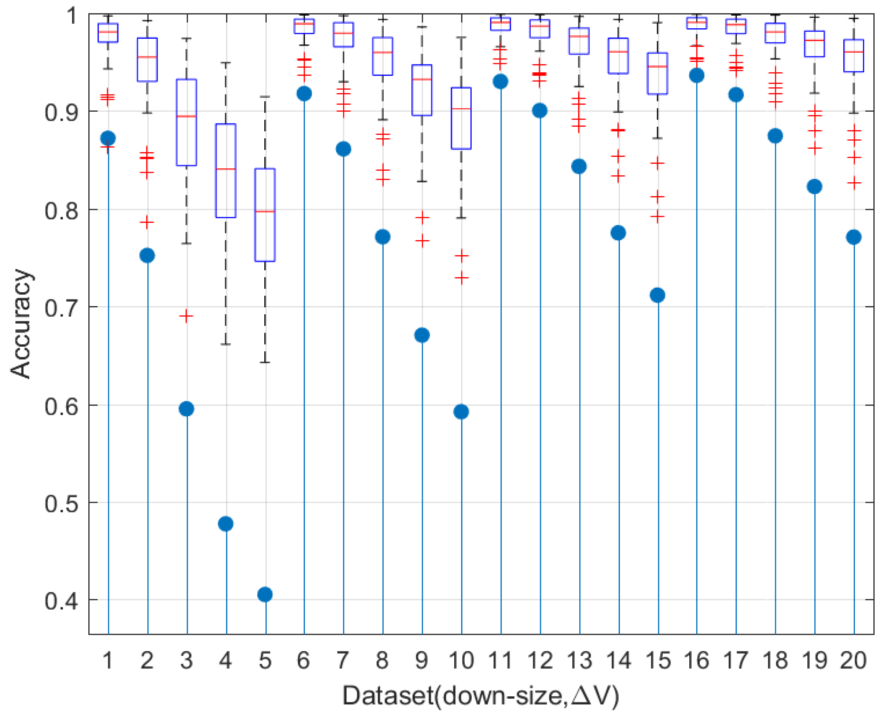


Fig 2.15 Binary accuracy VS final AVA accuracy

In Fig 2.15, the x-axis stands for the index for the different dataset in totally 20 combinations of down-sampling sizes {5×5, 7×7, 9×9, 11×11} and strengths of variability {0, 50mV, 100mV, 150mV, 200mV}. And the y-axis stands for the accuracy of classification in two groups. One group is the boxplot, standing for the statistics of binary accuracy of 45 boosted classifiers on each own subtask, noted as 20 boxes in each different dataset setting. The other is the final AVA in each dataset, noted as solid circles '●'.

From Fig 2.15, it can be found obviously that among all 20 datasets, the median level of the binary accuracy, marked as '-', greatly differs with the lower bound, marked as outliers '+'. In contrast, the final AVA accuracy marked as solid circle '●' is quite approximate with the lower bound among many datasets and some of them even overlap together.

Since the lower bound of the binary accuracy means the boosted classifier with the worst performance on its own subtask, it can be considered that the weakest boosted classifier affects the final AVA accuracy.

With the criterion that grow column of the weakest boosted classifier firstly within each heuristic search step, we propose the 'worst-care' selecting column reduction algorithm in this search.

The detailed procedures are as following Fig 2.16.

```
// Worst-care selecting
input: $col_{prn}$
output: $col_{wc}$, $acc_{final}$
// init
1.  $col_{wc}$, $acc_{final}$, iter = init($col_{prn}$)
2.  while sum($col_{wc}$) < sum($col_{prn}$)
3.      iter = iter + 1
4.      $acc_{binary}$ = ones(1,45)
5.      for i from 1 to 45
6.          if $col_{wc}[i] < col_{prn}[i] + 1$
7.              $col_{wc}[i] = col_{wc}[i] + 1$
8.              $acc_{binary}$ = calcAccBin($col_{wc}$)
9.              $col_{wc}[i] = col_{wc}[i] - 1$
10.         endif
11.     endfor
12.     $i_{min}$ = argmin($acc_{binary}$)
13.     $col_{wc}[i_{min}] = col_{wc}[i_{min}] + 1$
14.     $acc_{final}$[iter] = calcAccAVA($col_{wc}$)
15. endwhile
```

Fig 2.16 worst-care column reduction method

At first, the initialization procedures are similar with the previous greedy search algorithm. In the while loop, a temporal array to record the binary accuracy on 45 binary subtasks is necessary. Since the criterion has been changed to select the minimum (worst) rather than the maximum, the binary accuracy array should be initialized to ones, rather than zeros in the procedures of greedy search. Then calculate the binary accuracy of 45 subtasks by function ‘calcBinAcc()’. Finally, select the boosted classifier with the minimal binary accuracy and grow one column to it, recording the final AVA accuracy change in this step. Similarly, the stop condition is the same as previous.

Through applying the proposed worst-care column reduction algorithm to the model after being pruned, the similar accuracy result on D(11,0) is as following Fig 2.17 and Fig 2.18.

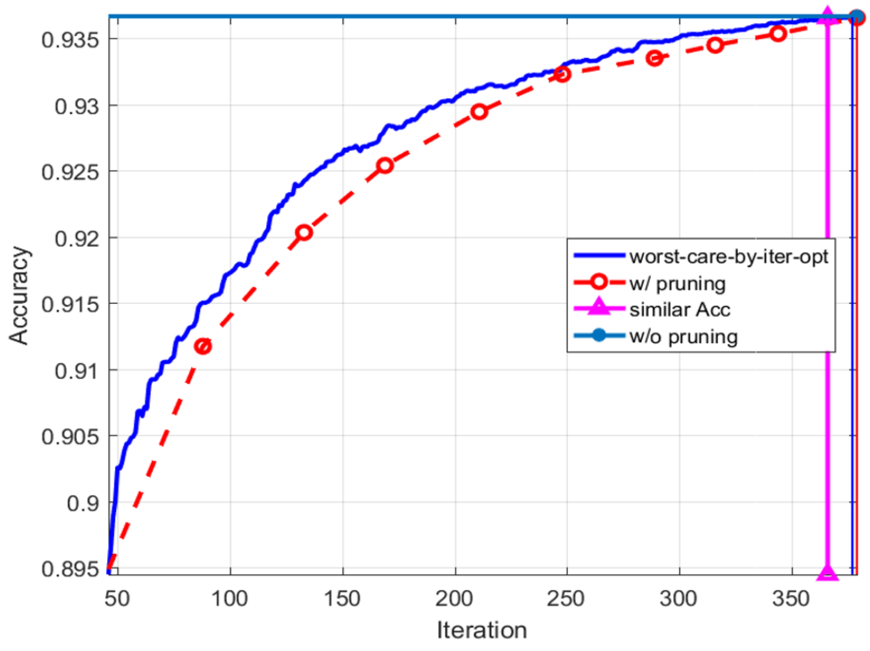


Fig 2.17 Accuracy by iteration of worst-care

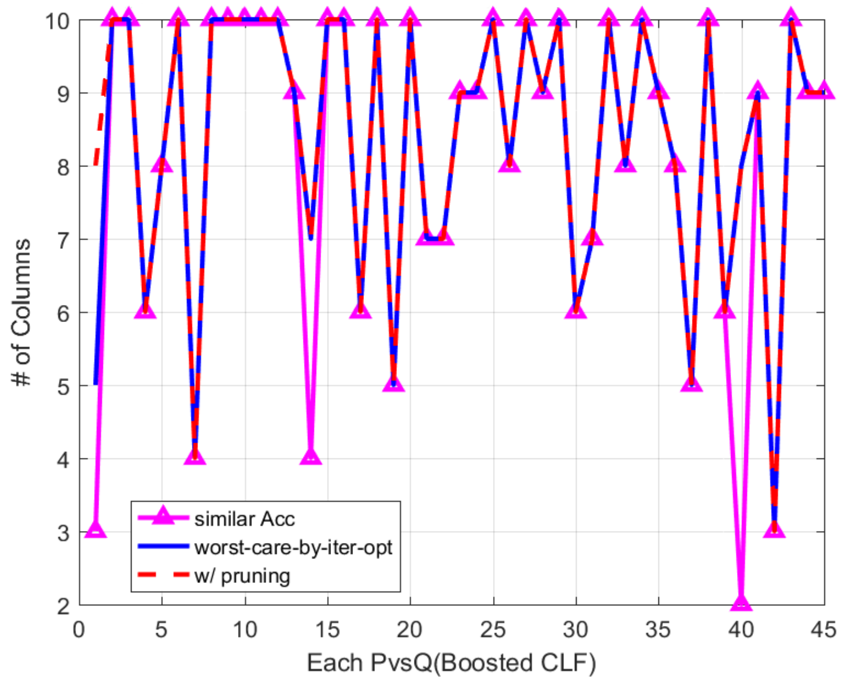


Fig 2.18 Column number for each boosted classifier by worst-care

And shows the result of column number of each boosted classifier by worst-care.

The detailed column growing process by steps of worst-care processing is as following Fig 2.19.

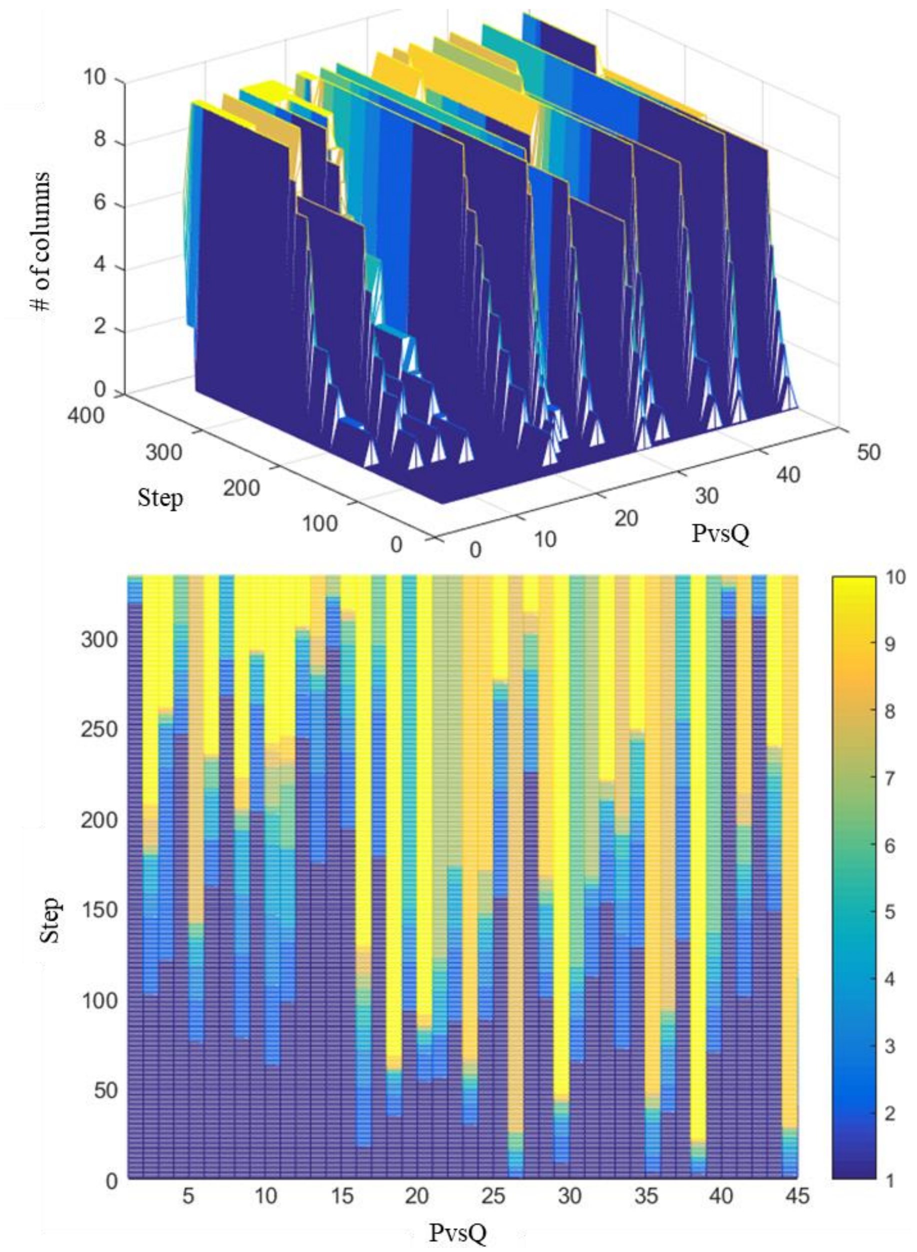


Fig 2.19 Column growing process (worst-care)

Since the searching rule is quite simple, the growing process becomes much more stable compared to the greedy search process. What's more, within the worst-care selecting optimization process, solving global optimal state is avoided, leading to a quite stable process and difficult to be trapped in the local optima like greedy search. In addition, only the binary accuracy history of each boosted classifiers necessary in each step, meaning that the historic training data from the previous training phase can be used again, so that the whole process can be quite fast as about 14 seconds which is about just 5% of greedy search's processing time.

From the accuracy by growing columns in Fig 2.18, it can be found that the worst-care processed model has always a higher accuracy than the pruned model marked as '○' and at its left top position at most of the steps, meaning that with the same number of columns the model processed by worst-care selecting optimization always has better performance or it can reach a similar performance as the pruned model with less columns. Compared to the result of greedy search, the result of worst-care is with less vibration and approximate to but better than the pruned result, showing the stability of the processing.

2.6 Experiments and results

In the previous chapter, the two-step column reduction processing method was proposed, namely the subtask adaptive column pruning as the first step processing and two kinds of heuristic search, greedy search and worst-care selecting optimization algorithms as the second step processing.

In this chapter, the simulation experiments to evaluate the effects of proposed column reduction algorithms and the results will be given.

2.7 Experiment settings

The target of this experiment is to validate the column reduction effect of the proposed algorithms and observe the effect to the performance of the model at the same time.

The detailed diagram for the experiment is as following Fig 2.20.

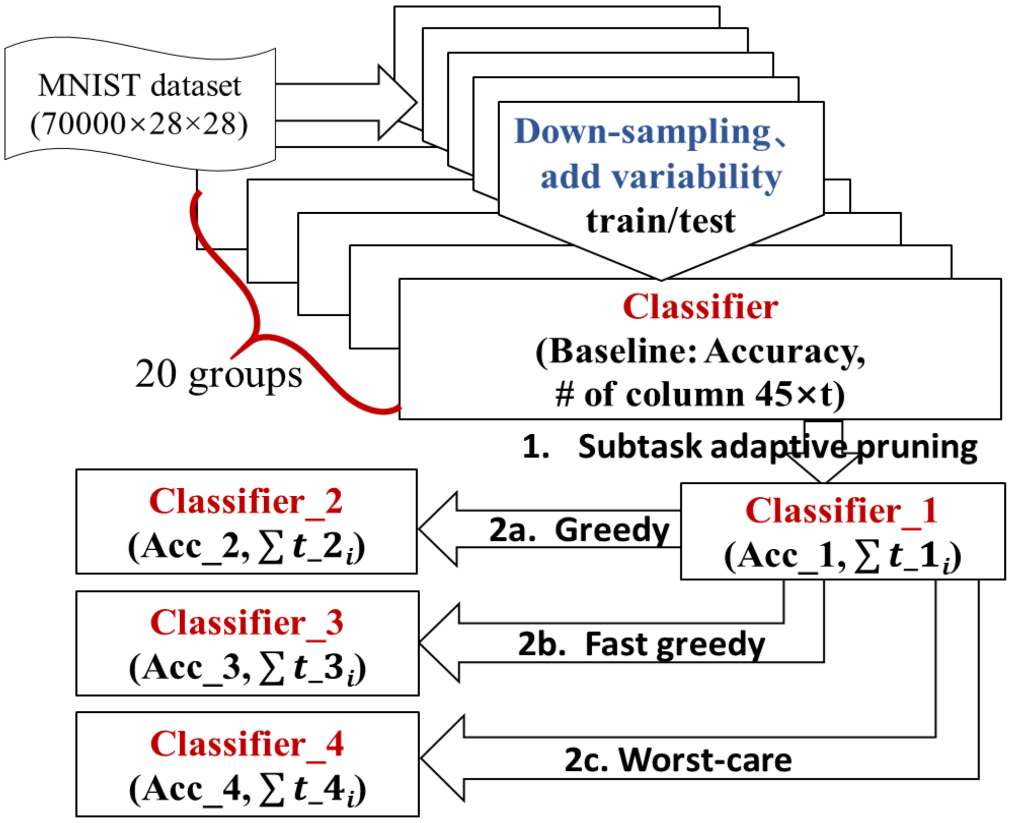


Fig 2.20 Experiment diagram

The object is the models with conventional implementation, which was introduced in the Fig 2.20. The models are trained with several groups of datasets, namely the 20 combinations D(S, ΔV) of down-sampling size S of {11×11, 9×9, 7×7, 5×5} and strengths of variability ΔV of {0, 0.05, 0.1, 0.15, 0.2} (V).

The baseline performance has two metrics, including the final AVA accuracy and the total column number of the model. The baseline accuracy is the AVA accuracy of the model on the test set when its accuracy on train/valid set reaches the maximum. The baseline column number is the total column number as $45\times t$ when the model reaches the selected baseline accuracy controlled by number of boosting iterations t, i.e., the number of weak classifiers included in each boosted classifier.

During the experiment, subtask adaptive pruning is first to apply to the original model in the first step and the responding performance is recorded, which includes AVA accuracy and total column number. After being pruned in first step, the model will be processed further through proposed heuristic algorithms for the second step, also recording the performance of the processed model.

Finally, there will be 5 groups of performance in total to be compared, including baseline, subtask adaptive pruning (1st step), greedy search (2nd step A), fast greedy search (2nd step B) and worst-care selecting (2nd step C).

2.8 Experiment result

2.8.1 Result of column reduction

The detailed column reduction result is as following Fig 2.21.

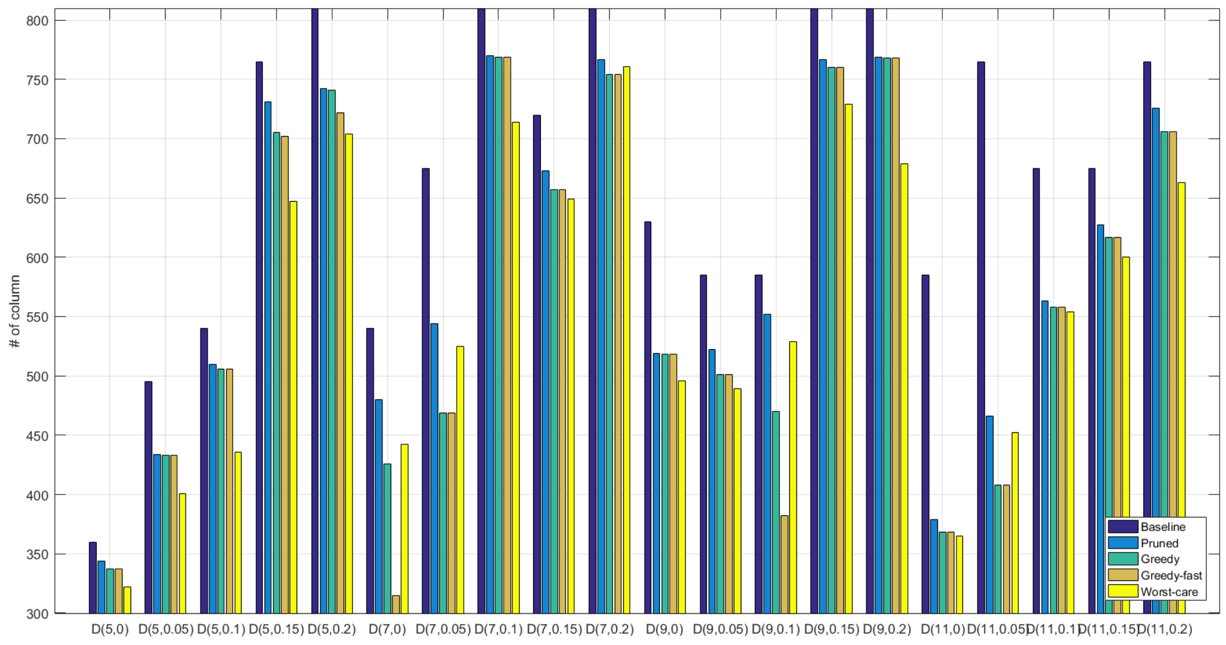


Fig 2.21 Detailed result of column reduction

The x-axis is the different dataset, noted as D(Size, ΔV), as introduced previously. The y-axis is the total column number of the model. The bar plot is totally divided into 20 groups referring to 20 datasets. Each group has 5 bars standing for different performance of model processed by the method by the legend labeled as ‘Baseline’, ‘Pruned’, ‘Greedy’, ‘Greedy-fast’ and ‘Worst-care’ from the left to right.

From the result in Fig 2.21, it can be found that in each group of result the bars on the right are totally with less column number compared to the result of the Baseline, which is the first bar on the left. In addition, the model after being second step is with further reduced column number, meaning that the proposed two-step processing method reaches the effect of reducing columns through the combined processing.

For the representative result, the percentage of column reduction is also given as following Fig 2.22.

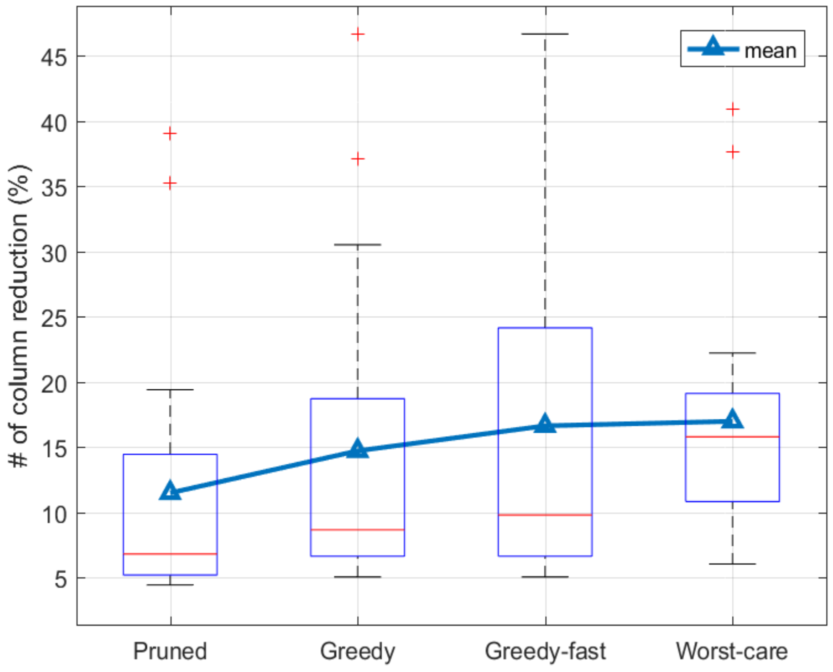


Fig 2.22 Percentage of columns reduction

From Fig 2.22 which is the boxplot of the percentage of reduction among the 20 groups of experiment for proposed 4 processing methods, the effect of reduction can be more obvious. In the first step, through subtask adaptive pruning, the column number can be reduced by 11.50% in average. In the second step, different results are reached through different methods.

Through greedy and fast greedy search, the column number can be reduced further by 3.23% and 5.14% in average, additive to the result of first step. Through worst-care selecting algorithm, the column number can be reduced further by 5.49% in average, which comes to be the best processing method among all in average.

However, it should also be noticed that the upper bounds and range of the results by different methods are also different. The fast greedy search reaches the largest reduction range and the second largest is greedy search, while worst-care selecting is just over the range of subtask adaptive pruning slightly, which is with the minimal range.

It is considered that greedy search type methods are with the target of maximal final accuracy in each search step and can reach quite good results on some of the datasets but also easily trapped into local optimal state, resulting in results not so good on the other datasets. Meanwhile, the weakest boosted classifier first based worst-care selecting method can be quite stable and effective without considering the optimal value of final AVA accuracy, though it cannot reach extremely good results on most of the datasets.

Therefore, it is commended that select the most effective processing method(s) for the actual data in practice, in order to reach the best effect for column reduction.

2.8.2 Change of final accuracy

The detailed result for comparison of final AVA accuracy is as following Fig 2.23

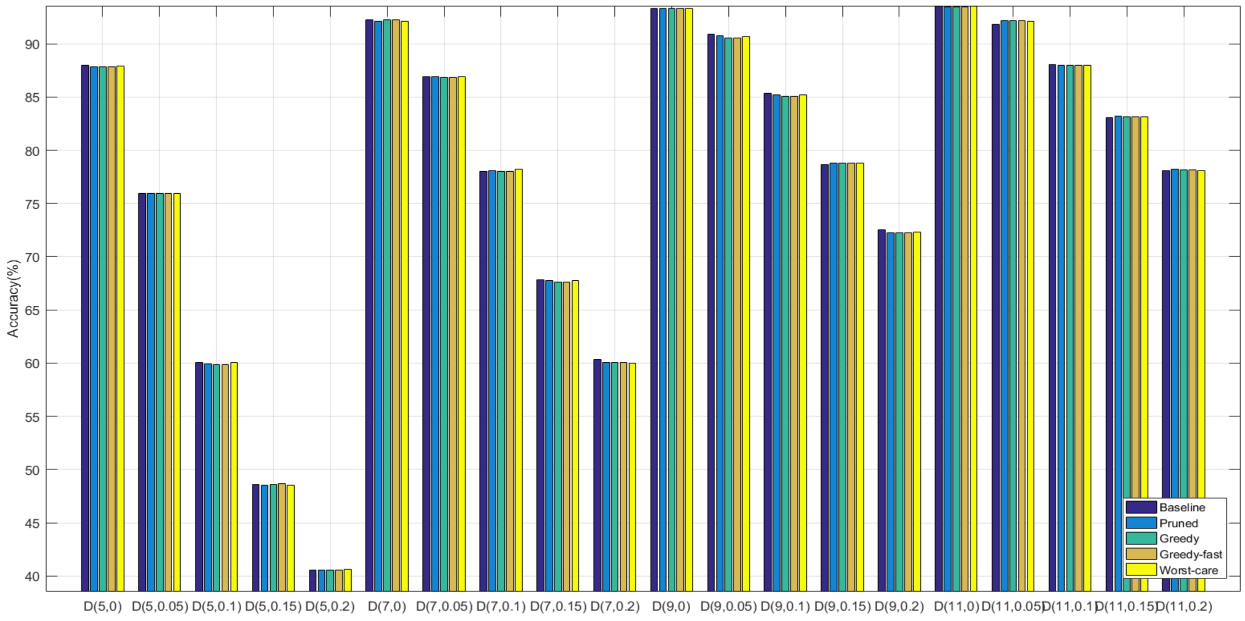


Fig 2.23 Detailed result of final AVA accuracy

With similar x-axis to the detailed result of column reduction, Fig 2.23 shows the result of accuracy as the y-axis. It can be found that the accuracy of the model processed by proposed methods has no obvious change, compared to the baseline performance.

For a more simplified comparing, refer to Fig 2.24.

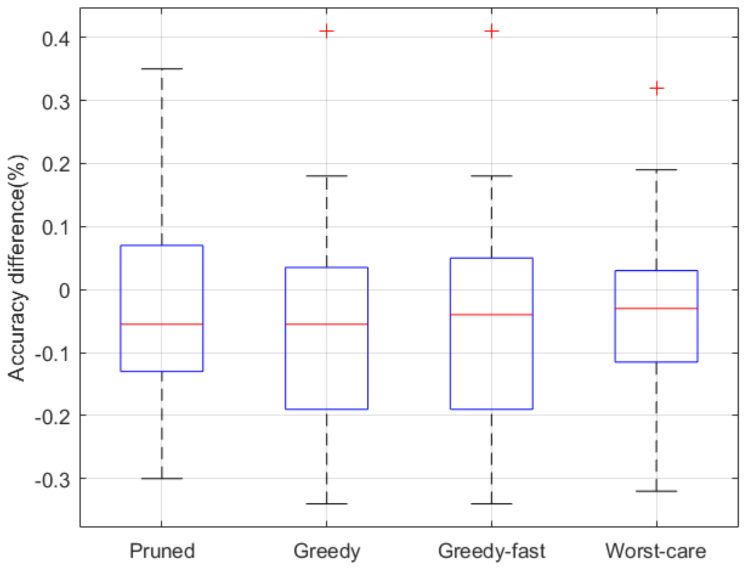


Fig 2.24 Accuracy difference

From Fig 2.24, it can be found that most of models after being column reduction processing get slight accuracy loss. The accuracy loss of model processed by subtask adaptive pruning and worst-care selecting is about 0.28% in average and by greedy and fast greedy search is 0.575% and 0.525% in average, respectively. Thus, it is recommended to choose the proper processing methods according to the minimal demand of accuracy in practical application.

2.9 Conclusion

Starting with the redundant columns existed in the conventional implemented model, this chapter explored a two-step column reduction processing method for the in-memory machine learning classifier from the point of heuristic searching algorithm.

After applying subtask adaptive column pruning to the original model as the first step, based on maximal accuracy gains first greedy search and weakest boosted classifier first selecting ('worst-care'), this research proposed 2 kinds of column reduction algorithms in the second step, achieving the target of reducing the columns of model with minimal performance loss.

This work mainly researched the SRAM based in-memory machine learning classifier and the responding column reduction algorithms. With the conventional implementation of in-memory classifier reaching great energy saving and available accuracy, hardware errors and nonlinear noises existed in the bit cells as well as the time-dependent variability makes the model need more memory arrays to achieve available performance, which calls the demand on the methods to reduce the impact of variability as well as the number of memory column arrays.

The conventional implementation for in-memory machine learning classifier was researched and simulated with the MNIST dataset, where 1-Bit Constraint-resolution-regression-based error adaptive classifier boosting algorithm is employed to implement in 6T SRAM. The effect of the classifier was verified.

In order to achieve a further level of energy saving, a two-step processing method was proposed. First, the subtask adaptive pruning process was proposed based on the different difficulty between the subtasks. Then, from the view of heuristic search, maximal accuracy gains first criterion based greedy search and fast greedy search were proposed. Based on weakest boosted classifier first criterion, the worst-care selecting algorithm was proposed. With comparison of the accuracy by proposed processing, some typical results of experiment verified the effect of column reduction by proposed algorithms.

Chapter 3 Improving stability and accuracy of low precision neural network

Abstract

Binary neural networks (BNNs) have drawn much attention because of the most promising techniques to meet the desired memory footprint and inference speed requirements. However, they still suffer from the severe intrinsic instability of the error convergence, resulting in increase in prediction error and its standard deviation, which is mostly caused by the inherently poor representation with only two possible values of -1 and +1. In this work, we have proposed a cost-aware layer-wise ensemble method to address the above issue without incurring any excessive costs, which is characterized by 1) layer-wise bagging, 2) cost-aware layer selection for the bagging. One of the experimental results has shown that the proposed method reduces the error and its standard deviation by 15% and 54% on CIFAR-10, respectively, compared to the BNN serving as a baseline.

This chapter demonstrated and discussed such error reduction and stability performance with high versatility based on the comparison results under the various cases of combinations of the network base model with the proposed and the state-of-the-art prior techniques while changing the network sizes and datasets of CIFAR-10, SVHN, and MNIST for the evaluation.

3.1 Introduction

Nowadays, deep neural networks (DNNs) are being widely used for many applications such as computer vision, speech recognition, natural language processing, neural machine translation, etc., because of powerful abilities of feature extraction and knowledge representation. However, energy-hungry devices like high-performance GPU are needed to train the DNNs with a large number of parameters whose data type is 32-bit floating-point precision (float32). Unfortunately, this has become a hurdle to deploy the deep neural networks to mobile devices and low-cost embedded systems because such systems cannot afford to exploit the power and cost hungry GPU for their applications. It could be a critical bottleneck to deploy the AI with DNN into the cost and power-sensitive fields like the internet of things (IoT) until their power and cost requirements have been much alleviated by taking a different approach to the DNNs.

Recently, to address the above issue, the network quantization techniques were proposed to meet the requirements for saving energy and memory cost that is critical in mobile devices and embedded systems. The quantization technique allows relaxing the precision requirements of the network parameters from the float32 to lower integer precision like 8-bit, 4-bit, 2-bit, and even binary (1-bit) while suppressing the accuracy degradation to an allowable level. It can contribute to reducing the required computing resources and the memory footprint to complete the training within a practical time. Courbariaux et al.[60] proposed a binary neural network

(BNN) which uses the binary precision (-1/+1) for expressing the values of weight and activation.

The outline of the training procedure for the BNN model is shown in Fig. 3.1.

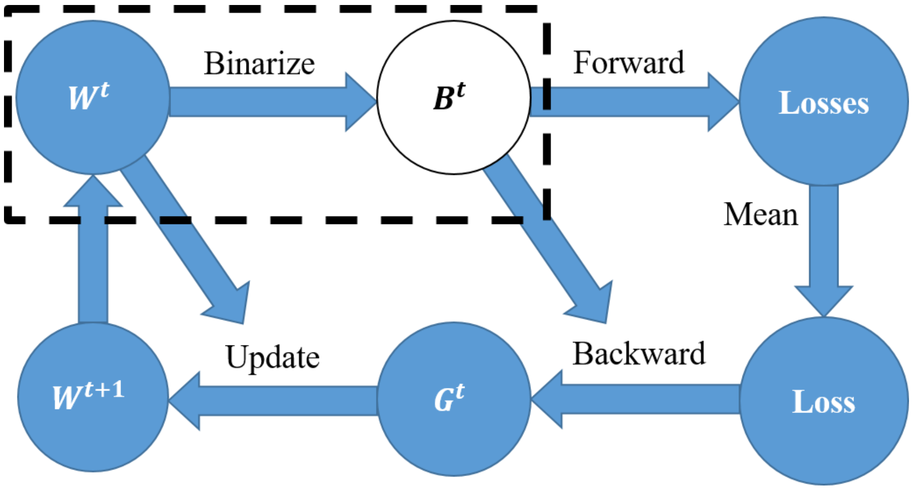


Fig. 3.1 The training procedure for the BNN model

In whole process, only $\boldsymbol{B^t}$ is expressed by a binary precision, and others ($\boldsymbol{W^t}$, $\boldsymbol{G^t}$, Losses, Loss) use the float32. The $\boldsymbol{B^t}$ is binarized data of the $\boldsymbol{W^t}$. The current $\boldsymbol{W^t}$ and gradient $\boldsymbol{G^t}$ are used to generate the next weight $\boldsymbol{W^{t+1}}$. The gradient $\boldsymbol{G^t}$ is given based on the mean value Loss and the current $\boldsymbol{B^t}$. The whole process is repeated to reduce the value of the Loss. It is noteworthy that the conventional floating-point operations in the forward propagation can be replaced by bit-wise operations thanks to using binarized parameters.

For example, the dot product of two binary vectors can be computed by XNOR logic as follows:

$$\mathbf{a} \cdot \mathbf{b} = \mathit{bitcount}\big(\mathrm{XNOR}(\mathbf{a}, \mathbf{b})\big)$$

where the *bitcount* is a function that counts the number of 1 in a binary vector. Since the bitwise operations are basically carried out by only XNOR logic instead of the arithmetic logic unit (ALU), it consumes much less energy than the full-precision neural networks. Thanks to the binarization of weights and fewer multiplications, the BNN model achieved about 32 times memory savings and 7 times run-time speed-ups[60].

Although the BNN achieved a big success in the run-time speedups in the forward stage while reducing the memory and the energy consumption, the error and its standard deviation of the predictions are increased as unacceptable side effects, which is mainly caused by the intrinsic risk factor of poor representation ability with (-1/+1). Thus, the various related works, including the state-of-the-art paper BENN[61] that used ensemble techniques to address this issue, were proposed. We will discuss the techniques of those papers in the following Section.

We will newly propose the cost-aware layer-wise bagging technique in this work. To the best of our knowledge, this is the first work to propose and discuss the following things.

1. Cost-aware layer-wise bagging method to address the cost issue faced by the BENN.

2. Smart layer selection for deploying the layer-wise bagging, which is characterized by cost-aware strategy for using the layer-wise bagging. Thanks to this tactic, the number of additional parameters is suppressed within several percent, which is much fewer than the case for the BENN (several hundred percent).

3. Breakdown analysis of the uniqueness of the binary filter patterns generated by the training which is used for evaluating the expressive efficiency of the filter (a better representation ability than other related works will be demonstrated based on its comparison results between the proposed layer-wise bagging and the conventional techniques).

In addition to the above, this work will discuss the advantages over other state-of-the-art counterparts with higher versatility by comparing the experimental results under the various cases of combinations of the network base models with the proposed and the most recent techniques while changing the network sizes and datasets for the evaluations.

The rest of this chapter is organized as follows. Section 2 reviews quantized neural networks and ensemble methods, followed by discussing the instability issue of BNN in Sec. 3. The proposed method is introduced and compared with conventional techniques in Sec. 4, and the concerns and its handling for the additional cost are explained in Sec. 5. We demonstrate and discuss the experiment results in Sec. 6, followed by the conclusion of this chapter in Sec. 7.

3.2 Related works

3.2.1 Quantized neural networks

There are many works focusing on network quantization techniques in recent years. Rastegari et al.[62] proposed XNOR-Net which is the modified version of the BNN and achieved an improved result on the ImageNet dataset. But it still uses the full precision (float32) in the first and last layers of the network. Miyashita et al.[63] proposed a method to quantize the activations/weights in logarithmic representation. They quantized the gradients with 6-bit precision without any significant accuracy loss on the CIFAR-10 dataset. Zhou et al.[64] proposed DoReFa-net which also employed logarithmic representations for the weight (1-bit), activation (2-bit) and the gradient (6-bit). It achieved a 46.1% top-1 classification accuracy on the ImageNet dataset. Merolla et al.[65] applied weight binarization and several different kinds of noises to the DNN and demonstrated the robustness of the DNN networks to the various distortions. Bulat et al.[66] proposed to fuse the activation and weight scaling factors into a single one that is learned discriminatively via backpropagation. They also explored different ways of constructing the scaling factors and demonstrated the accuracy gain of up to 6% on the ImageNet dataset using ResNet-18. Simons et al.[67] gave a tutorial of the general BNN methodology and reviewed various contributions, implementations, and applications of the BNN models.

To deal with the severe accuracy degradation of BNNs, Lin et al.[68] proposed ABC-Net which employs the linear combinations of the multiple binary weight bases to approximate the original high-precision weights and different activation functions by using the shift parameter. It reduces the loss of input information, achieving a 65.0% top-1 accuracy on the ImageNet dataset which is competitive performance compared to the full precision counterpart.

In the following sections, we will discuss how the ensemble learning method can contribute to further error reduction compared with the ABC-Net.

3.2.2 Ensemble methods

One of the well-known techniques for prediction accuracy improvement is ensemble learning which uses multiple weak classifiers and combines their output decisions. The two well-known ensemble learning techniques are boosting and bootstrap aggregating which are abbreviated as bagging[69][70].

The boosting in fact refers to a family of algorithms that consists of multiple weak learners and strong learners. Each output of the weak learners is multiplied with different weight and all outputs are aggregated to the final strong learner. Using different weighting rules for training samples and hypothesis has led to many variations of boosting algorithm such as AdaBoost and LogitBoost[70] The bagging is trained from independent and identically distributed (IID) training samples by averaging or voting the outputs of weak classifiers. In general, if the goal is to get **N** weak classifiers, **N** training phases are necessary, while in each training phase every single weak classifier is trained with only approximate 63.2% of samples randomly sampled with replacement from the whole training set.

Recently, these ensemble techniques have come to be applied to the deep neural networks instead of the conventional decision tree models. As a noteworthy work, the Binary ensemble neural network (BENN) was proposed to solve the accuracy degradation which is predominantly caused by intrinsic instability issue of the training of the BNN. However, since the BENN applies the ensemble method to the entire network, an intolerable additional cost of several times larger numbers of parameters for representing the whole network model are unfortunately incurred, which is not an acceptable drawback for the low power and low-cost systems.

In order to solve this issue, we newly proposed the cost-aware layer-wise bagging technique. This is characterized by the two things of 1) layer-wise bagging and 2) cost-aware selection of the applied layers for bagging (only to the first and last layers). Thanks to the cost-aware layer-wise bagging, the increase of the number of additional parameters for the layer-wise bagging is to suppress to less than 2%, which is much smaller than the case for the BENN. We will explain this matter further in detail based on the actual example data in Sec. 5.

3.3 Instability of Binary Neural Network

One of the critical problems for the BNN training process is instability which affects the training time as well as the error convergence. The comparisons of validation error convergence curves and its standard deviation between the BNN and the full precision (FP) network for the CIFAR-10 image classification task are shown in Fig. 3.2a and b, respectively.

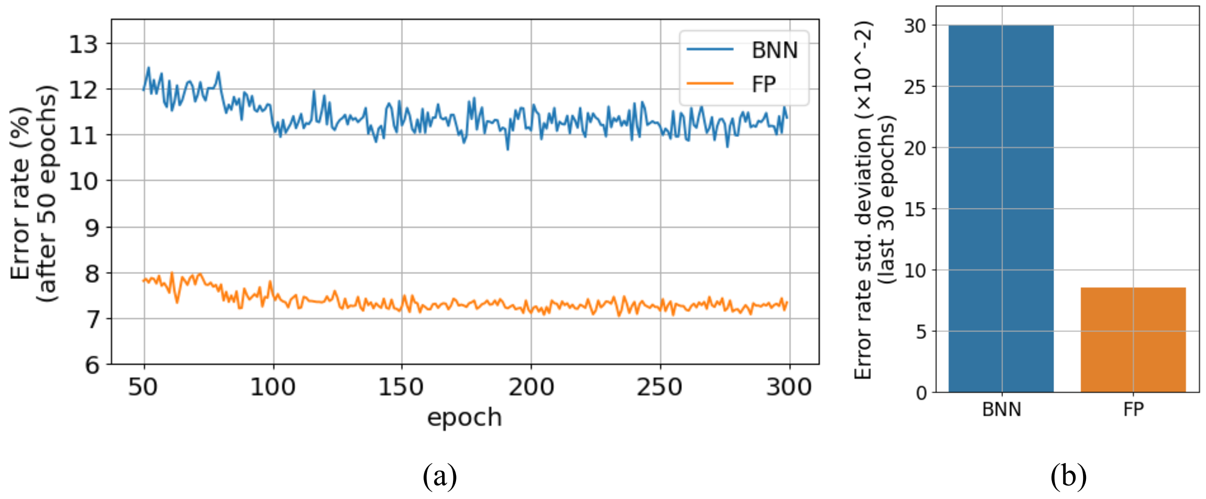


Fig. 3.2 Comparisons of (a) error convergence curves and (b) its standard deviation between the BNN and full-precision (float32) network on the CIFAR-10 dataset.

The error convergence curves from epoch 50 and its standard deviation in the last 30 epochs are shown in Fig. 3.2a and b, respectively. It can be seen that the error convergence curve for the BNN is more fluctuated and its standard deviation is much larger than the FP network.

There are mainly two reasons for having a higher variation of the prediction error rate in the training process of BNN.

1) Significantly reduced numeric precision of the back-propagation due to the quantized weights, which makes the training accuracy worsen.

2) The increased errors of the quantized activations calculated from the propagated gradients compared to the values for the FP network.

3.4 Layer-wise ensemble

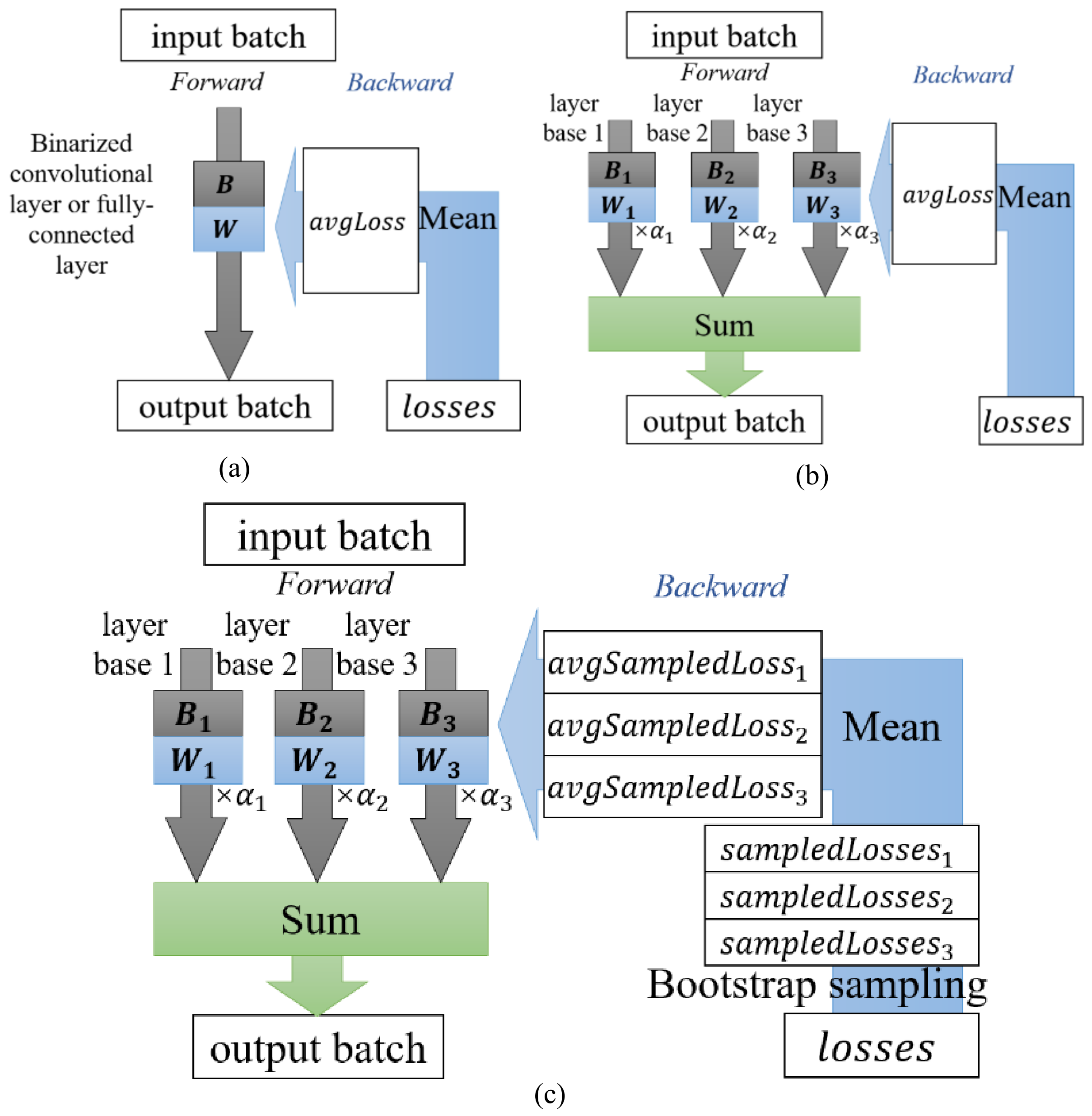


Fig. 3.3 Comparisons of the structures among the three methods (a) single binarized layer, (b) linear combination, and (c) proposed Layer Bagging.

The Fig. 3.3 shows the structure comparisons among a single binarized layer, linear combination, and proposed layer-wise ensemble method, Layer Bagging.

The structure of a single binarized layer used in the BNN is shown in Fig. 3.3a, which is used for a binarized convolutional layer or binarized fully-connected layer. In the forward propagation, the input batch and binarized weight B generate the output batch. In the backward propagation, the full-precision weight W is updated based on the full-precision gradient value of the *avgLoss* that is the mean value of the *losses* of all the samples in the minibatch. The binarized weight B is given by binarizing the full precision W as shown in the detailed procedures in Fig. 3.1.

The set of the weight $\{B,W\}$ for the single binarized layer is changed to $\{B_1,W_1\},\{B_2,W_2\},\{B_3,W_3\}$ for the multiple binarized layer bases, as shown in Fig. 3.3. In the forward propagation, the outputs of multiple layer bases are generated in the multiple layer bases for the same input batch, which are weighted with the multiple trainable full-precision scaling factors α_1, α_2, α_3 and then summed up. In the backward propagation, it has the same process as the case for a single binarized layer.

The Fig. 3.3c shows the structure of the proposed Layer Bagging method. In the backward propagation, we insert a bootstrap sampling process to generate independent $sampledLosses_1$, $sampledLosses_2$, $sampledLosses_3$ of different layer bases and use the mean values ($avgSampledLosses_1$, $avgSampledLosses_2$, $avgSampledLosses_3$) to generate the gradients independently which are then used to update corresponding W and α. In the forward propagation, it has the same process as the linear combination method.

In the following subsections, we will explain the linear combination method more in detail which is proposed in the ABC-Net followed by elaborating our proposed Layer Bagging method so that the differences between the two can be made clear.

3.4.1 Linear combination

The following shows the concept of the training procedure for the linear combination method. The outline of the process is similar to the conventional training procedure for the BNN. The key difference from the conventional BNN is to employ linear combination method in the multiple sets of parameters $\{W_1^t,\dots,W_n^t\},\{W_1^{t+1},\dots,W_n^{t+1}\},\{B_1^t,\dots,B_n^t\},\{G_1^t,\dots,G_n^t\}$ for the corresponding multiple binarized layer bases, as shown in Fig. 3.4.

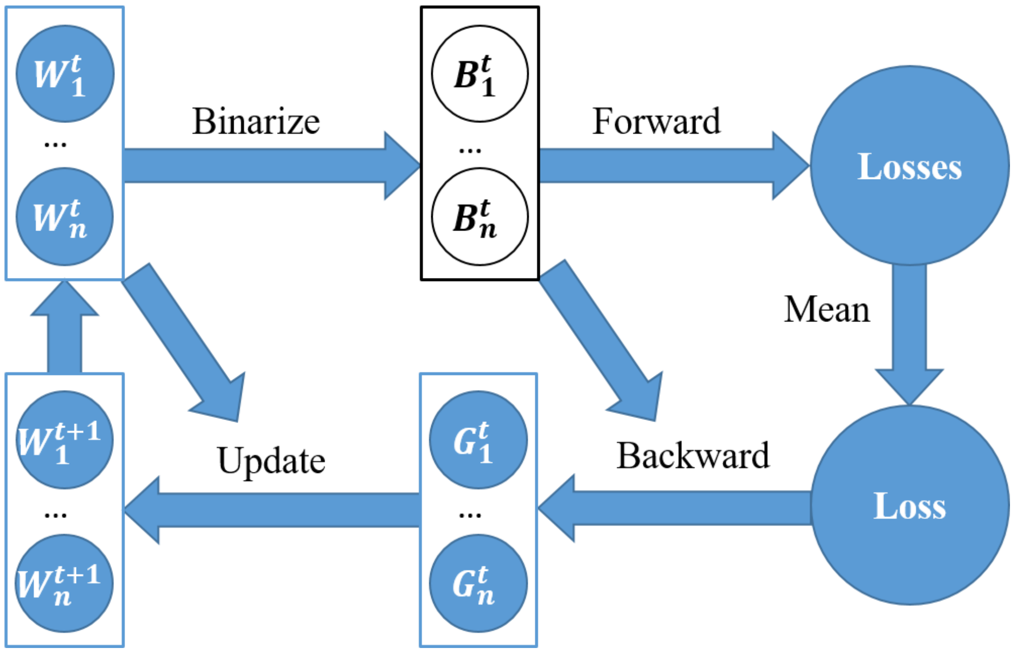


Fig. 3.4 The training procedure for the linear combination method.

The key features of the linear combination are as follows:

1) Multiple binarized layers (n layer bases in total) are randomly initialized which have different initial values but share the same parameter settings (e.g., input channels, output channels, kernel size, and stride) for convolutional layer, or the number of input features, output channels for the fully-connected layer.

2) Since the multiple layer bases are sharing the same layer setting of the initialization among them, the same input comes to be used in each layer of the same linear combination group. Independent training of the weights for the multiple layer bases generates more variable binary representations than the case for a single binarized layer.

3) The accuracy of the final output becomes better and more stable, because the outputs from multiple layer bases are aggregated and averaged.

In the forward part of Algorithm Alg. 3.1, n, B and X denote the number of layer bases, the binary weights of the convolutional or fully-connected layer, and the input batch, respectively. H_i and the final output h are given by X and weight B_i of each layer base and binarizing the aggregation of the product of all H_i and full-precision scaling factor α_i,

```
// Forward of linear combination module
**Input**: $B$; $\alpha$; $X$; $n$;
**Output**: $h$;
for $i \leftarrow 1$ to $n$ do
    $H_i \leftarrow Forward(B_i, X)$
end
$h \leftarrow Binarize(\sum_{i=1}^{n} H_i \cdot \alpha_i)$
// Backward of linear combination module
**Input**: $W$; $B$; $\alpha$; $losses$; $n$;
**Output**: $W$; $B$; $\alpha$;
$avgLoss \leftarrow mean(losses)$
for $i \leftarrow 1$ to $n$ do
    $G_{W_i^t}, G_{\alpha_i^t} \leftarrow Backward(avgLoss, B_i^t, \alpha_i^t)$
    $W_i^{t+1} \leftarrow Update\left(W_i^t, G_{W_i^t}\right)$
    $B_i^{t+1} \leftarrow Binarize(W_i^{t+1})$
    $\alpha_i^{t+1} \leftarrow Update\left(\alpha_i^t, G_{\alpha_i^t}\right)$
end
```

Alg. 3.1 Linear combination pseudo code

respectively.

In the backward part, W represents the full precision counterpart of the corresponding binary weights. $losses$ is a list of loss of each sample in an input batch. $G_{W_i^t}, G_{\alpha_i^t}$ denote the full precision gradient for the corresponding weight and the activation calculated by $avgLoss$ (mean value of $losses$), binary weight B and α at current iteration t. W, B and α are updated with current variables W_i^t, B_i^t, α_i^t and gradients for $G_{W_i^t}$ and $G_{\alpha_i^t}$.

3.4.2 Layer bagging

To reduce the error rate and its standard deviation more than the linear combination (LC) method, we propose a layer-wise ensemble method called "Layer Bagging". The 'LC' method has an obvious drawback to the cost of the network because many identical convolutional filters are unfortunately generated for the different binary layer bases. The identical filters (whose pattern with +1/-1 is completely the same) of the total accounts for over 30% based on our experimental results. As a result, it leads to a redundant consumption of the filters, resulting in a decrease in the usage for the model representation.

We suppose the key reason why the 'LC' is prone to generate the same patterns in the filter is based on the combination of the two constraints: 1) limited combinations of +1/-1 due to the binarization and 2) using the same input to the different layer bases. Thus, in order to address this fundamental issue, the bagging algorithm has been introduced in this work. This allows each weak learner for the bagging to have more independent variables which can contribute to reducing the error while suppressing its standard deviation. The Fig. 3.5 shows the training procedure for the proposed Layer Bagging method.

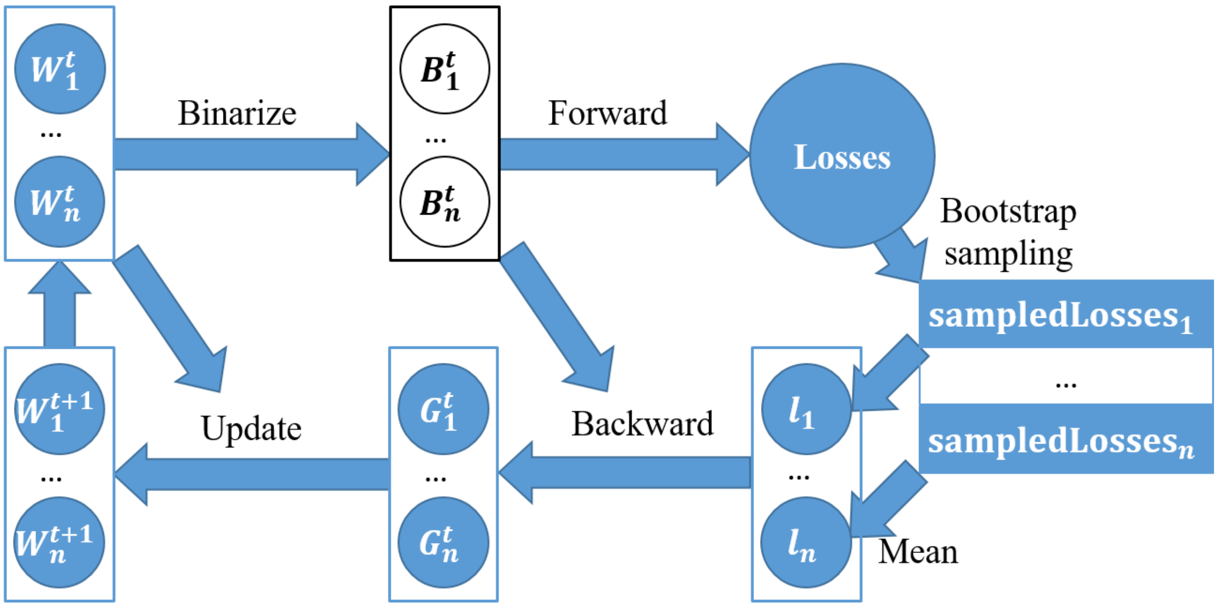


Fig. 3.5 The training procedure for the Layer Bagging method.

Based on the 'LC' method shown in Fig, we mainly insert a bootstrap sampling part into the procedure. The bootstrap sampling is applied on the sample-wise $losses$ multiple times to generate the independent subsets $sampledLosses_1, \dots, sampledLosses_n$. And the corresponding mean values $l_1, \dots, l_n$ are used to generate $G_1^t, \dots, G_n^t$.

The pseudo code of the proposed Layer Bagging method is shown in Algorithm Alg. 3.2. In the backward part, sr (we use $sr = 0.632$ in this work), representing the sample ratio of the losses of the input samples, is used in the function $BootstrapSampling$ to sample $losses$ randomly with replacement to get $sampledLosses$. The mean value $avgSampledLoss$ is used to update the weight and α. The forward part is the same as the forward part of the linear combination.

```
// Forward of layer bagging module
**Input**: $B$; $\alpha$; $X$; $n$;
**Output**: $h$;
for $i \leftarrow 1$ to $n$ do
    $H_i \leftarrow Forward(B_i, X)$
end
$h \leftarrow Binarize(\sum_{i=1}^{n} H_i \cdot \alpha_i)$
// Backward of layer bagging module
**Input**: $W$; $B$; $\alpha$; $losses$; $n$;
**Output**: $W$; $B$; $\alpha$;
for $i \leftarrow 1$ to $n$ do
    $sampledLosses \leftarrow BootstrapSampling(losses, sr)$
    $avgSampledLosses \leftarrow mean(sampledLosses)$
    $G_{W_i^t}, G_{\alpha_i^t} \leftarrow Backward(avgSampledLosses, B_i^t, \alpha_i^t)$
    $W_i^{t+1} \leftarrow Update\left(W_i^t, G_{W_i^t}\right)$
    $B_i^{t+1} \leftarrow Binarize(W_i^{t+1})$
    $\alpha_i^{t+1} \leftarrow Update\left(\alpha_i^t, G_{\alpha_i^t}\right)$
end
```

Alg. 3.2 Layer Bagging pseudo code

Here we also propose a general backward propagation process for the whole model to employ the proposed Layer Bagging method in Algorithm Alg. 3.3. There are several processes in Algorithm. $FreezeAllModules$ is used to disable the parameter updating of the whole model. $UnfreezeBaggingModule(model, i)$ and $FreezeBaggingModule(model, i)$ are used to enable and disable the parameter updating of i-th layer base of the Layer Bagging modules in the model, respectively. Such two processes are used to preserve independent parameter updating with different $avgSampledLosses$ in different layer bases of the Layer Bagging modules. $UnfreezeNonBaggingModules$ is used to enable the parameter updating of the modules other than the Layer Bagging module in the model.

```
**Input**: *model*; *losses*; *n*; *sr*;
**Output**: *model*;
$model \leftarrow FreezeAllModules(model)$
for $i \leftarrow 1$ to $n$ do
    $sampledLosses \leftarrow BootstrapSampling(losses, sr)$
    $avgSampledLosses \leftarrow mean(sampledLosses)$
    $model \leftarrow UnfreezeBaggingModule(model, i)$
    $model \leftarrow Backward(model, avgSampledLoss)$
    $model \leftarrow FreezeBaggingModule(model, i)$
end
$model \leftarrow UnfreezeNonBaggingModules(model)$
$avgLoss \leftarrow mean(losses)$
$model \leftarrow Backward(avgLoss)$
```

Alg. 3.3 General backward process of training with Layer Bagging

3.5 Additional Cost Concern

To apply the ensemble method to the entire network at the cost of the required much more training time and memory footprint than a single network is not a practical way for the low power and low-cost systems. For example, the conventionally proposed BENN (which relying on whole network level 5× ensembles) brings about an intolerable increase (by 400%) in the number of parameters. Thus, we newly proposed the Layer Bagging technique to avoid such kind of unbearable side effect, which actually can suppress the overhead (increased number of parameters) to less than 1.7% for a VGG-like CNN model used in this work whose number of parameters is shown in following table.

Table 3 Number of parameters of a VGG-like CNN model used in this work.

Layer block	# of parameters
conv3x3-128 (input)	(3*3*3)*128 +(3*3*128)*128=150,912
conv3x3-128	(3*3*128)*256 +(3*3*256)*256=884,736
conv3x3-128	(3*3*256)*512 +(3*3*512)*512=3,538,944
fc1	8*8*512*1024=33,554,432
fc2	1024*1024=1,048,576
fc3 (output)	1024*10=10,240
Total	39,187,840

Since the proposed Layer Bagging technique is applied only to the first and the last layers unlike the BENN (which applies to the whole layers), the overhead can be negligibly small which is the prerequisite requirement for a low power and low-cost system. In addition to the above, we will explain the reason why the first and the last layers were chosen for applying the bagging technique as follows: 1) the first and last layers have very few parameter numbers than others (see Table 3), 2) more sensitive to the accuracy degradation as the related work mentioned. According to the related work, many BNN works use the higher precision only for the first and last layers than other layers to avoid a significant accuracy degradation. Based on such kind of background, we finally decided to employ the proposed Layer Bagging method only to the first and last layers instead of using a higher precision like float32. To the best of our knowledge, this is the first work to apply the layer bagging while almost avoiding the additional network overhead.

3.6 Experiments and results

We conducted several image classification experiments mainly based on the CIFAR-10 dataset, which is a common benchmark dataset used for the image classification task. The dataset consists of 60k (train/test: 50k/10k) 32 by 32 color images in 10 classes with 6000 images in each class. The program for the experiments is implemented in PyTorch and the program is executed on Ubuntu 16.04 with the NVidia GeForce GTX 1080 Ti graphic card.

We also conducted the experiments based on other datasets of SVHN and MNIST and different base model of XNOR-Net++, so that we can evaluate the advantages from the proposed technique with versatility.

To make a fair comparison, the similar setups and model architecture compared to the original BNN paper are used in this experiment as much as possible.

3.6.1 Experiment settings

The preprocessing part of the experiment simply used random crop, random horizontal flip, and normalizing with image net statistics. The basic binary CNN model architecture used in this paper is shown in Fig. 3.6.

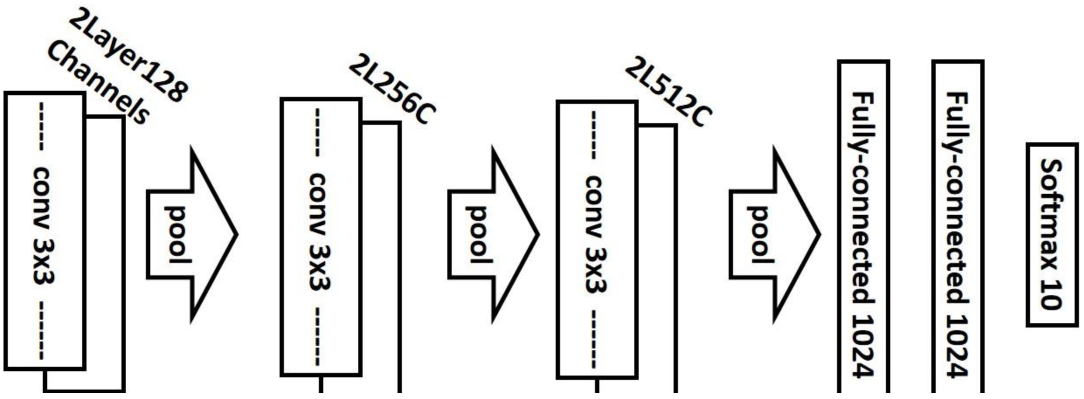


Fig. 3.6 Architecture of the basic CNN model used in this work.

In Fig. 3.6, "conv3×3" represents a 3 by 3 convolution layer with batch normalization layer and ReLU activation and "pool" is short for 2 by 2 max-pooling. "2L256C" is short for a 2-layer block of 256-channel convolutional in each layer.

The CNN model architecture is mainly composed of three convolution layer blocks (each block consists of two layers, i.e., 2-2-2) followed by three fully-connected layers with the output feature numbers of 1024-1024-10 for the classification. To compare the errors among the networks in various sizes, whose number of layers is different, between the proposed and the conventional methods.

we used four different combinations of the number of convolution layer blocks and the number of fully-connected layers to '2-2-2-3' (standard), '1-2-2-3', '1-1-2-3', and '1-1-1-3', while keeping other parts unchanged.

In order to discuss the dependencies on some conditions, we conducted the experiments while changing the conditions as shown in following table. The conditions can be varied by changing the combination of 1) using the method from {'BNN', 'LC-3', 'LC-5', 'Bag-3', 'Bag-5'}, 2) choosing to use one technique from the linear combination and Layer Bagging or not using each technique at all. Therefore, there are results for the 4*5=20 combinations in total.

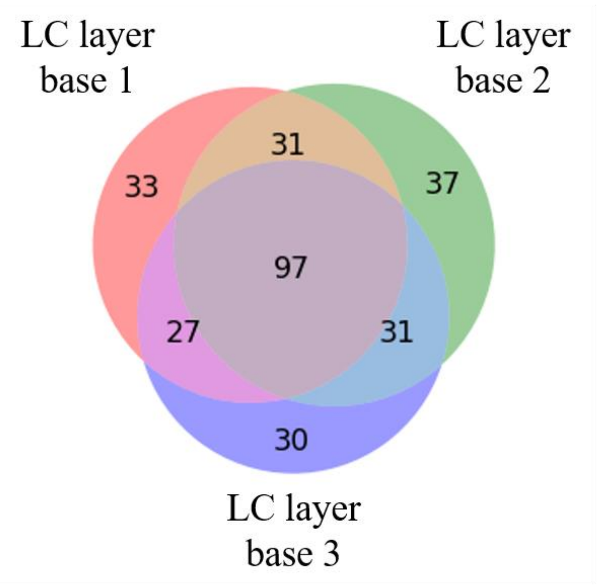
Table 4 Definitions for various methods (BNN, LC-3, Bag-3, LC-5, and Bag-5).

Methods	# of layer basis	If use LC	If use Bag
BNN	1	N/A	N/A
LC-3	3	True	False
Bag-3	3	False	True
LC-5	5	True	False
Bag-5	5	False	True

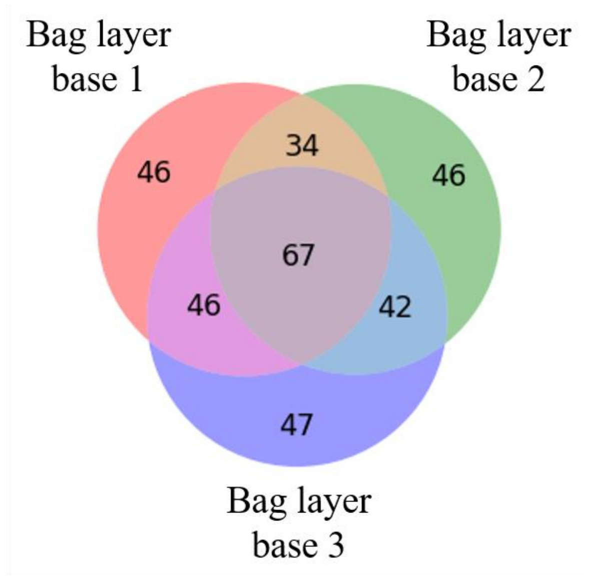
3.6.2 Comparisons with the-state-of-the-art on various conditions

To highlight the difference in how many identical patterns are being used in the 3×3 convolutional filters of all different layer bases between the two cases of the linear combination and Layer Bagging, the Venn diagrams are shown in Fig. 3.7.

The comparisons of percentage of each category for "identical", "semi-identical", and "unique" filters in different layer bases between the 'LC-3' and 'Bag-3' for the '1-1-1-3' network is shown in following Table 5.



(a)



(b)

Fig. 3.7 The Venn diagrams of the 3×3 convolutional filters in different layer bases of (a) linear combination and (b) Layer Bagging from '1-1-1-3' network.

Table 5 Comparisons of percentage of each category for "identical", "semi- identical", and "unique" filters in different layer bases between the LC-3 and Bag-3 for the '1-1-1-3' network.

Methods	Identical	Semi-identical (2/3 are identical)	Unique
Linear Combination	34%	31%	35%
Layer Bagging (Ours)	20%	37%	42%

As for some short-term in the figures, '1-1-1-3' denotes that the architecture of the model is composed of the convolutional layer blocks with layer number of 1,1,1 followed by 3 fully-connected layers, 'LC-3' and 'Bag-3' represent the results for the linear combination with the number of bases (=3) and the proposed Layer Bagging method, respectively.

We took the case of '1-1-1-3' network as an example to visualize the number in the Venn diagram. It is found that the identical filters account for over 34% (97/286) of all for the 'LC-3' while its number for the 'Bag-3' is suppressed to 20% (67/328).

The distributions of the percentage of the identical convolutional filters of all four different model architectures are compared between 'LC-3' and 'Bag-3', as shown in Fig. 3.8. It is found that the number of the identical filters is significantly decreased by employing the proposed Layer Bagging method.

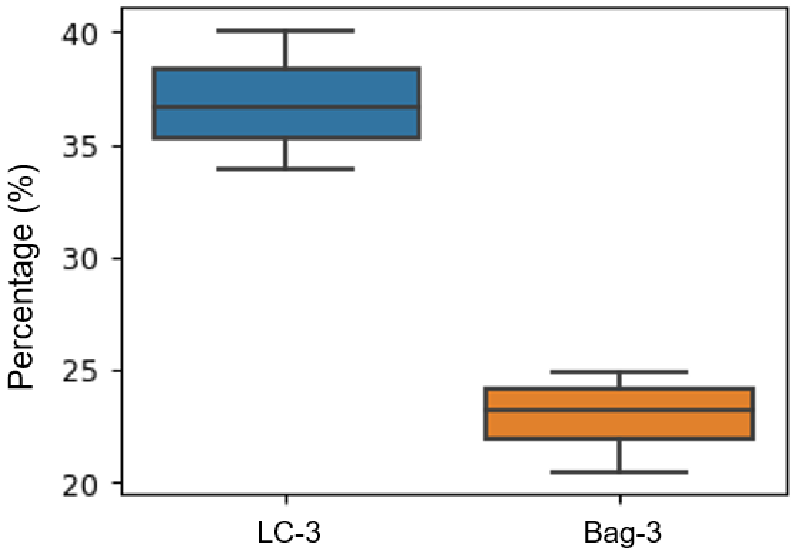


Fig. 3.8 Comparison of the distributions of the percentage of identical convolutional filters from networks of all architectures ("2-2-2-3", "1-2-2-3", "1-1-2-3", "1-1-1-3") between the 'LC-3' and 'Bag-3'.

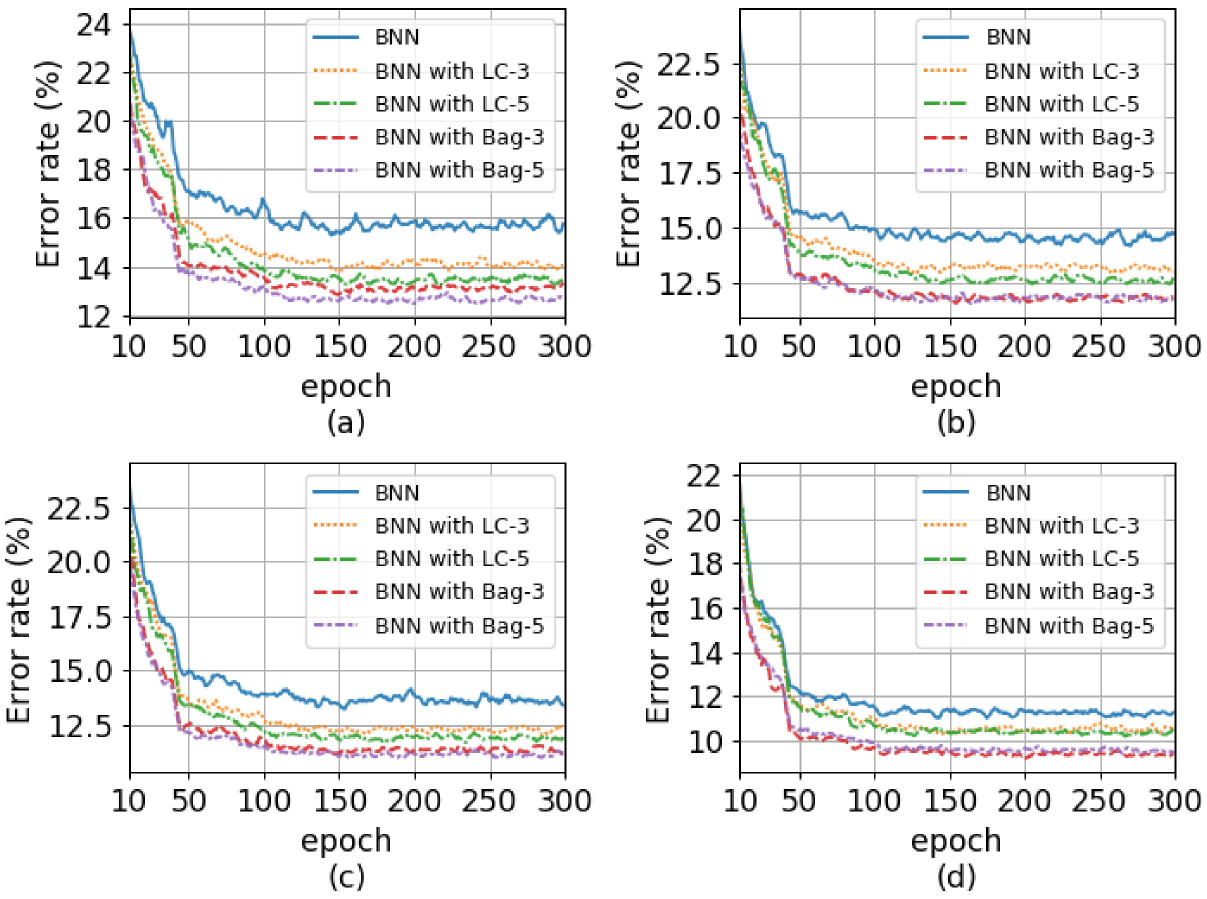


Fig. 3.9 The moving average of the validation error convergence curves of different network architectures (a) "1-1-1-3", (b) "1-1-2-3", (c) "1-2-2-3", and (d) "2-2-2-3".

The Fig. 3.9(a)-(d) show the moving average curves (moving window=5) of the validation error rate in percentage for the different networks of (a) '1-1-1-3', (b) '1-1-2-3', (c) '1-2-2-3', and (d) '2-2-2-3', respectively.

The five curves in each sub-plot represent the error rates for the different methods, where the blue solid curve labeled as 'BNN' and the orange dotted curve labeled as 'LC-3' (3 means the number of binarized layer bases is 3) denote for the baseline BNN and the linear combination with three bases, respectively. The green dash-dot curve labeled as 'LC-5', the red dashed curve labeled as 'Bag-3', and the purple densely dash-dotted curve labeled as 'Bag-5' represents the linear combination with 5 bases, the Layer Bagging with 3 bases and the Layer Bagging with 5 bases, respectively.

It is obvious from Fig. 3.9 that (1) the error convergence curves for the proposed Layer Bagging ('Bag-3', 'Bag-5') are below the error convergence curves of the baseline BNN and the linear combination ('LC-3', 'LC-5'), and (2) as the size of the model is getting smaller (from '2-2-2-3' at the bottom right to '1-1-1-3' at the top left), the difference in error rate is becoming larger, and (3) the proposed Layer Bagging method always needs the number of epochs to reach the convergence less than those for the BNN and the linear combination method. Based on the above observations, it can be said that the proposed Layer Bagging method provides significant improvements in the training time cost.

Table 6 shows the comparisons of the best top-1 classification error rate (%) among the proposed Layer Bagging method, linear combination, and the baseline BNN for the different network architectures of '1-1-1-3', '1-1-2-3', '1-2-2-3', and '2-2-2-3'.

It is shown that the error reduction capability of the proposed Layer Bagging method outperforms the linear combination and the baseline BNN for the different sizes of the networks.

Table 6 Comparisons of the validation error rates (%) for the different network architectures among the cases of using different methods.

Methods	1-1-1-3	1-1-2-3	1-2-2-3	2-2-2-3
BNN	14.88	13.72	12.96	10.66
BNN with LC-3	13.49	12.41	11.72	10.09
BNN with LC-5	12.87	12.15	11.19	9.81
BNN with Bag-3 (Ours)	12.63	10.97	10.88	8.93
BNN with Bag-5 (Ours)	11.99	11.22	10.65	9.16

The bar-plot of the error reduction rate in percentage (%) are shown in Fig. 3.10 and compared between the proposed Layer Bagging and linear combination method.

The four bars for each network architecture are divided into the two sets, which show the error reduction rates for the linear combination (LC) and the proposed Layer Bagging method (Bag) on the left and right sides, respectively. The sets for the 'LC' and 'Bag' consist of 'LC-3' and 'LC-5' and 'Bag-3' and 'Bag-5', respectively. The two things can be made clear that (1) the 'Bag' can reduce the error rate by around 13% to 20% compared to the baseline BNN and its numbers are larger than those for the linear combination method and (2) the error reduction rate

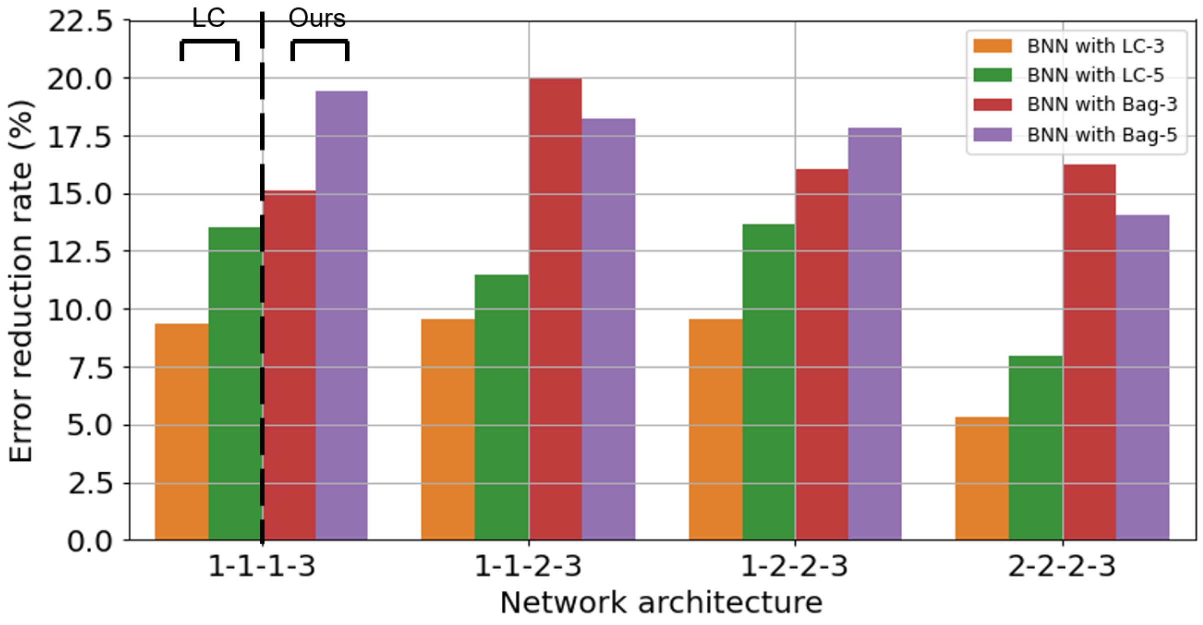


Fig. 3.10 Comparisons of the error reduction rate (%) for the different network architectures between the linear combination and the proposed Layer Bagging method.

for a smaller network like '1-1-1-3' is larger than that for a larger network like '2-2-2-3'.

Since one of the key purposes of this work is to suppress the fluctuation of the error convergence curves (i.e., to increase stability in the training process), the standard deviation of the error rates across the last 30 epochs (from the epoch of 271 to 300) and its reduction (%) (compared to the BNN) are shown in Fig. 3.11 and compared between the proposed Layer Bagging and the linear combination method.

It has been made clear from Fig. 3.11 that (1) the proposed Layer Bagging method can reduce the standard deviation of the error rates by about 0.1 to 0.13 in absolute value (reduced by about 29% to 37% from Fig. 3.11b) compared to the baseline BNN, respectively and its reductions are larger than those for the linear combination and (2) the reductions are increased as the number of layer bases increases.

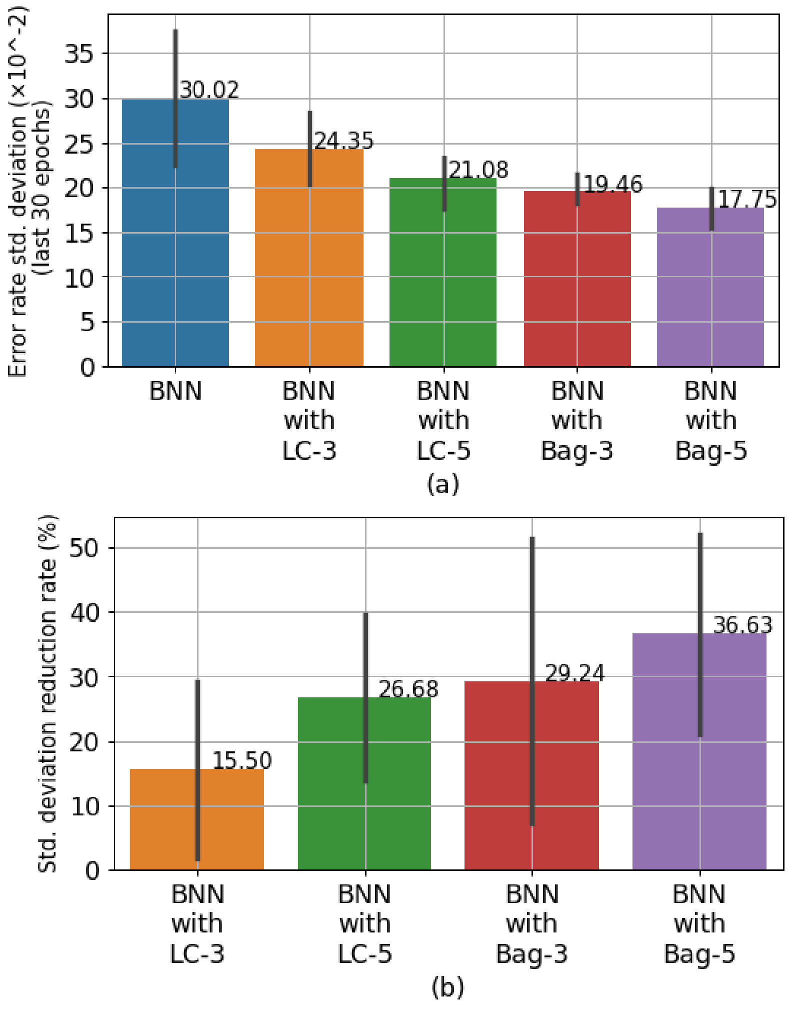


Fig. 3.11 Comparisons of (a) the standard deviation of the error rates (in last 30 epochs) and (b) standard deviation reduction rate (%), among the cases of using different methods.

In addition to evaluating on the different datasets for improving the versatility of our discussions, we also conducted the experiments by using the XNOR-Net++ as the base model which is more state-of-the-art than the BNN. The detailed results are summarized in Table 7, Table 8, and Table 9.

Table 7 Comparisons of the validation error rate and its standard deviation in the last 10% epochs on the CIFAR-10 dataset among six different methods.

Methods	Error rates (%)		Std. deviation ($\times 10^{-2}$)	
	1-1-1-3	2-2-2-3	1-1-1-3	2-2-2-3
BNN	14.88	10.66	39.88	19.24
BNN with LC-3	13.49	10.09	30.08	19.28
BNN with Bag-3 (Ours)	12.63	8.93	18.19	18.75
XNOR-Net++	14.09	10.04	30.05	23.95
XNOR-Net++ with LC-3 (Ours)	12.60	9.32	21.38	15.15
XNOR-Net++ with Bag-3 (Ours)	12.62	9.12	20.64	19.60

Table 8 Comparisons of the validation error rate and its standard deviation in the last 10% epochs on the SVHN dataset among six different methods.

Methods	Error rates (%)		Std. deviation ($\times 10^{-2}$)	
	1-1-1-3	2-2-2-3	1-1-1-3	2-2-2-3
BNN	3.58	2.71	35.73	10.02
BNN with LC-3	3.23	2.58	14.45	6.78
BNN with Bag-3 (Ours)	3.22	2.61	10.83	6.21
XNOR-Net++	3.11	2.28	32.68	7.75
XNOR-Net++ with LC-3 (Ours)	2.73	2.19	10.96	6.19
XNOR-Net++ with Bag-3 (Ours)	2.70	2.26	8.69	4.52

Table 9 Comparisons of the validation error rate and its standard deviation in the last 10% epochs on the MNIST dataset among six different methods.

Methods	Error rates (%)		Std. deviation ($\times 10^{-2}$)	
	1-1-1-3	2-2-2-3	1-1-1-3	2-2-2-3
BNN	0.67	0.58	25.88	25.23
BNN with LC-3	0.68	0.51	8.84	9.45
BNN with Bag-3 (Ours)	0.66	0.57	19.25	5.73
XNOR-Net++	0.56	0.43	42.27	6.81
XNOR-Net++ with LC-3 (Ours)	0.55	0.50	58.55	3.64
XNOR-Net++ with Bag-3 (Ours)	0.52	0.49	6.94	5.92

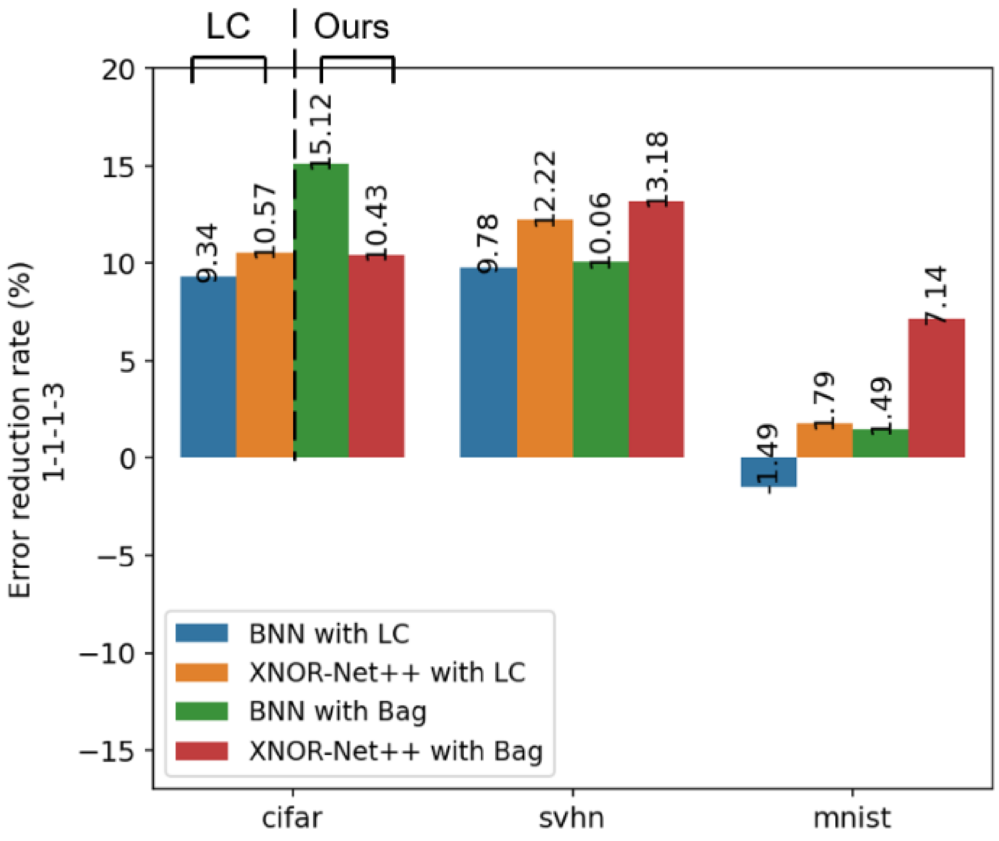
The error rate and its standard deviation in the last 10% epochs are compared among the two baseline binary models of the BNN and the XNOR-Net++, as shown in Fig. 3.12a and b for the '1-1-1-3' network and Fig. 3.12c and d for the '2-2-2-3' network, respectively.

The reductions of error rate and its standard deviation for the different datasets of CIFAR-10, SVHN, and MNIST compared to the baselines are compared among the four cases when using 'LC' or 'Bag' for the two baselines of BNN and XNOR-Net++ so that the benefits from the 'Bag' than 'LC' can be clear with versatility.

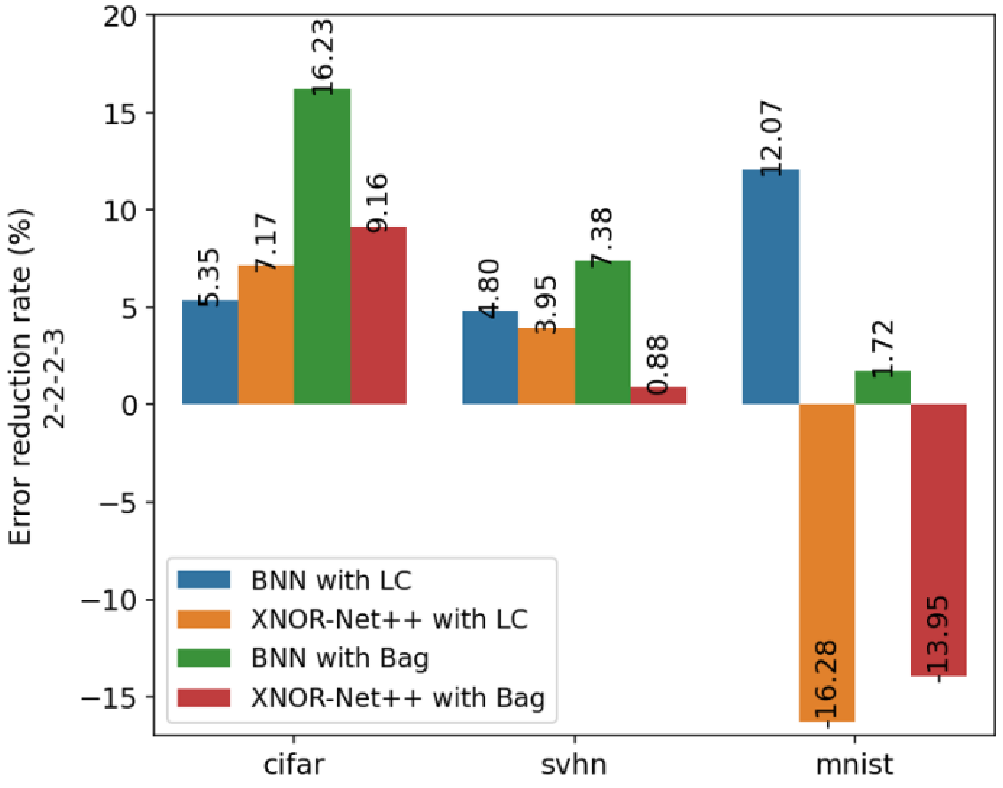
The results have shown that the proposed 'Bag' has achieved up to 16.23% and 73.41% reductions on error rate and its standard deviation, respectively, while the corresponding values for 'LC' are stopped within 12.22% and 24.57%, respectively.

When we take a look at the cases for '1-1-1-3' network on the CIFAR-10 dataset, it is found that the proposed 'Bag' reduces the error rate and its standard deviation by 15.12% and 54.39¥%, respectively, compared to the baseline of the conventional BNN.

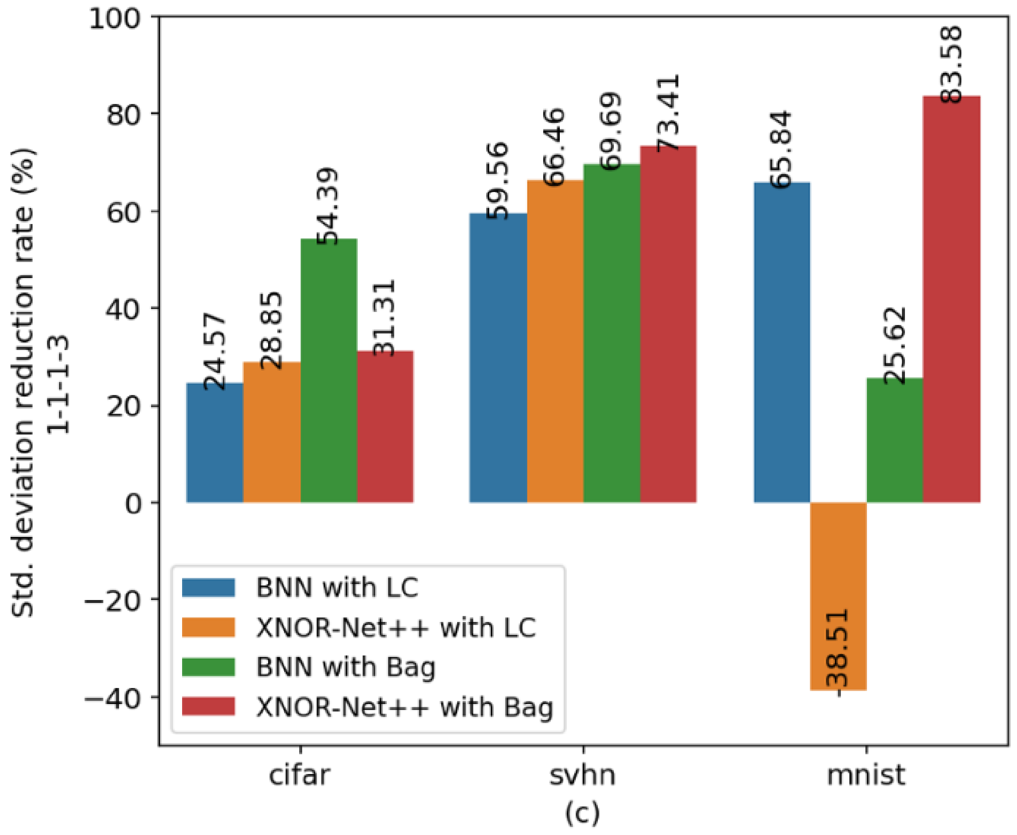
Regarding the results for the MNIST, the trends of the reductions from the baselines are less clear than the CIFAR-10 and SVHN. However, it can be said that 'Bag' provided a more stable trend for the reduction than 'LC' because of the two reasons that 1) the only one negative case is seen for the 'Bag' in Fig. 3.12 while the three negative cases for the 'LC' and 2) the absolute value of the negative value for 'Bag' is smaller than 'LC' under the same condition corresponding to the Fig. 3.12b.


LC
Ours
Error reduction rate (%) 1-1-1-3
9.34
10.57
15.12
10.43
9.78
12.22
10.06
13.18
-1.49
1.79
1.49
7.14
BNN with LC
XNOR-Net++ with LC
BNN with Bag
XNOR-Net++ with Bag
cifar
svhn
mnist

(a)


Error reduction rate (%) 2-2-2-3
5.35
7.17
16.23
9.16
4.80
3.95
7.38
0.88
12.07
-16.28
1.72
-13.95
BNN with LC
XNOR-Net++ with LC
BNN with Bag
XNOR-Net++ with Bag
cifar
svhn
mnist

(b)



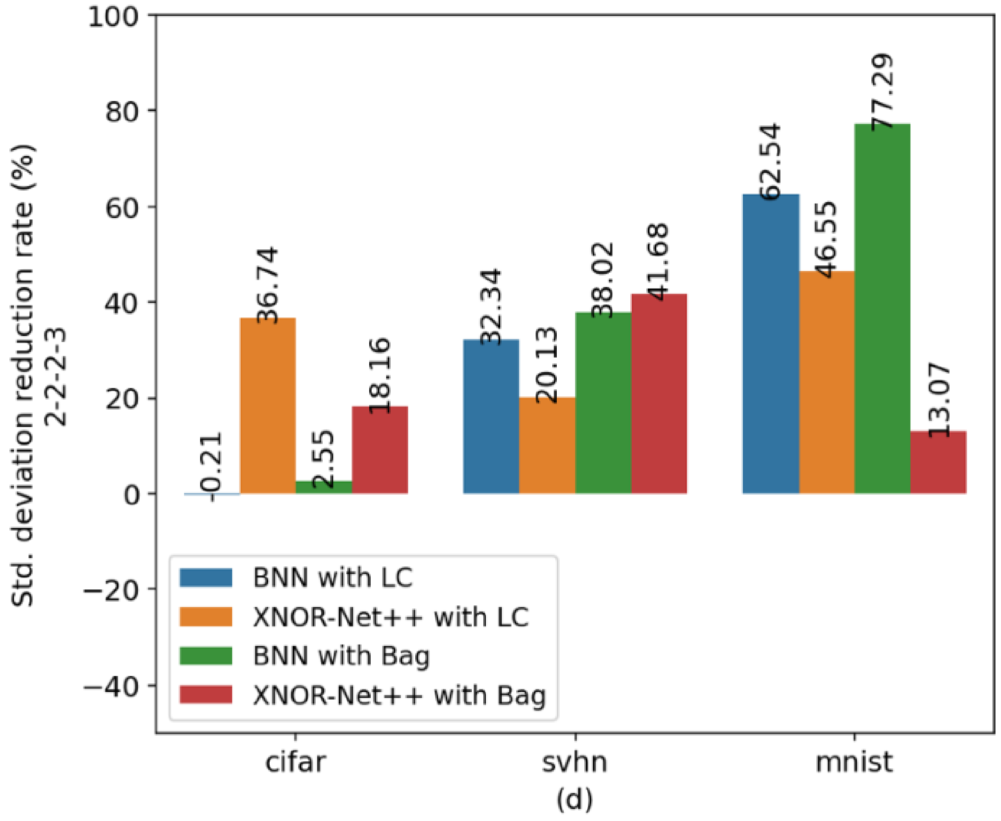


Fig. 3.12 Comparisons of error reduction rate (%) for (a) "1-1-1-3" and (b) "2-2-2-3" networks and its standard deviation reduction rate (%) for (c) "1-1-1-3" and (d) "2-2-2-3" networks among the various combinations of the methods and datasets.

3.7 Conclusion

The following points will be summarized including from the discussion to the insights based on the experimental results in order to shed light on the contributions of this work:

1. We revealed its own drawbacks of incurring too few unique patterned filters and too many additional parameters for the ABC-Net and the BENN, respectively, which both had drawn great attention in recent conferences and papers because it was believed to be the most attractive technique for binary neural networks.

2. We proposed a cost-aware layer-wise bagging technique to address both issues of the ABC-Net and the BENN.

3. We demonstrated that the percentage of incurring identical patterns in the filters is reduced from over 33% to 20% compared with the ABC-Net. The reductions in the error rate and its standard deviation by (10.43% and 31.31%) and (9.16% and 18.16%) for the networks of '1-1-1-3' and '2-2-2-3' were demonstrated and compared with the case for the state-of-the-art XNOR-Net++, respectively. We have also shown that the increase of the number of additional parameters for the proposed layer-wise bagging is suppressed to less than 2%, which is much smaller than the number of 400% for the BENN. We demonstrated that the key to success in avoidance of excessive overhead for bagging is to apply the layer-wise bagging only to the first and last layers, which is different from the conventional two cases of using float32 precision only for both first and last layers and applying the bagging to whole layers.

4. We compared the two techniques of the linear combination (LC) and the proposed layer bagging (Bag) under the same conditions to make clear the reducing trends of the error rate and its standard deviation compared with the 'LC'. We conducted the experiments under the various combinations of the baseline models of BNN and XNOR-Net++ with the techniques of 'LC' and 'Bag', datasets of CIFAR-10, SVHN, and MNIST, and different sizes of the network architectures of '1-1-1-3', '1-1-2-3', '1-2-2-3', and '2-2-2-3'. As a result, we have confirmed the advantages of the proposed 'Bag' compared with the conventional 'LC' with high versatility.

Chapter 4 Improving stability of low-precision network training with relaxation

Abstract

Binary neural networks (BNNs) have been focused by many researchers recently because of the most promising techniques to meet the desired memory footprint and inference speed requirements. However, not only the inherently poor representation with only two possible values of -1 and +1, and the training process with quantization also is an important factor which causes them suffered from the severe intrinsic instability of the error convergence, resulting in increase in prediction error and its standard deviation.

In this work, we have proposed a new training procedure with relaxed quantization to address the above issue without incurring any excessive costs, which discusses the possibility of employing relaxed quantization on both activations and weights. The experimental results have shown that the proposed method reduces the error and its standard deviation by 1.71% and 13.08% on CIFAR-10, respectively, compared to the conventional BNN training method serving as a baseline.

This chapter demonstrated and discussed such error reduction and stability performance with high versatility based on the comparison results under the various cases with the proposed and the conventional technique while changing the hyper-parameter of relaxation strength, optimizer, and using or not using batch normalization technique on the datasets of CIFAR-10 for the evaluation.

4.1 Introduction

Recently, many achievements in the fields like computer vision, speech recognition, and natural language processing have been implemented through Deep neural networks (DNNs). Therefore, researchers also begin to focus on the deployments of DNNs in the scenes with extreme limit of memory and power such as mobile phones and other portable devices with power supported by the battery.

Typical deep neural networks usually need over several megabytes of memory footprint to train the 32-bit floating-point weights. Single prediction making also requires billion level FLOPs for the DNNs. Therefore, the direct deployment of DNNs on the battery-powered devices or other portable equipment is very difficult.

Quantized neural networks (QNN) employ the low numerical precision (less than 8-bit fixed points) weights to train the DNNs. By the QNNs, huge memory footprint can be reduced to achieve higher power efficiency with only little performance drop on the prediction accuracy, compared with the full precision networks. Furthermore, bitwise calculations employed by the QNNs can be greatly accelerated on the optimized hardware at the inference time[71][72].

The training of QNN is simply abstracted as an optimization problem of minimizing some empirical risk subject to a set of constraint that characterizes the quantization of weights. Binary neural network (BNN), the 1-bit case of QNN, firstly employed a hybrid gradient updating policy which achieved impressive improvement on the accuracy performance. Based on BNN, more and more modified versions and complicated quantization algorithms including XNOR-Net, TWN[73], DoReFa-Net, etc. were proposed. BinaryRelax[74] was the most famous one of them which employed a novel relaxed quantization approach. BinaryRelax employed a novel relaxed quantization approach and achieved impressive accuracy and stable performance for Binary Weight Network (BWN), which binarizes the weights but the activations remain full precision. However, it has not yet adapted for BNN. Inspired by BinaryRelax, the general training procedure for BNN is firstly proposed in this work and detailed comparison between baseline BNN and the binary network with proposed relaxed training method in different cases via various experiment results.

4.2 Conventional relaxation technique in the BNN training process

4.2.1 Quantization

With the set of quantized weights represented as follows, all weights in the network share a single scaling factor[74].

$$Q = \mathbb{R}_+ \times \{\pm q_1, \ldots, \pm q_m\}^n \tag{4.1}$$

For b-bit quantization,

$$Q = \bigcup_{i=1}^{p} \mathcal{L}_i \tag{4.2}$$

is the union of p distinct one-dimensional subspaces $\mathcal{L}_i \subset \mathbb{R}^n, i = 1,2,\ldots,p$, where $\mathcal{L}_\mathrm{i} = \{s \cdot L_i : s \in \mathbb{R}\}$.

Given a float weight vector W, its quantization W_Q is basically the projection of W onto the set Q, which gives rise to the optimization problem

$$W_Q = \underset{z \in Q}{\operatorname{argmin}} \|z - W\| = \operatorname{proj}_Q(W) \tag{4.3}$$

Note that Q is a non-convex set, then the projection may not be unique. In that case, we just assume W_Q is one of them. The above projection/quantization problem can be reformulated as

$$(s^*, Q^*) = \underset{s,Q}{\operatorname{argmin}} \|s \cdot Q - W\|^2 \quad \text{subject to} \quad Q \in \{\pm q_1, \ldots, \pm q_m\}^n$$

(4.4)

The quantization of W is then given by $\text{proj}_Q(W) = s^* \cdot Q^*$. (4.4) becomes a constrained K-means clustering problem of one-dimensional points. The centroids are of the form $\pm(s \cdot q_j)$ with $1 \leq j \leq m$, and they are determined by a single parameter s since q_j's are fixed. For uniform quantization where $q_j = j - 1$, these centroids are equi-spaced. Given s, the assignment of float weights is then governed by Q. So, the problem (4.4) in principle can be solved by a variant of Lloyd's algorithm, which iterates between the assignment step (Q-update) and centroid update step (s-update).

However, such procedure is impractical, as the quantization is needed in every iteration of training. It has been shown that the closed form (exact) solution of (4.4) can be computed at $O(n)$ complexity for binarization where $Q \in \{\pm 1\}^n$:

$$s^* = \frac{\|W_Q\|_1}{n}, Q_i^* \begin{cases} 1 & \text{if } W_i \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

(4.5)

In the case of ternarization where $Q \in \{0, \pm 1\}^n$:

$$t^* = \underset{1 \leq t \leq n}{\text{argmax}} \frac{\|W_{[t]}\|_1^2}{t}, s^* = \frac{\|W_{[t^*]}\|_1}{t^*}, Q^* = \text{sign}(W_{[t^*]}),$$

(4.6)

where $W_{[t^*]} \in \mathbb{R}^n$ keeps the t largest component in magnitude of W, while zeroing out the others. For quantization with wider bit-width ($b > 2$), accurately solutions of (4.4) becomes computationally intractable. Empirical formulas have thus been proposed for an approximate quantized solution, and they turn out to be sufficient for practical use[75].

4.2.2 Relaxed quantization

Let us begin with the alternative form of DNNs quantization problem

$$\min_{W_Q \in \mathbb{R}^n} f(W_Q) + \frac{\lambda}{2} \text{dist}(W_Q, Q)^2$$

(4.7)

where $\lambda > 0$ is the regularization parameter.

Based on the BNN gradient update policy, a two-line solver for the minimization problem of (4.7) can be expressed as follows.

$$\begin{cases} W^{k+1} = W^k - \gamma_k \nabla f_k(W_Q^k) \\ W_Q^{k+1} = \underset{W_Q \in \mathbb{R}^n}{\text{argmin}} \frac{1}{2} \|W_Q - W^{k+1}\|^2 + \frac{\lambda}{2} \text{dist}(W_Q, Q)^2 \end{cases}$$

(4.8)

The algorithm constructs two sequences: an auxiliary sequence of float weights W^k and a sequence of nearly quantized weights W_Q^k. The mismatch of non-continuous projection and continuous gradient descent is resolved by the relaxed quantization step in (4.8), which calls for computing the proximal mapping of the function $\lambda/2 \cdot \mathrm{dist}(W_Q, Q)^2$.

Let the following expression be the quantization of W^{k+1},

$$\mathrm{proj}_Q(W^{k+1}) = \underset{W_Q \in Q}{argmin} \frac{1}{2} \left\| W_Q - W^{k+1} \right\|^2$$

(4.9)

then the solution to relaxed quantization sub-problem in (4.8) is given by

$$W_Q^{k+1} = \frac{\lambda \mathrm{proj}_Q(W^{k+1}) + W^{k+1}}{\lambda + 1}$$

(4.10)

Note that the exact quantization is still needed to perform relaxed quantization. The update W_Q^{k+1} is essentially a linear interpolation of W^{k+1} and its quantization $\mathrm{proj}_Q(W^{k+1})$, and λ controls the weighted average. W_Q^{k+1} is thus not quantized because $W_Q^{k+1} \notin Q$, but W_Q^{k+1} approaches Q as λ increases. Hereby a continuation strategy is adopted and let λ grow slowly. Specifically, the λ is increased after a certain number of epochs by a factor $\rho > 1$. Intuitively, the relaxation with continuation will help skip over some bad local minima located in Q, because they are not local minima of the relaxed formulation in general.

The BinaryRelax algorithm is summarized in Alg. 4.1.

```
**Input**: input batch; number of epochs, batches; schedule of learning rate $\gamma_k$; growth factor $\rho > 1$
for $i \leftarrow 1$ to *nb-epoch* do
    for $j \leftarrow 1$ to *nb-batch* do
        $input_Q \leftarrow \text{proj}_Q(input)$
        *loss*←*forward*$(W_Q^k, input_Q))$
        $\nabla f_k(W_Q^k) \leftarrow backward(loss, W_Q^k)$
        $W^{k+1} \leftarrow W^k - \gamma_k \nabla f_k(x^k)$
        if $i \leq T$ then
            $W_Q^{k+1} \leftarrow (\lambda_k \text{proj}_Q(W^{k+1}) + W^{k+1})/(\lambda_k + 1)$ // Phase I
            if increase $\lambda$ then
                $\lambda_{k+1} \leftarrow \rho\lambda_k$
            end if
        else
            $W_Q^{k+1} \leftarrow \text{proj}_Q(W^{k+1})$ // Phase II
        end if
        k← $k + 1$
    end for
end for
```

Alg. 4.1 The pseudocode for BinaryRelax

In order to obtain quantized weights in the end, the relaxation mode was turned off in the Phase II and enforce quantization.

4.3 BNN training with relaxation in both activations and weights

The BinaryRelax (weight relaxation) alleviated the variance of the weights during the training process. However, for the conventional BNN training process also quantizes the activation, the variance from the quantization of the activation harms the stability of the training and becomes a new problem which has to be solved. Therefore, to solve such issue, we propose new training procedures for BNN to employ relaxation to the activations (activation relaxation) and both activation and weight (activation & weight relaxation).

The Algorithm above shows the details of proposed new training procedures with relaxation

```
**Input**: input batch; number of epochs, batches; schedule of learning rate $\gamma_k$; growth
factor $\rho > 1$
for $i \leftarrow 1$ to *nb-epoch* do
    for $j \leftarrow 1$ to *nb-batch* do
        if $i \leq T_a$ then
            $input_Q \leftarrow (\lambda_k \mathrm{proj}_Q(input) + input)/(\lambda_k + 1)$ // Phase I
        else
            $input_Q \leftarrow \mathrm{proj}_Q(input)$ // Phase II
        *loss*←*forward*$(W_Q^k, input_Q)$)
        $\nabla f_k(W_Q^k) \leftarrow backward(loss, W_Q^k)$
        $W^{k+1} \leftarrow W^k - \gamma_k \nabla f_k(W_Q^k)$
        if $i \leq T_w$ then
            $W_Q^{k+1} \leftarrow (\lambda_k \mathrm{proj}_Q(W^{k+1}) + W^{k+1})/(\lambda_k + 1)$ // Phase II
            if increase $\lambda$ then
                $\lambda_{k+1} \leftarrow \rho\lambda_k$
            end if
        else
            $W_Q^{k+1} \leftarrow \mathrm{proj}_Q(W^{k+1})$ // Phase II
        end if
        k← $k + 1$
    end for
end for
```

Alg. 4.2 The pseudocode for proposed Activation &Weight Relaxation

for the BNN. $input$ denotes the input batch or activations from previous layer. *nb-epoch* and *nb-batch* stand for the number of epochs and batches, respectively. T_a and T_w are the number of epochs to split the Phase I or II of the relaxed quantization for the activations and weights, respectively. Here, we set separate T_a and T_w parameters to achieve the controlling of different mode of the training.

4.4 Experiments and results

We conducted several image classification experiments mainly based on the CIFAR-10 dataset, which is a common benchmark dataset used for the image classification task. The dataset consists of 60k (train/test: 50k/10k) 32 by 32 color images in 10 classes with 6000 images in each class. The program for the experiments is implemented in PyTorch and the program is executed on Ubuntu 16.04 with the NVidia GeForce GTX 1080 Ti graphic card.

To make a fair comparison, the similar setups and model architecture compared to the original BNN paper are used in this experiment as much as possible.

4.4.1 Experiment settings

The preprocessing part of the experiment simply used random crop, random horizontal flip, and normalizing with image net statistics. The basic binary ResNet-20 CNN model architecture used in this work is shown in following Fig. 4.1.

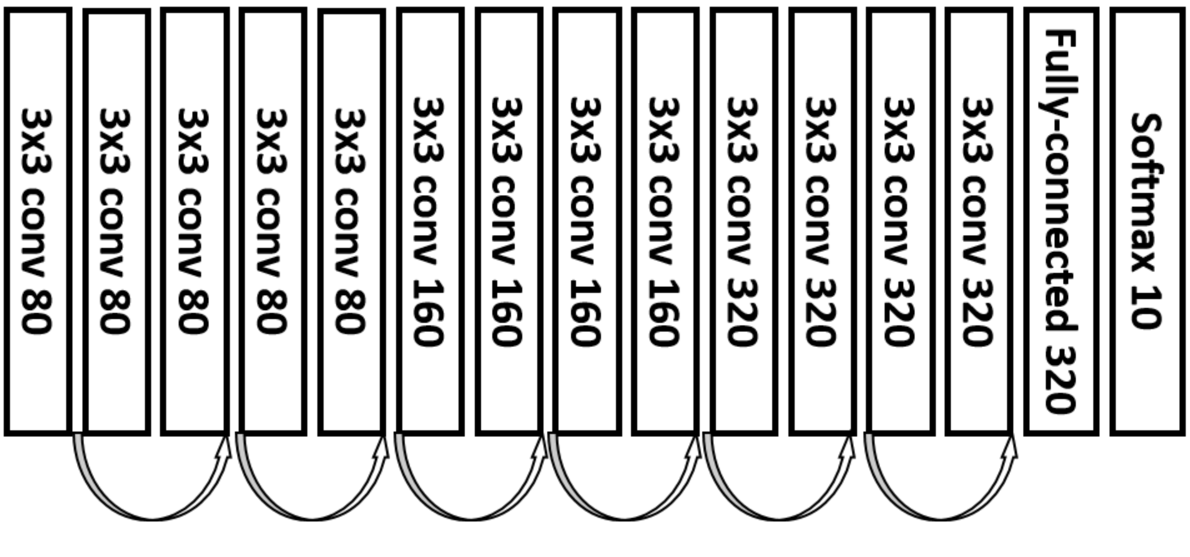


Fig. 4.1 The ResNet-20 CNN model architecture used in this work

In Fig. 4.1, "3×3 conv 80" represents a 3 by 3 convolution layer with 80 channels followed by the batch normalization layer and ReLU activation layer.

The ResNet-20 CNN model architecture is mainly composed of three convolution layer blocks (each block consists of four layers and two residual paths) followed by three fully-connected layers with the output feature numbers of 320 before the final 'Softmax' layer which outputs the classification predictions.

In order to discuss the dependencies on some conditions, we conducted the experiments while changing the conditions as shown in Table 10. The conditions can be varied by changing the combination of 1) weight relaxation only, activation relaxation only and activation & weight both relaxation, 2) using various different lambdas of {0.125, 0.25, 0.5, 1, 2, 4, 8, 16} which control the strength of the relaxation and take the best result point, 3) using Adam or SGD optimizer during training and 4) whether using trained or fixed batch normalization ('fixed BN' means that the parameters of the BN layer are fixed to the initial values).

Table 10 The variations of the experiments in different cases

Methods	# of lambda	If use Adam or SGD optimizer	If use trained or fixed BN
Weight Relaxation	8 values	True/False	True/False
Activation Relaxation	8 values	True/False	True/False
Activation & Weight Relaxation	8 values	True/False	True/False

4.4.2 Overall results

The Fig. 4.2 shows the comparison of error rate (%) curve of 4 methods including BNN, weight relaxation (abbreviated as 'WeightRlx' in the figure), activation relaxation (abbreviated as 'ActRlx' in the figure), and activation and weight relaxation (abbreviated as 'ActWeightRlx' in the figure).

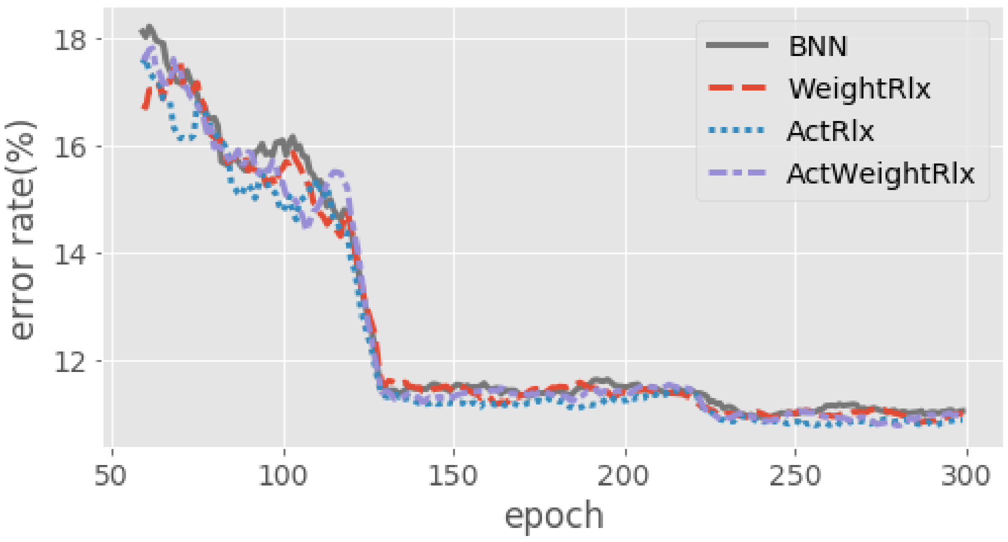


Fig. 4.2 The comparison of error curve error rate (%) among the 4 methods

It is shown that the proposed relaxed training methods ('WeightRlx', 'ActRlx', and 'ActWeightRlx') are almost always under the curve of BNN. It is shown that the proposed methods can achieve lower error rates within fewer training epochs which can improve the efficiency in the training time.

The Fig. 4.3 shows the comparison of 'current/final' of absolute mean of weight (full precision) by epoch among the 4 methods. The x-axis is the epoch of training. The y-axis is the ratio of current and final value of absolute mean of weight (full precision). This simply measures how the current weight is converging to the final weight by epoch. We take absolute mean of full-precision weight to see the average of the absolute value of the weight. We take the 'current/final' ratio to see the progress of the training.

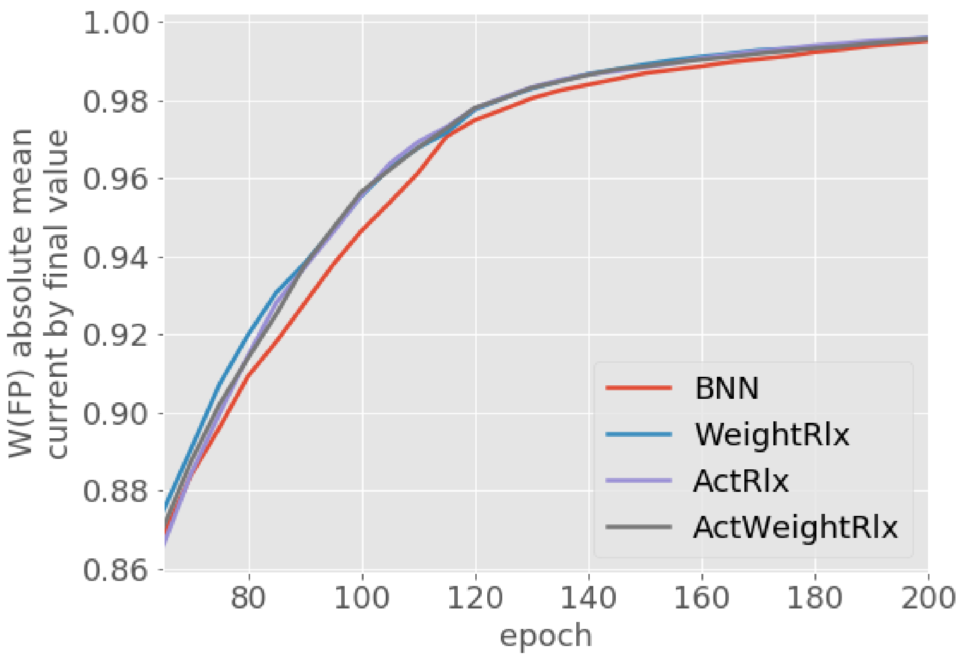


Fig. 4.3 Comparison of 'current/final' of absolute mean of weight (full precision) by epoch among the 4 methods

It is shown that in epoch 60~160 the ratios of proposed methods are increasing much quicker than the ratio of BNN. Therefore, with the proposed methods, the training process is quicker compared to the conventional BNN training method.

4.4.3 Result of error rate improvement

The following Fig. 4.4 shows the comparisons of error rate improvement (%) in different cases among the BNN and the proposed methods ('WeightRlx', 'ActRlx', and 'ActWeightRlx').

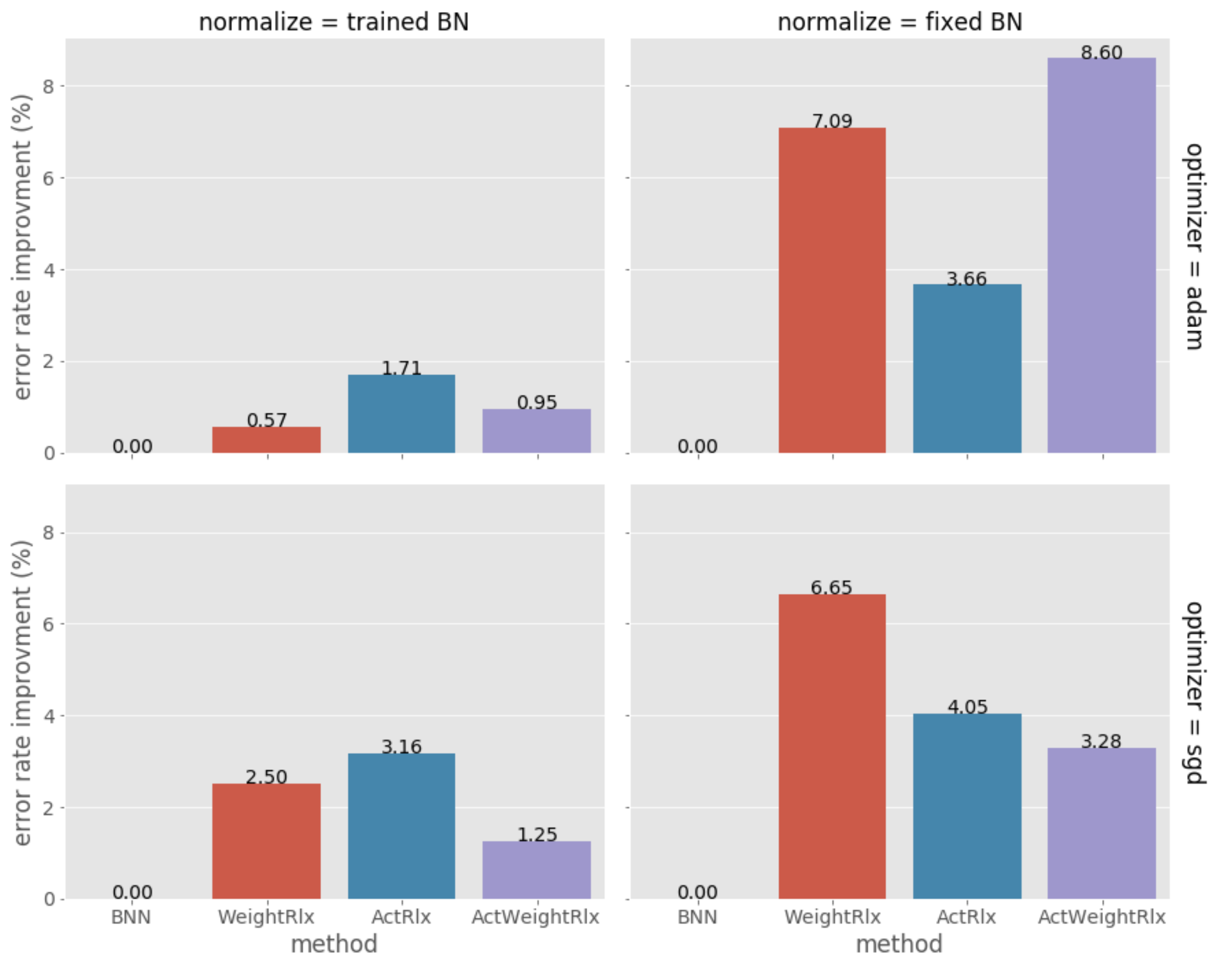


Fig. 4.4 Comparison of error rate improvement (%) in different cases among the 4 methods

The error rate improvement (%) is calculated by $100\% \times (error_{BNN} - error)/error_{BNN}$. The row of the sub-figures means the case of using the Adam or SGD optimizer. The column of the sub-figures means the case of using the trained batch normalization or fixed batch normalization.

It is shown that all the results of using proposed methods have improvements compared to the traditional BNN training method. In the case of 'optimizer=adam & normalize=trained BN', which is the most common case for the model training, the proposed activation relaxation can achieve an improvement by 1.71% compared to BNN baseline. In the case of 'optimizer=adam & normalize=fixed BN', the proposed activation & weight relaxation can achieve an improvement by 8.60% compared to BNN baseline. In the case of 'optimizer=sgd', the proposed training with relaxation methods also can achieve improvements by 1.25%~6.65% compared to BNN baseline.

4.4.4 Result of error rate standard deviation improvement

The following Fig. 4.5 shows the comparisons of error rate standard deviation improvement (%) in different cases among the BNN and the proposed methods ('WeightRlx', 'ActRlx', and 'ActWeightRlx').

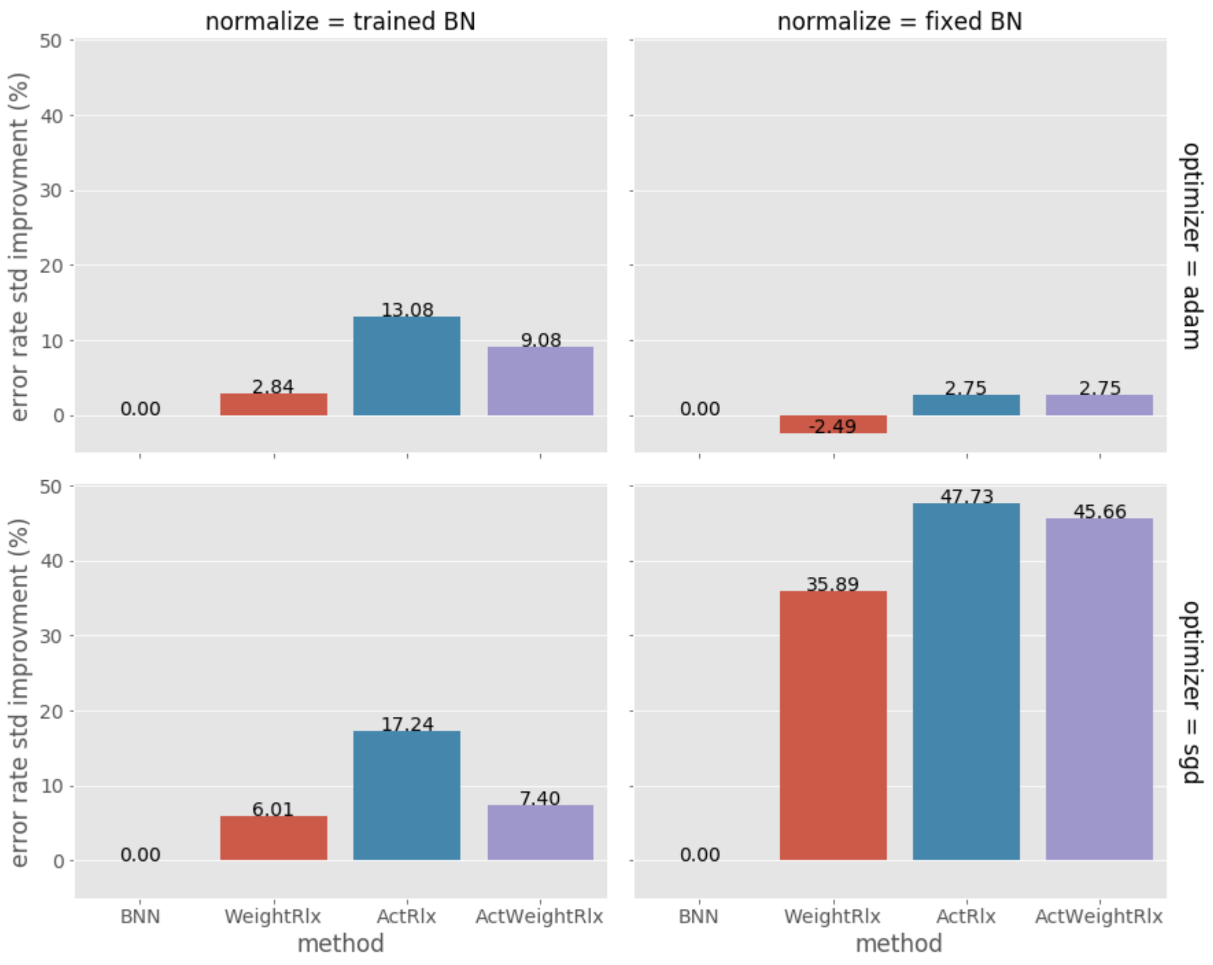


Fig. 4.5 Comparison of error rate standard deviation improvement (%) in different cases among the 4 methods

The error rate standard deviation improvement (%) is calculated by $100\% \times (\sigma(error_{BNN}) - \sigma(error))/\sigma(error_{BNN})$. The row of the sub-figures means the case of using the Adam or SGD optimizer. The column of the sub-figures means the case of using the trained batch normalization or fixed batch normalization.

It is shown that almost all the results of using proposed methods have improvements compared to the traditional BNN training method. In the case of 'optimizer=adam & normalize=trained BN', which is the most common case for the model training, the proposed activation relaxation can achieve an improvement by 13.08% compared to BNN baseline. In the case of 'optimizer=adam & normalize=fixed BN', the proposed activation relaxation and activation & weight relaxation can achieve an improvement by 2.75% compared to BNN baseline. In the case of 'optimizer=sgd & normalize=trained BN ', the proposed training with relaxation methods also can achieve improvements by 6.01%~17.24% compared to BNN baseline. In the case of 'optimizer=sgd & normalize=fixed BN ', the most difficult case for training, the proposed training with relaxation methods can achieve improvements by 35%~47% compared to BNN baseline, showing that the proposed training with relaxation methods works even better during some conditions which are hard for training.

4.5 Conclusion

This chapter mainly discussed the relaxation techniques in the BNN training process. The conventional relaxed training method was introduced firstly, which only employed the relaxed quantization to the weights. To adapt the conventional method to the BNN and further improve the stability of the BNN models in the training process, we proposed the weight relaxation, activation relaxation, and activation & weight relaxation to employ the relaxed quantization to the activations or both the activations and the weights.

To compare the proposed methods and the conventional method in detail and fairly, we conducted a number of experiments in various cases including changing optimizers, using trained or fixed batch normalization, and with different strengths of the relaxation. Through the experiments on the CIFAR-10 dataset, we confirmed that the proposed activation & weight relaxation method can improve the error rate by over 1.71% and improve the standard deviation of the error rates by over 13.08%, compared to the conventional BNN method.

Chapter 5 Conclusion

This work mainly researched the techniques to improve the bit energy efficiency of the machine learning systems. With the conventional implementation of in-memory classifier reaching great energy saving and available accuracy, hardware errors and nonlinear noises existed in the bit cells as well as the time-dependent variability makes the model need more memory arrays to achieve available performance, which calls the demand on the methods to reduce the impact of variability as well as the number of memory column arrays. The conventional network quantization and training techniques also need breakthroughs to achieve higher accuracy and more stable training process.

Based on the conventional implementation for in-memory machine learning classifier employing 1-Bit Constraint-resolution-regression-based error adaptive classifier boosting algorithm, in order to achieve a further level of the energy saving, a two-step processing method was proposed. First, an adaptive pruning process was proposed based on the different difficulty between the classification tasks. Then, from the view of heuristic search, maximal accuracy gains first criterion based greedy search and its fast version were proposed. Based on weakest boosted classifier first criterion, the worst-care selecting algorithm was proposed. With comparison of the accuracy by proposed processing, some typical results of experiment verified the effect of column reduction by proposed algorithms.

The stability and accuracy loss issue of conventional binary neural networks was revealed. To improve the stability and accuracy of low precision networks, a layer-wise ensemble technique for BNN was proposed in this work. Through the experiments on the CIFAR-10 dataset, it is shown that the proposed technique can reduce the error and its standard deviation by 15% and 54%, respectively, compared to the BNN serving as a baseline.

To further improve the BNN training with relaxed quantization, we propose new training procedures with relaxation of both weights and activations for BNN. It is proved to be able to alleviate the variance by the conventional BNN training process (>2%) with the experiments of various cases including different optimizers and with or without batch normalization.

Acknowledgements

I would first like to thank my thesis supervisor Prof. Hiroyuki Yamauchi of Graduate school of Computer Science & Engineering at Fukuoka Institute of Technology. The door of Prof. Yamauchi's office was usually open to me whenever I got into some problems or came into a question whether about my research or writing. He always not only the writing contents of the papers and this thesis, but also gave many constructive advice in the respective of presentation preparation, form of illustration and contents arrangement.

I would also thank the professors on the doctoral review committee, Prof. Eguchi, Prof. Oida, and Prof. Fukumoto. Their insightful comments and useful advice greatly helped me to refine my PhD thesis.

I would like to thank master student Yu and Li at the same laboratory who were also involved in this research project. Without their active discussion and inspirations, this research work could not have been successfully completed as now.

Finally, I must give my faithful gratitude to my parents and friends for providing me with unfailing support and continuous encouragement throughout these three and a half years and through the process of researching and writing my PhD thesis. This accomplishment would not have been possible without them. Thank you.

Publication list

[1] Jiazhen Xi and Hiroyuki Yamauchi. "A Column Reduction Technique for an In-Memory Machine-Learning Classifier." *International Journal of Machine Learning and Computing* 8, no. 2 (2018).

[2] Jiazhen Xi and Hiroyuki Yamauchi. "A Layer-wise Ensemble Technique for Binary Neural Network." *International Journal of Pattern Recognition and Artificial Intelligence* (2021): 2152011.

[3] Peng Yu, Jiazhen Xi, and Hiroyuki Yamauchi. "Time and Environment Dependency Aware Fuel Consumption Tracking Method for Improving Drivers and Trucks Management." Circuits / Systems, Computers and Communications (ITC-CSCC), The 36th International Technical Conference on, ITC-CSCC 2021, Jun. 2021.

[4] Jiazhen Xi and Hiroyuki Yamauchi. "Layer-wise Ensemble for Binary Neural Networks." The 1st NKUST-FIT International Seminar on Advanced Technology, Dec. 2018.

[5] Jiazhen Xi and Hiroyuki Yamauchi. "Approximately Quantizing Algorithm for In-memory Machine Learning Classifier." The 1st NKUST-FIT International Seminar on Advanced Technology, Dec. 2018.

[6] 鶴隆介, セキカテイ, 山内寛行. "SRAM アレイ内機械学習分類器のコラムアレイ削減手法." 令和元年度(第 72 回）電気・情報関係学会九州支部連合大会, 12-1A-09 (2019-9).

References

[1] Mingu Kang, Min-Sun Keel, Naresh R. Shanbhag, Sean Eilert, and Ken Curewitz. "An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM." In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8326-8330. IEEE, 2014.

[2] Puneet Gupta, Yuvraj Agarwal, Lara Dolecek, Nikil Dutt, Rajesh K. Gupta, Rakesh Kumar, Subhasish Mitra et al. "Underdesigned and opportunistic computing in presence of hardware variability." *IEEE Transactions on Computer-Aided Design of integrated circuits and systems* 32, no. 1 (2012): 8-23.

[3] Todd Sepke, Peter Holloway, Charles G. Sodini, and Hae-Seung Lee. "Noise analysis for comparator-based circuits." *IEEE Transactions on Circuits and Systems I: Regular Papers* 56, no. 3 (2008): 541-553.

[4] Junjie Lu, Steven Young, Itamar Arel, and Jeremy Holleman. "A 1 TOPS/W analog deep machine-learning engine with floating-gate storage in 0.13 µm CMOS." *IEEE Journal of Solid-State Circuits* 50, no. 1 (2014): 270-281.

[5] David J. DeWitt, Randy H. Katz, Frank Olken, Leonard D. Shapiro, Michael R. Stonebraker, and David A. Wood. "Implementation techniques for main memory database systems." In *Proceedings of the 1984 ACM SIGMOD international conference on management of data*, pp. 1-8. 1984.

[6] Mark Horowitz. "1.1 computing's energy problem (and what we can do about it)." In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10-14. IEEE, 2014.

[7] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberley Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. "Intelligent RAM (IRAM): Chips that remember and compute." In *1997 IEEE International Solids-State Circuits Conference. Digest of Technical Papers*, pp. 224-225. IEEE, 1997.

[8] Duncan G. Elliott, Michael Stumm, W. Martin Snelgrove, Christian Cojocaru, and Robert McKenzie. "Computational RAM: Implementing processors in memory." *IEEE Design & Test of Computers* 16, no. 1 (1999): 32-41.

[9] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, William J. Dally, and Mark Horowitz. "Smart memories: A modular reconfigurable architecture." In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No. RS00201)*, pp. 161-171. IEEE, 2000.

[10] Hans Jurgen Mattausch, Takayuki Gyohten, Yoshihiro Soda, and Tetsushi Koide. "Compact associative-memory architecture with fully parallel search capability for the minimum Hamming distance." *IEEE Journal of Solid-State Circuits* 37, no. 2 (2002): 218-227.

[11] Yusuke Oike, Makoto Ikeda, and Kunihiro Asada. "A high-speed and low-voltage associative co-processor with exact Hamming/Manhattan-distance estimation using word-parallel and hierarchical search architecture." *IEEE Journal of Solid-State Circuits* 39, no. 8 (2004): 1383-1387.

[12] Roman Genov and Gert Cauwenberghs. "Kerneltron: support vector" machine" in silicon." *IEEE Transactions on Neural Networks* 14, no. 5 (2003): 1426-1434.

[13] Christoforos E. Kozyrakis and David A. Patterson. "Scalable, vector processors for embedded systems." *IEEE Micro* 23, no. 6 (2003): 36-45.

[14] Zhuo Wang, Jintao Zhang, and Naveen Verma. "Reducing quantization error in low-energy FIR filter accelerators." In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1032-1036. IEEE, 2015.

[15] Vinay K. Chippa, Debabrata Mohapatra, Anand Raghunathan, Kaushik Roy, and Srimat T. Chakradhar. "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency." In *Design automation conference*, pp. 555-560. IEEE, 2010.

[16] Larkhoon Leem, Hyungmin Cho, Jason Bau, Quinn A. Jacobson, and Subhasish Mitra. "ERSA: Error resilient system architecture for probabilistic applications." In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 1560-1565. IEEE, 2010.

[17] Naveen Verma, Kyong Ho Lee, Kuk Jin Jang, and Ali Shoeb. "Enabling system-level platform resilience through embedded data-driven inference capabilities in electronic devices." In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5285-5288. IEEE, 2012.

[18] Kyong Ho Lee, Sun-Yuan Kung, and Naveen Verma. "Improving kernel-energy trade-offs for machine learning in implantable and wearable biomedical applications." In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1597-1600. IEEE, 2011.

[19] Kyong Ho Lee, and Naveen Verma. "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals." *IEEE Journal of Solid-State Circuits* 48, no. 7 (2013): 1625-1637.

[20] Zhuo Wang, and Naveen Verma. "Enabling hardware relaxations through statistical

learning." In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 319-326. IEEE, 2014.

[21] Zhuo Wang, Robert Schapire, and Naveen Verma. "Error-adaptive classifier boosting (EACB): Exploiting data-driven training for highly fault-tolerant hardware." In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3884-3888. IEEE, 2014.

[22] Zhuo Wang, Robert E. Schapire, and Naveen Verma. "Error adaptive classifier boosting (EACB): Leveraging data-driven training towards hardware resilience for signal inference." *IEEE Transactions on Circuits and Systems I: Regular Papers* 62, no. 4 (2015): 1136-1145.

[23] Mingu Kang, Sujan K. Gonugondla, Min-Sun Keel, and Naresh R. Shanbhag. "An energy-efficient memory-based high-throughput VLSI architecture for convolutional networks." In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1037-1041. IEEE, 2015.

[24] Yann LeCun and Corinna Cortes. "MNIST handwritten digit", AT&T Labs [Online]. Available: http://yann.2010,database,lecun.com/exdb/mnist

[25] Jintao Zhang, Zhuo Wang, and Naveen Verma. "18.4 A matrix-multiplying ADC implementing a machine-learning classifier directly with data conversion." In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*, pp. 1-3. IEEE, 2015.

[26] Warren Rieutort-Louis, Tiffany Moy, Zhuo Wang, Sigurd Wagner, James C. Sturm, and Naveen Verma. "A large-area image sensing and detection system based on embedded thin-film classifiers." *IEEE Journal of Solid-State Circuits* 51, no. 1 (2015): 281-290.

[27] Scott Hanson, and Dennis Sylvester. "A 0.45–0.7 V sub-microwatt CMOS image sensor for ultra-low power applications." In *2009 symposium on VLSI circuits*, pp. 176-177. IEEE, 2009.

[28] Mingu Kang, Sujan Gonugondla, Ameya Patil, and Naresh Shanbhag. "A 481pJ/decision 3.4 M decision/s multifunctional deep in-memory inference processor using standard 6T SRAM array." *arXiv preprint arXiv:1610.07501* (2016).

[29] Amir Morad, Leonid Yavits, and Ran Ginosar. "GP-SIMD processing-in-memory." *ACM Transactions on Architecture and Code Optimization (TACO)* 11, no. 4 (2015): 1-26.

[30] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." *IEEE journal of solid-state circuits* 52, no. 1 (2016): 127-138.

[31] Yunhui Guo. "A survey on methods and theories of quantized neural networks." *arXiv preprint arXiv:1808.04752* (2018).

[32] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. "Pruning and Quantization for Deep Neural Network Acceleration: A Survey." *arXiv preprint arXiv:2101.09671* (2021).

[33] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey." *The Journal of Machine Learning Research* 20, no. 1 (2019): 1997-2017.

[34] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." *arXiv preprint arXiv:1503.02531* (2015).

[35] Michael Mathieu, Mikael Henaff, and Yann LeCun. "Fast training of convolutional networks through ffts." *arXiv preprint arXiv:1312.5851* (2013).

[36] Kumar Chellapilla, Sidd Puri, and Patrice Simard. "High performance convolutional neural networks for document processing." In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

[37] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. "What is the state of neural network pruning?." *arXiv preprint arXiv:2003.03033* (2020).

[38] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. "Survey and benchmarking of machine learning accelerators." In *2019 IEEE high performance extreme computing conference (HPEC)*, pp. 1-9. IEEE, 2019.

[39] Marius Cornea. "Intel AVX-512 instructions and their use in the implementation of math functions." *Intel Corporation* (2015).

[40] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S. Vetter. "Nvidia tensor core programmability, performance & precision." In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 522-531. IEEE, 2018.

[41] AMD, Radeon Instinct™ MI25 Accelerator [Online]. Available: https://www.amd.com/en/graphics/servers-radeon-instinct-mi.

[42] Arm, 2015. ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile [Online]. Available: https://developer.arm.com/documentation/ddi0487/latest.

[43] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates et al. "In-datacenter performance analysis of a tensor processing unit." In *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1-12. 2017.

[44] Yang Jiao, Liang Han, and Xin Long. "Hanguang 800 NPU–The Ultimate AI Inference Solution for Data Centers." In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1-29. IEEE

Computer Society, 2020.

[45] Jian Ouyang, Mijung Noh, Yong Wang, Wei Qi, Yin Ma, Canghai Gu, SoonGon Kim et al. "Baidu Kunlun An AI processor for diversified workloads." In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pp. 1-18. IEEE Computer Society, 2020.

[46] Duncan JM Moss, Eriko Nurvitadhi, Jaewoong Sim, Asit Mishra, Debbie Marr, Suchit Subhaschandra, and Philip HW Leong. "High performance binary neural networks on the Xeon+ FPGA™ platform." In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1-4. IEEE, 2017.

[47] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. "Efficient processing of deep neural networks: A tutorial and survey." *Proceedings of the IEEE* 105, no. 12 (2017): 2295-2329.

[48] Jintao Zhang, Zhuo Wang, and Naveen Verma. "A machine-learning classifier implemented in a standard 6T SRAM array." In *2016 ieee symposium on vlsi circuits (vlsi-circuits)*, pp. 1-2. IEEE, 2016.

[49] Yoav Freun. "Boosting a weak learning algorithm by majority." *Information and computation* 121, no. 2 (1995): 256-285.

[50] Zhuo Wang and Naveen Verma. "A low-energy machine-learning classifier based on clocked comparators for direct inference on analog sensors." *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, no. 11 (2017): 2954-2965.

[51] Gurobi Optimization, Inc. (2015). Gurobi Optimizer Reference Manual. [Online]. Available: http://www.gurobi.com

[52] Jintao Zhang, Zhuo Wang, and Naveen Verma. "In-memory computation of a machine-learning classifier in a standard 6T SRAM array." *IEEE Journal of Solid-State Circuits* 52, no. 4 (2017): 915-924.

[53] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[54] Robert E. Schapire and E. Freund Boosting. "Foundations and algorithms." (2012).

[55] Yoav Freund and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of computer and system sciences* 55, no. 1 (1997): 119-139.

[56] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)." *The annals of statistics* 28, no. 2 (2000): 337-407.

[57] Thomas G. Dietterich. "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization." *Machine*

learning 40, no. 2 (2000): 139-157.

[58] John Davis, Don Plass, Paul Bunce, Y. Chan, Antonio Pelella, R. Joshi, A. Chen et al. "A 5.6 GHz 64kB dual-read data cache for the POWER6TM processor." In *2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers*, pp. 2564-2571. IEEE, 2006.

[59] Joseph K. Bradley and Robert E. Schapire. "FilterBoost: Regression and Classification on Large Datasets." In *NIPS*, pp. 185-192. 2007.

[60] 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).

[61] Shilin Zhu, Xin Dong, and Hao Su. "Binary ensemble neural network: More bits per network or more networks per bit?." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4923-4932. 2019.

[62] 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, Cham, 2016.

[63] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. "Convolutional neural networks using logarithmic data representation." *arXiv preprint arXiv:1603.01025* (2016).

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

[65] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K. Esser, and Dharmendra Modha. "Deep neural networks are robust to weight binarization and other non-linear distortions." *arXiv preprint arXiv:1606.01981* (2016).

[66] Adrian Bulat, and Georgios Tzimiropoulos. "Xnor-net++: Improved binary neural networks." *arXiv preprint arXiv:1909.13863* (2019).

[67] Taylor Simons, and Dah-Jye Lee. "A review of binarized neural networks." *Electronics* 8, no. 6 (2019): 661.

[68] Xiaofan Lin, Cong Zhao, and Wei Pan. "Towards accurate binary convolutional neural network." *arXiv preprint arXiv:1711.11294* (2017).

[69] Leo Breiman. "Bagging predictors." *Machine learning* 24, no. 2 (1996): 123-140.

[70] R. Schapire and Y. Freund. "A decision-theoretic generalization of on-line learning and an application to boosting." In *Second European Conference on Computational Learning Theory*, pp. 23-37. 1995.

[71] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression." In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535-541. 2006.

[72] Vadim Lebedev, and Victor Lempitsky. "Speeding-up convolutional neural networks: A survey." *Bulletin of the Polish Academy of Sciences. Technical Sciences* 66, no. 6 (2018).

[73] Fengfu Li, Bo Zhang, and Bin Liu. "Ternary weight networks." *arXiv preprint arXiv:1605.04711* (2016).

[74] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. "Binaryrelax: A relaxation approach for training deep neural networks with quantized weights." *SIAM Journal on Imaging Sciences* 11, no. 4 (2018): 2205-2223.

[75] Penghang Yin, Shuai Zhang, Yingyong Qi, and Jack Xin. "Quantization and training of low bit-width convolutional neural networks for object detection." *arXiv preprint arXiv:1612.06052* (2016).