A Column Reduction Technique for an In-Memory Machine-Learning Classifier

Jiazhen Xi and Hiroyuki Yamauchi

***Abstract*—A column reduction technique for an in-memory machine-learning classifier in 6T SRAM cells is discussed in this paper, 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 downsampled 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 a similar accuracy to ensure that the corresponding part of the model can be reduced to achieve better energy saving.**

***Index Terms*—In-memory, machine learning, column reduction, error-adaptive classifier boosting.**

I. Introduction

Scaling of memory technology increases the crisis of operating power and hardware variability in fields like the Internet of Things and sensor networks, where the constraints of energy cost and hardware reliability are most rigorous. To overcome such challenges, directions of in-memory computation, where the computation is performed within the memory (SRAM bit-cell), are becoming more highlighted recently.

Fig. 1 shows the concepts of the in-memory computation process and conventional compute-out-of-memory (out-memory) process. The goal of the process in Fig. 1 is to recognize images with a size of 128 × 128 pixels. The conventional out-memory process first serially accesses data from the memory and then performs a 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 the frequent access and only costs 1 nJ per image inference.

Manuscript received January 12, 2018; revised March 10, 2018.

J. Xi is with the Department of Computer Science and Engineering, Fukuoka Institute of Technology, Fukuoka, Japan and the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing, China (e-mail: mfm16104@bene.fit.ac.jp).

H. Yamauchi is with the Department of Computer Science and Engineering, Fukuoka Institute of Technology, Fukuoka (e-mail: yamauchi@fit.ac.jp).

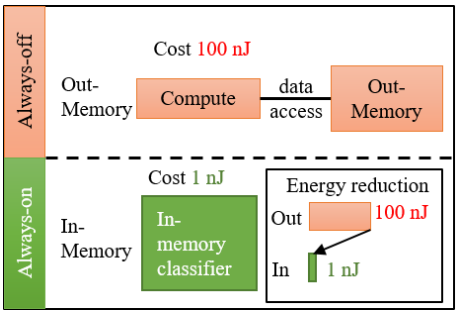

Fig. 1. Comparison between the out-memory process and in-memory process.

In recent years, there have been noteworthy studies in in-memory computation related fields. In 2014, Kang *et al.* [1] proposed the concept of in-memory computation, which is performed inside the memory (SRAM cells). A behavioral model for circuit nonlinearity was presented to research its impact. The demonstrated system used a standard memory and parallel structure device and achieved 63% energy saving compared with the conventional system for pictures of size 256 × 256 pixels. In 2015, Wang *et al.* [2] proposed an error-adaptive classifier boosting (EACB) algorithm to train an error-tolerant system with a performance of 65× reduction of the required memory and 10× energy saving, compared with conventional boosting algorithms (i.e., AdaBoost and FilterBoost). Kang *et al.* [3] proposed a novel VLSI design specialized for convolutional neural networks by employing in-memory computation. The demonstrated system reached an accuracy of 99% on the MNIST database, achieving an energy delay product reduced by 24.5×, energy cost reduced by 5.0×, and a 4.9× higher throughput compared with the conventional systems. In 2016, Rieutort-Louis *et al.* [4] proposed a large-area system for image sensing and detection, integrating sensors and thin-film transistor circuits to achieve detection and classification for images from data of sensors. The proposed system reached an accuracy performance of more than 85%–95%, which is at a similar level of an ideal support vector machine classifier. The proposed system reduced the total signal numbers by 3.5×–9× from 36 sensors in the large-area electronics domain by detecting the shapes of images and employing the EACB algorithm. Zhang et al. [5] proposed an in-memory machine-learning classifier where the computation is embedded into standard 6T SRAM cells and the demonstrated digit recognition system for the MNIST database achieved an accuracy of over 90% with features reduced to 9 × 9 by employing 18 iterations of EACB and achieving 113× energy saving compared with the conventional ideal digital SRAM cell system.

However, only a few studies focus on the reduction of model complexity to reach a higher level of energy saving,

doi: 10.18178/ijmlc.2018.8.2.675

which is the exact purpose of this work. The proposed technique is mainly applied to the in-memory machine-learning classifier system implemented in 6T SRAM cells employing the EACB algorithm [2].

II. System Summary

A. System Diagram

Our system uses the MNIST dataset [6] of 28 ×28 images of 0–9 digits, and the evaluation is mainly based on the final recognition accuracy and corresponding column number of the boosted model, which reflects the energy costs of the system. The diagram of the system is shown in Fig. 2.

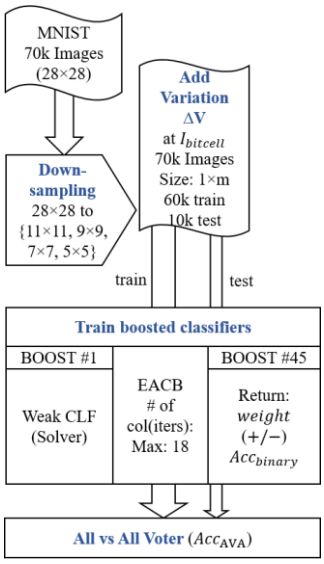

Fig. 2. System diagram.

First, the original images are downsampled to several smaller sizes, 11 ×11, 9 ×9, 7 ×7, and 5 ×5, applicable for the standard 6T SRAM cell structure. Then, they are flattened to 1 × *m* and transformed to a word-line voltage vector connected to SRAM bit-cells, wherein the time-dependent variability with different strengths (0–200 mV) is added to simulate the SRAM bit-cell (see Fig. 3). The time-dependent variability occurs in SRAM bit-cells, and we assume that it is approximately normally distributed. Its strength is described by Δ*V*, the standard deviation of variation, in the following sections of this paper.

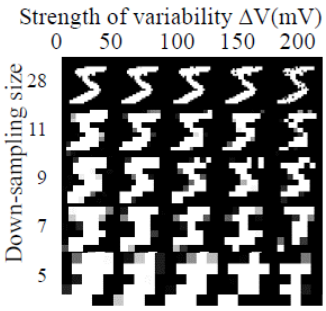

Fig. 3. The effect of downsampling and time-dependent variability.

B. In-Memory Model Diagram

Weak basic linear classifiers are restricted to 1 bit to be suitable for the structure of bit-cells, employing a constrained resolution regression algorithm [7].

Their weights are trained from *t* (the maximum is selected to be 18 in simulation) iterations of the EACB. All *t* trained weak classifiers compose one boosted strong classifier. All pairs of digits are represented by 45 boosted classifiers which compose an EACB model. The in-memory model diagram is shown in Fig. 4.

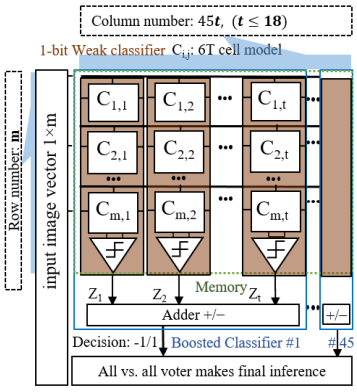

Fig. 4. In-memory model diagram.

C. Inference Flow

The inference process comprises three steps:

- Step 1: all weak classifiers of a boosted classifier make −1/1 decisions for the input image.
- Step 2: weighted decisions are added to be −1/1 as the decision of a boosted classifier.
- Step 3: applying all-versus-all (AVA) voting, 45 decisions of boosted classifiers make a 10-class inference of 0–9 digits.

III. Column Reduction Technique

A. Baseline Performance

Using the same EACB algorithm and binary linear model setup, the baseline model employs *m* rows and 45 ×*t* columns, where *m* is dependent on the image's size and *t* is represented for EACB iterations for each boosted classifier. The following demonstration in this paper takes one dataset of *D*(11,0), which represents 11 × 11 pixel images and time-dependent variability of Δ*V* = 0, as an example for simplicity. For *D*(11,0), the baseline model reaches an accuracy of 93.5% with 585 columns (*t* = 13) and 121 rows. Fig. 5 shows the accuracy performance by the EACB iteration *t*.

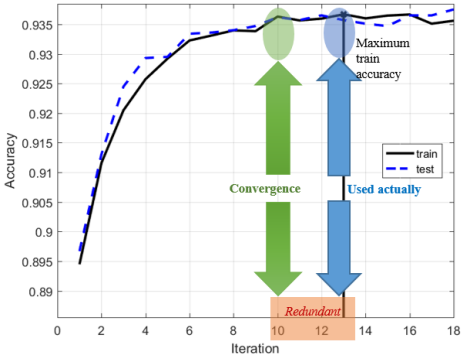

Fig. 5. Baseline accuracy by iterations.

To avoid underfitting and accuracy loss, *t* = 13 with

maximum train accuracy is selected to be the baseline EACB iteration. As Fig. 5 shows, the convergence comes quite earlier than the maximum point, making it possible to reduce the redundant columns between the convergence point and the actual selected point.

B. Pruning Columns for Different Boosted Classifiers

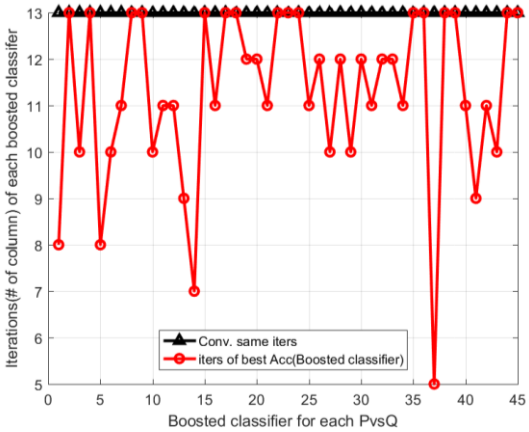

Fig. 6. Conventional same iterations versus iterations of the best accuracy.

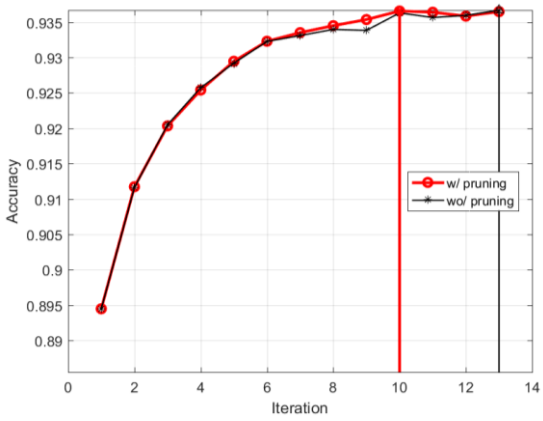

Fig. 7. Accuracy with pruning versus accuracy without pruning.

Within the AVA voting strategy, the original 10-class classification problem is divided into 45 binary classification tasks. However, the difficulties for different binary tasks are different and the best EACB iteration for each boosted classifier also varies. Thus, the first step to reduce the columns is by pruning the columns for different boosted classifiers. Figure 6 shows the comparison of columns of fixed same iterations and columns with pruning for $D(11,0)$.

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

A comparison of the accuracy of the model with or without pruning by iteration is shown in Fig. 7. Both models have the same level of accuracy and the model with pruning gets earlier convergence compared with the model without pruning. For $D(11,0)$, this reduces 206 columns from 585–379.

C. Reducing Columns by Greedy Search

The proposed greedy search method is based on an assumption that if there is an optimum point between t = 1(45 columns maximum) and the current pruned columns (i.e., 379 columns), the total number of columns can be optimized. For $D(11,0)$, the greedy search starts from 45 columns and ends at 379 columns.

The basic idea of the proposed greedy search is to iteratively add a column to a boosted classifier, which brings about the best accuracy gain until its column number reaches the result with pruning.

For $D(11,0)$, first, initialize the number of columns of each boosted classifier to 1. Then, calculate each AVA accuracy if only one column is added to each boosted classifier. Only add one column to the boosted classifier with the best AVA accuracy. Iteratively repeat this process until the end, and then select the earliest iteration with a similar accuracy as the final result. The pseudocode is shown in Fig. 8.

```
// Greedy search
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
20 endwhile
```

Fig. 8. Greedy search column reduction method.

For $D(11,0)$, this reduces 11 columns from 379–368; the result is shown in Fig. 9.

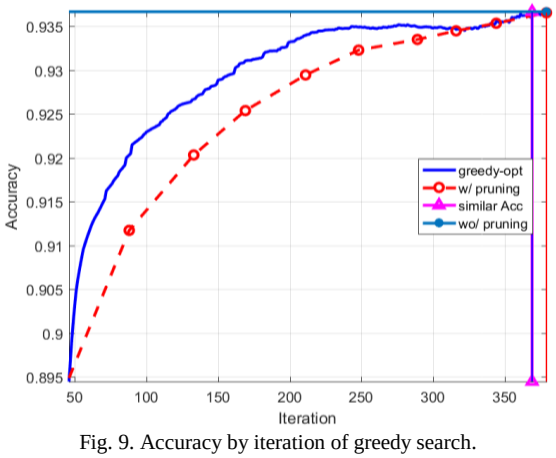

Fig. 9. Accuracy by iteration of greedy search.

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 within less iterations, which achieves the reduction for the columns. The curve labeled with "w/ pruning" and "wo/ pruning" is the same as the result with and without pruning. It shows that the accuracy path of the greedy search is nearer to the top-left corner within most of the iterations, meaning that with the same columns, it finds a column setting with a higher accuracy than the pruned result by the greedy search. Figure 10 shows the result of column number of each boosted classifier by greedy search labeled as the same rule for $D(11,0)$.

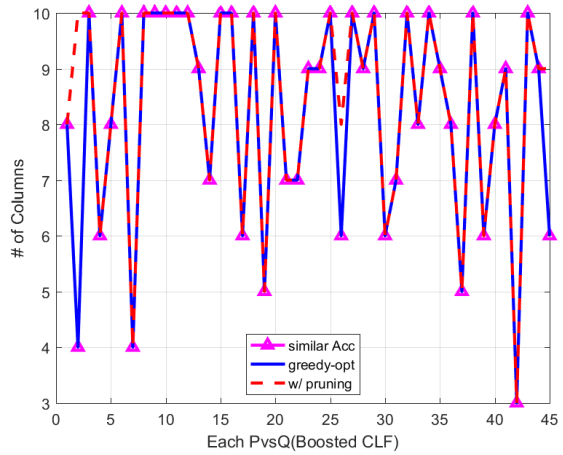

Fig. 10. Column number for each boosted classifier by greedy search.

```
// Greedy search fast version
input: col_prn output: col_grdfast, acc_final
  // init
1 col_grdfast, acc_final, iter = init(col_prn)
  // iter
7 while sum(col_grdfast) < sum(col_prn)
  // one step greedy
8  col_grdfast, acc_final, iter, i_max = oneStepGrd()
  // end of one step greedy
  // fast
20   while col_grdfast[i_max] < col_prn[i_max]
21      col_grdfast[i_max] = col_grdfast[i_max] + 1
22      acc_fast = calcAccAVA(col_grdfast)
23        if acc_fast > acc_final[iter]
24        iter = iter + 1
25        acc_final[iter] = acc_fast
26      else
27        col_grdfast[i_max] = col_grdfast[i_max] – 1
28        break
29      endif
30   endwhile
31 endwhile
```

Fig. 11. Greedy search (fast) column reduction method.

However, there is a problem with the method: it calculates the AVA accuracy 45 times for each step, which brings about quite large time consumption. To solve this, we can modify the proposed greedy search method to a fast version as in the pseudocode shown in Fig. 11.

For simplicity here, lines 1–6 of the pseudocode of the greedy search are abstracted as procedure "init()" which is exactly the same in the three proposed methods and will be omitted in the following pseudocodes. Similarly, lines 8–19 of the pseudocode of the greedy search are abstracted as procedure "oneStepGrd()" representing one single step of the greedy search, which is employed again in the fast version of the greedy search at line 8. The modification part as well as the logic of the fast version of the greedy search is described at lines 20–30.

We observed that column updating tends to be fixed at the same boosted classifier within a few iterations. So, in each iteration after the column is updated, just add one column again to the same boosted classifier and calculate the AVA accuracy and repeat until the accuracy is no longer larger than the last step. For $D(11,0)$, the fast version shows a similar result to the normal greedy search (see Figs. 12 and 13), while saving over 50% of the time consumed on average.

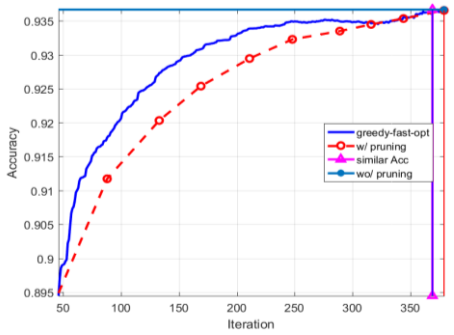

Fig. 12. Accuracy by iteration of greedy search (fast).

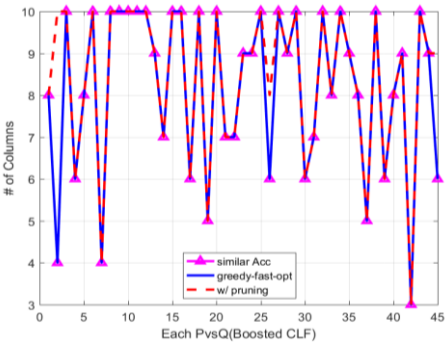

Fig. 13. Column number for each boosted classifier by greedy search (fast).

D. Reducing Columns by Worst-Care

Different binary classification tasks (P versus Q) bring about different classification difficulties, leading to variations in the accuracy for each binary task. Figure 14 shows a comparison between a binary accuracy variation boxplot and final AVA accuracy (solid circles) for different down-sizes and ΔV setups.

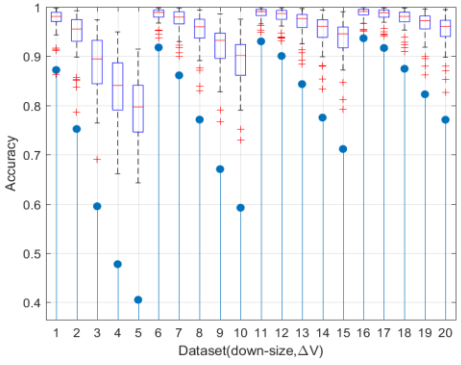

Fig. 14. Comparison between the binary accuracy variation boxplot and final AVA accuracy for different down-sizes and ΔV setups.

It is shown that in most cases, the worst binary classification accuracy of boosted classifiers is the upper bound of the final AVA accuracy. So, we assume that there are some redundant columns of boosted classifiers with an accuracy better than the worst one; based on this assumption, we propose the worst-care reduction method. Similarly, the search starts from 45 columns and ends at 379 columns. The pseudocode is shown in Fig. 15.

```
// Worst-care
input: col_prn output: col_wc, acc_final
  // init
 1 col_wc, acc_final, iter = init(col_prn)
  // iter
 7 while sum(col_wc) < sum(col_prn)
 8 iter = iter + 1
 9 acc_binary = ones(1,45)
10   for i from 1 to 45
11     if col_wc[i] < col_prn[i]
12       col_wc[i] = col_wc[i] + 1
13       acc_binary[i] = calcAccBin(col_wc)
14       col_wc[i] = col_wc[i] – 1
15     endif
16   endfor
17   i_min = argmin(acc_binary)
18   col_wc[i_min] = col_wc[i_min] + 1
19   acc_final[iter] = calcAccAVA(col_wc)
20 endwhile
```

Fig. 15. Worst-care column reduction method.

The basic idea of the proposed worst-care method is to iteratively add a column to a boosted classifier whose train binary accuracy is the worst until its column number reaches the result with pruning. For $D(11,0)$, first, initialize the number of columns of each boosted classifier to 1. Then, compare the train accuracy of each boosted classifier. Only add one column to the boosted classifier with the worst train accuracy. Iteratively repeat this process until the end, and then select the earliest iteration with a similar accuracy as the final result. For $D(11,0)$, this reduces 14 columns from 379–365. The result of the worst-care method for $D(11,0)$ is shown in Fig. 16.

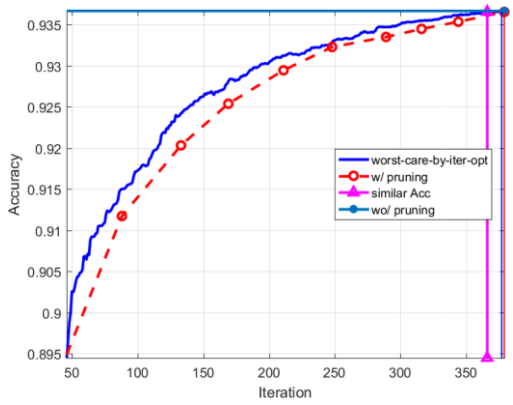

Fig. 16. Accuracy by iteration of worst-care.

Similarly, the curve labeled with "worst-care-by-iter" represents the accuracy changing with the total column number (iteration of worst-care), and it reaches the same accuracy ("similar Acc") as the result with pruning within less iterations which achieves the reduction for the columns. The curve labeled with "w/ pruning" and "wo/ pruning" is also the result with and without pruning.

Figure 17 shows the result of column number of each boosted classifier by worst-care labeled as the same rule for $D(11,0)$.

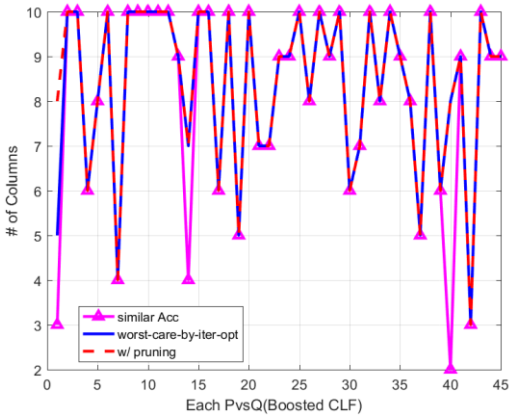

Fig. 17. Column number for each boosted classifier by worst-care.

IV. Simulation Result

Here, we apply the proposed column reduction methods to different downsampling sizes and time-dependent variability setups, namely, sizes of 11 ×11, 9 ×9, 7 ×7, and 5 ×5 and ΔV = 0 mV, 50 mV, 100 mV, 150 mV, and 200 mV. The simulation result mainly focuses on the reduced column number as well as the accuracy difference compared with the performance of the baseline model after applying the proposed column reduction methods. The overall result is shown in Fig. 18.

In Fig. 18, the figure at the top shows the comparison of the AVA accuracy and the figure at the bottom shows the total used column number. As labeled in the legend, each bar from left to right represents the result of the baseline model, pruned model, greedy search applied model, fast greedy search applied model, and worst-care applied model. This shows that after applying the proposed methods, the column number is reduced for different setups with a similar accuracy compared with the baseline.

Fig. 18. Overall result of the comparison of the test accuracy and column numbers.

Fig. 19 is the boxplot of the test accuracy difference and

reduced column numbers compared with the baseline model. Five entries, namely, Baseline, Pruned, Greedy Search, Greedy Search fast version, and Worst-care, are compared.

Fig. 19 also shows that the accuracy of different models is in a similar range and the model applied worst-care achieves the least accuracy loss, within 0.4%, compared with the baseline, and achieves the most robust result of column reduction. Greedy search and fast greedy search reach similar accuracy results, whereas the fast version achieves better results on column reduction.

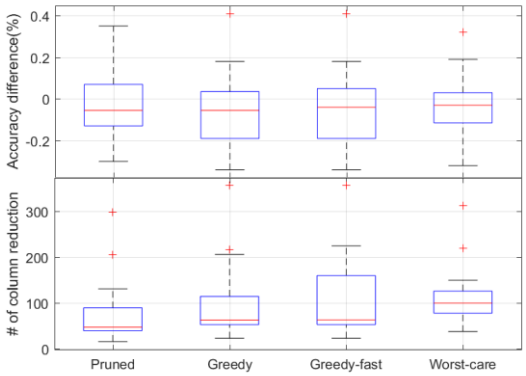

Fig. 19. Boxplot of the test accuracy difference and reduced column numbers compared with the baseline model.

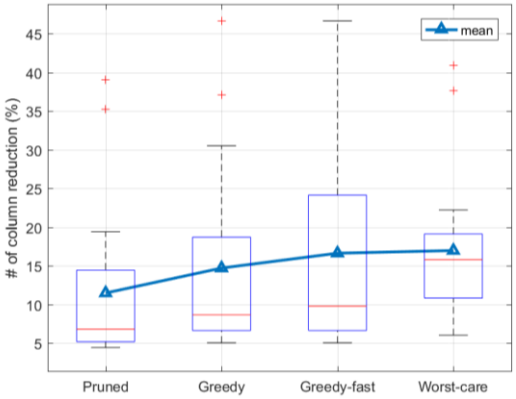

Fig. 20. Boxplot of the column number reduction percentage compared with the baseline model.

Fig. 20 is the boxplot of the column number reduction percentage compared with the baseline model, and the line marked triangle is the mean value of each result.

Fig. 20 also shows that the pruning process contributes 11.50% column reduction on average, and greedy search, fast greedy search, and worst-care contribute extra 3.23%, 5.14%, and 5.49% column reduction on average of the total number of columns of the baseline model.

V. Conclusion

In this paper, a column number reduction technique for an in-memory machine-learning classifier based on a 6T SRAM cell structure was proposed. In the first stage of pruning, the column number of each boosted classifier is pruned, achieving 11.50% column reduction. In the second stage of reduction, three methods were proposed and 3.23%, 5.14%, and 5.49% of the columns were reduced on average, achieving the purpose of column reduction for higher-level energy saving with the accuracy at a similar level.

References

[1] M. Kang, S. K. Gonugondla, M. S. Keel, and N. R. Shanbhag, "An energy-efficient memory-based high-throughput vlsi architecture for convolutional networks," in *Proc. International Conf. on Acoustics, Speech and Signal Processing*, 2015, pp. 1037-1041.

[2] Z. Wang, R. E. Schapire, and N. Verma, "Error adaptive classifier boosting (EACB): leveraging data-driven training towards hardware resilience for signal inference," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 62, no. 4, pp. 1136-1145, April 2015.

[3] M. Kang, M. S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, "An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in sram," in *Proc. IEEE International Conf. on Acoustics, Speech and Signal Processing*, 2014, pp. 8326-8330.

[4] W. Rieutort-Louis, T. Moy, Z. Wang, S. Wagner, J. C. Stum, and N. Verma, "A large-area image sensing and detection system based on embedded thin-film classifiers," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 281-290, Jan. 2016.

[5] J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in *Proc. IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, 2016, pp. 1-2.

[6] Y. LeCun, and C. Cortes, "MNIST handwritten digit database," AT&T Labs, 2010.

[7] Z. Wang and N. Verma, "A low-energy machine-learning classifier based on clocked comparators for direct inference on analog sensors," *IEEE Trans. on Circuits and Systems I: Regular Papers*.

Jiazhen Xi is a master student in the Department of Computer Science and Engineering, Fukuoka Institute of Technology (FIT). He is pursuing a double master degree offered by FIT and Nanjing University of Science and Technology (NUST) together.

He received his bachelor degree in NUST in 2015. Currently, his research field includes in-memory machine learning systems and related cost reduction techniques.

His interests of research are machine learning, signal processing, signal inference, compressed sensing, application of machine learning in hardware domain, especially high applicable low energy cost application and Internet of Things.

Hiroyuki Yamauchi received the Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1997. In 1985 he joined the Semiconductor Research Center, Panasonic, Osaka, Japan. From 1985 to 1987 he had worked on the research of the submicron MOS FET model-parameter extraction for the circuit simulation and the research of the sensitivity of the scaled sense amplifier for ultrahigh-density DRAM's. From 1988 to 1994, he was engaged in research and development of 16-Mb CMOS DRAM's including the battery-operated high-speed 16 Mbit CMOS DRAM and the ultra low-power, three times longer, self-refresh DRAM. He also invented the charge-recycling bus architecture in 1994 which was used as a reference to develop the technologies for NVIDIA GPU bus designs and low-voltage operated high-speed VLSI's, including 0.5V/100MHz-operated SRAM and Gate-Over-Driving CMOS architecture. After experienced general manager for development of various embedded memories, eSRAM, eDRAM, eFlash, eFeRAM, and eReRAM for system LSI in Panasonic, he has moved to Fukuoka Institute of Technology as a professor since 2005. His current interests are focused on study for variation tolerant memory circuit designs and power aware machine-learning model for in-memory artificial intelligence (AI) inference system of everything era. He holds 212 Patents including 87 U.S. Patents and has presented over 70 journal papers and proceedings of international conferences including 11 for ISSCC and 11 for Symposium on VLSI Circuits. Dr. Yamauchi received the 1996 Remarkable Invention Award from Science and Technology Agency of Japanese government and the highest ISOCC2008 and ISOCC2013 Best Paper Award.