ECE544NA Final Project: Robust Machine Learning Hardware via Classifier Ensemble

Sai Zhang, *szhang12@illinois.edu*
Dept. of Electr. & Comput. Eng., Univ. of Illinois at Urbana-Champaign, Urbana, IL, USA

***Abstract*—In this paper, we propose to use classifier ensemble (CE) as a method to enhance the robustness of machine learning (ML) kernels in presence of hardware error. Different ensemble methods (Bagging and Adaboost) are explored with decision tree (C4.5) and artificial neural network (ANN) as base classifiers. Simulation results show that ANN is inherently tolerant to hardware errors with up to 10% hardware error rate. With simple majority voting scheme, CE is able to effectively reduce the classification error rate for almost all tested data sets, with maximum test error reduction of 48%. For tree ensemble, Adaboost with decision stump as weak learner gives best results; while for ANN, bagging and boosting outperform each other depending on data set.**

I. Introduction

This project is motivated by recently works appeared in machine learning (ML) on silicon [1]. There is a growing interest in VLSI and circuit area to efficiently bring ML kernels onto silicon, largely due to the performance and power limitation of software only approaches. On the other hand, in deep sub-micro era, devices exhibit statistical behaviors which degrade the system reliability. To reduce the energy consumption of circuits, sub/near threshold computing has been proposed to offer 10X power reduction at the expense of large delay variation as shown in Fig. 1. To enhance the reliability and robustness of ML kernels on silicon, Verma et al. propose to utilize ML algorithm to efficiently learn the error behavior and make correct prediction/classification in presence of hardware errors. Kim [2] et al. shows that some ML algorithm inherently has tolerance to hardware errors. These work motivate us to look into methods to effectively enhance the robustness of ML kernels in presence of hardware errors.

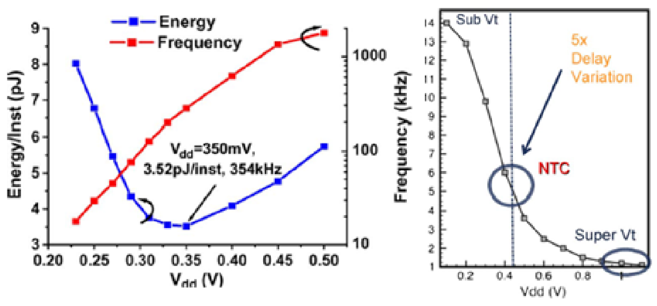


Fig. 1. With reduced supply voltage, energy efficiency is improved but delay variation is larger.

Classifier ensemble (CE, also referred to as Multiple Classifier System) have been employed to enhance the performance of single classifier system [3], [4]. As summarized in [5]–[8], popular ensemble methods include: Bagging, Adaboost, Bayesian voting, Random forest, Rotation forest, Error correcting output coding, etc. In terms of decision combining, [9], [10] summarizes typical classifier fusion methods.

Therefore, in this paper, we explore the idea of using CE to enhance the robustness of ML kernels. Different ensemble methods (Bagging and Adaboost) are explored with decision tree (C4.5) and artificial neural network (ANN) as base classifiers. Simulation results show that ANN is inherently tolerant to hardware errors with up to 10% hardware error rate. With simple majority voting scheme, CE is able to effectively reduce the classification error rate for almost all tested data sets, with maximum test error reduction of 48%. For tree ensemble, Adaboost with decision stump as weak learner gives best results; while for ANN, bagging and boosting outperform each other depending on data set. The rest of the report is organized as follows: section 2 gives background of the CE methods we use in the project, section 3 presents error model, and simulation setup; section 4 presents simulation results with conclusion provided in section 5.

II. Background

A. Ensemble Methods

We give a brief overview of methods to construct classifier ensembles.

1) Average Bayes Classifier: In the average Bayes setting, every classifier output is interpreted as a posterior probability of class membership, condition on the input and the hypothesis.

$$h(\mathbf{x}) = P(f(x) = y|\mathbf{x}, h) \tag{1}$$

Here we assume the hypothesis follow a distribution in the hypothesis space H. If the hypothesis space is small, it is possible to enumerate all possible classifiers ($h(\boldsymbol{x})$). The final output is the average of these hypothesis, weighted by the posterior probability of $h(\boldsymbol{x})$, as shown in the following equation:

$$P(f(\mathbf{x}) = y|S, \mathbf{x}) = \sum_{h \in \mathbf{H}} h(\mathbf{x}) P(h|S) \tag{2}$$

Using Bayes rule, we can write the posterior probability $P(h|S)$ as:

$$P(h|S) \propto P(S|h)P(h) \tag{3}$$

The practical difficulty in implementing this method lies in that when the hypothesis space is large, it is difficult to enumerate all $h(\boldsymbol{x})$, also it is not always possible to know the prior of each of the hypothesis. Fig. 2 provides a illustration of the algorithm. The true mapping which the algorithm tries to learn is denoted as $f(\boldsymbol{x})$. To construct the ensemble, all hypothesis $h(\boldsymbol{x})$ in the space H is learned, and the final mapping is a weighted average based on (2).

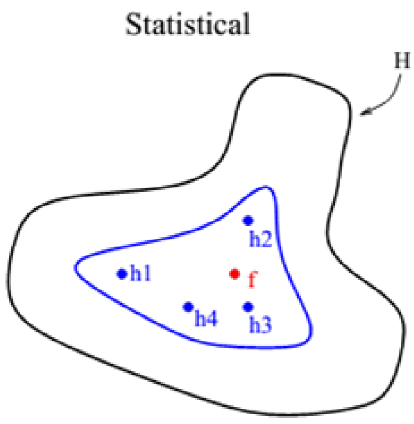


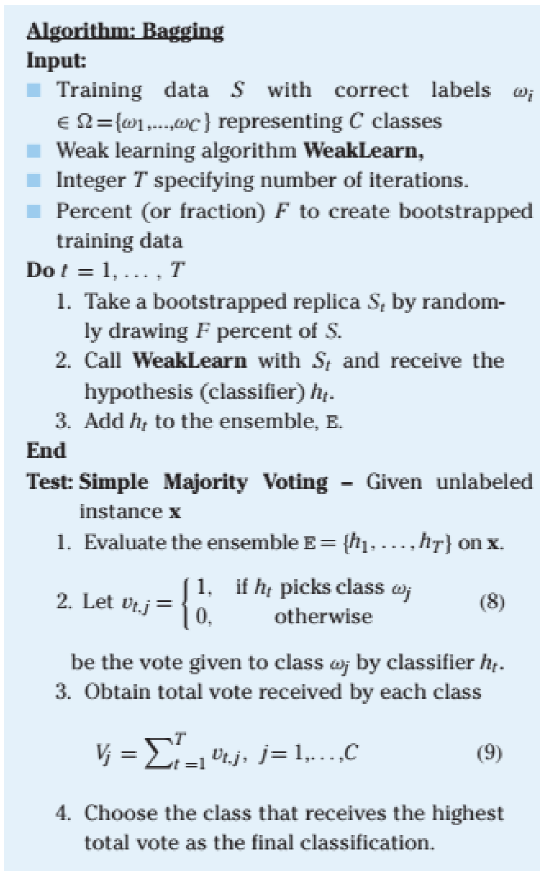
Fig. 2. Average Bayes Classifier

2) Bagging: Bootstrap aggregating is a popular method to construct ensembles. The basic idea is to first generate many training sets from original training samples by random sampling with replacement, then train multiple classifiers, each based on one of the training sets, as shown in Fig. 3. By random sampling with replacement, each training set contains on average 63.3% of the original training set. The final decision is made by taking the majority voting of individual weak classifiers. Bagging has been shown to improve the performance of unstable classifiers, such as neural networks, decision trees etc.

3) Adaptive Boosting: Adaboost is another popular method for ensemble generation. The basic idea can be depicted in the Fig. 4. The training has T iterations; each iteration will produce a new weak classifier. The basic idea is that after each iteration step, the examples are re-weighted so that miss-classified examples get higher weight. In the subsequent iteration step, the training process will focus more on classifying the miss-classified samples from previous iteration.

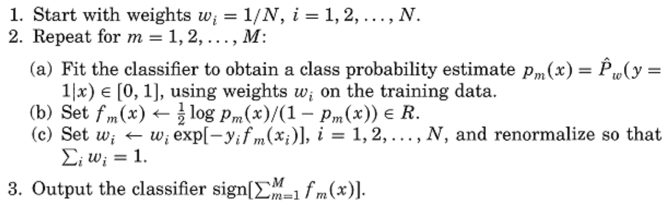
Fig. 5 gives the algorithm flow of Adaboost. Note that the basic idea of Adaboost is very similar with Bagging, with two notable differences: 1) in generating different training sample, Bagging uses random sample with replacement while Adaboost uses the accuracy of previous classifiers to guide the selection of subsequent training samples; and 2) Bagging uses simple majority voting to generate decision while Adaboost uses weighted average.

It can be shown that the selection of training sample distribution and weights in Adaboost will minimize an exponential loss function defined as



```
Algorithm: Bagging
Input:
  ■ Training data $S$ with correct labels $\omega_i \in \Omega = \{\omega_1,\ldots,\omega_C\}$ representing $C$ classes
  ■ Weak learning algorithm WeakLearn,
  ■ Integer $T$ specifying number of iterations.
  ■ Percent (or fraction) $F$ to create bootstrapped training data
Do $t = 1, \ldots, T$
  1. Take a bootstrapped replica $S_t$ by randomly drawing $F$ percent of $S$.
  2. Call WeakLearn with $S_t$ and receive the hypothesis (classifier) $h_t$.
  3. Add $h_t$ to the ensemble, $E$.
End
Test: Simple Majority Voting – Given unlabeled instance x
  1. Evaluate the ensemble $E = \{h_1, \ldots, h_T\}$ on x.
  2. Let $v_{t,j} = \begin{cases} 1, & \text{if } h_t \text{ picks class } \omega_j \\ 0, & \text{otherwise} \end{cases}$   (8)
     be the vote given to class $\omega_j$ by classifier $h_t$.
  3. Obtain total vote received by each class
     $V_j = \sum_{t=1}^{T} v_{t,j}, \; j = 1,\ldots,C$   (9)
  4. Choose the class that receives the highest total vote as the final classification.
```

Fig. 3. Bagging



```
1. Start with weights $w_i = 1/N$, $i = 1, 2, \ldots, N$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the classifier to obtain a class probability estimate $p_m(x) = \hat{P}_w(y = 1|x) \in [0, 1]$, using weights $w_i$ on the training data.
   (b) Set $f_m(x) \leftarrow \frac{1}{2} \log p_m(x)/(1 - p_m(x)) \in R$.
   (c) Set $w_i \leftarrow w_i \exp[-y_i f_m(x_i)]$, $i = 1, 2, \ldots, N$, and renormalize so that $\sum_i w_i = 1$.
3. Output the classifier $\text{sign}[\sum_{m=1}^{M} f_m(x)]$.
```

Fig. 5. Adaboost algorithm

$$L(f(x), y) = \sum_{i}^{N} \exp(-y_i f(\mathbf{x}_i)) \tag{4}$$

Proof:
We are trying to find a classifier of the form:

$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_i f_{m-1}(\mathbf{x})$$

to minimize loss in (4). At step m, we have

$$H_m(\mathbf{x}) = H_{m-1}(\mathbf{x}) + \alpha_m f_m(\mathbf{x})$$

The minimization problem can be formulated as:

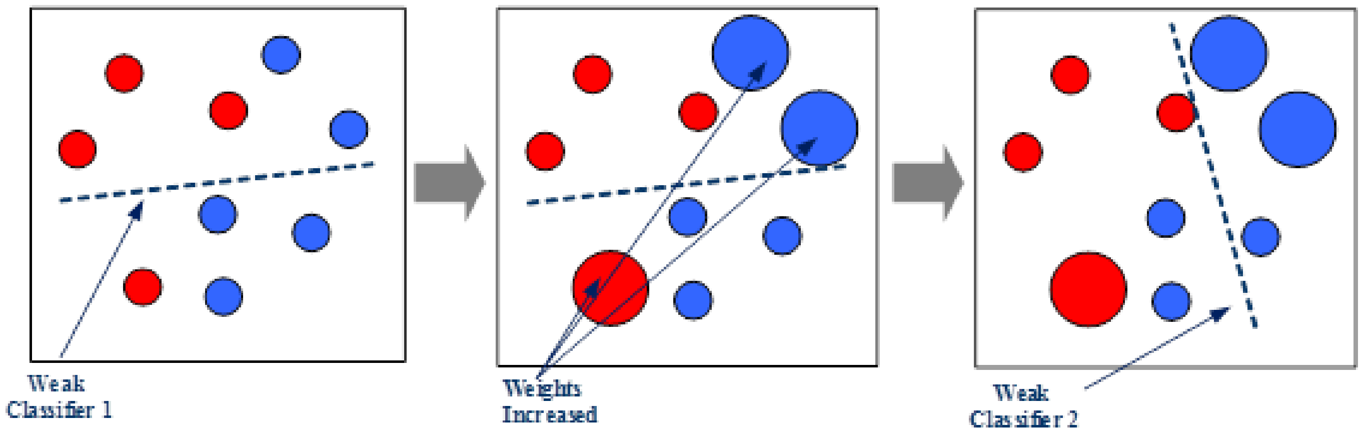


Fig. 4. Adaboost

$$[\alpha_m, f_m] = \arg\min_{\alpha,f} \sum_i^N \exp(-y_i(H_{m-1}(\mathbf{x}) + \alpha f(\mathbf{x})))$$
$$\Leftrightarrow$$
$$[\alpha_m, f_m] =$$
$$\arg\min_{\alpha,f}\{\sum_i^N \exp(-y_i H_{m-1}(\mathbf{x})) \cdot \exp(\alpha) 1[y_i \neq f(\mathbf{x})]$$
$$+ \sum_i^N \exp(-y_i H_{m-1}(\mathbf{x})) \cdot \exp(-\alpha) 1[y_i = f(\mathbf{x})]\}$$
$$\Leftrightarrow$$
$$[\alpha_m, f_m] =$$
$$\arg\min_{\alpha,f}\{\exp(\alpha) \sum_i^N \exp(-y_i H_{m-1}(\mathbf{x})) \cdot 1[y_i \neq f(\mathbf{x})]$$
$$+ \exp(-\alpha)(\sum_i^N \exp(-y_i H_{m-1}(\mathbf{x}))$$
$$- \sum \exp(-y_i H_{m-1}(\mathbf{x})) \cdot \exp(-\alpha) 1[y_i = f(\mathbf{x})])\}$$

This is equivalent to

$$[\alpha_m, f_m] =$$
$$\arg\min_{\alpha,f}\{\exp(\alpha) \sum_i^N w_{m-1}(i) \cdot 1[y_i \neq f(\mathbf{x})]$$
$$+ \exp(-\alpha)(1 - w_{m-1}(i)) \cdot \exp(-\alpha) 1[y_i = f(\mathbf{x})])\}$$

where $w_{m-1}(i) = \frac{\exp(-y_i H_{m-1}(\mathbf{x}))}{\sum_i^N \exp(-y_i H_{m-1}(\mathbf{x}))}$, we can further simplify the optimization into:

$$[\alpha_m, f_m] = \arg\min_{\alpha,f}\{\exp(-\alpha) +$$
$$[\exp(\alpha) - \exp(-\alpha)] \sum_i^N w_{m-1}(i) \cdot 1[y_i \neq f(\mathbf{x})]\}$$

The optimal α and f can be solved by setting the derivative with respect to them to 0, we can thus obtain the optimal α_m, f_m as:

$$f_m = \arg\min_f \sum_i^N w_{m-1}(i) \cdot 1[y_i \neq f(\mathbf{x_i})]$$
$$\alpha_m = \frac{1}{2} \log(\frac{1 - \varepsilon_m}{\varepsilon_m})$$

where $\varepsilon_m = \sum_i^N w_{m-1}(i) \cdot 1[y_i \neq f(\mathbf{x})]$. Therefore, we can see that in Adaboost, the distribution and weight are selected such that the loss function in (4) is minimized.

4) Injection Randomness: The final method to construct ensemble is by injecting randomness into the classifier. For good ensemble performance, the weak classifier used to construct the ensemble need to be unstable. Good unstable classifiers are trees or neural networks. In the random injecting method, the ensemble is constructed by simply changing the initial the weights of NN, or choosing randomly the features used to split the trees.

B. Decision Combination

The second important aspect of CE is to decide how to combine the output of each classifier to obtain the final decision. Many decision combination methods have been proposed, as shown in Fig. 6. If the output of each classifier is not a hard decision but a continuous measure. We can use averaging method to get the ensemble output, popular averaging method includes mean, median, weighted mean etc. If the output of classifier is a discrete number, voting methods can be used. The simplest voting methods is majority voting, which takes the final output as the class most of the classifiers agree on. Weighted average method can also be used to weight the decision of each classifier based on their accuracy measure as in the case of Adaboost.

III. Methodology

A. Hardware error simulation

Error injection is performed in simulation level since we do not have realistic hardware realization of the classifiers. The error injection presents difficulty due to at least two

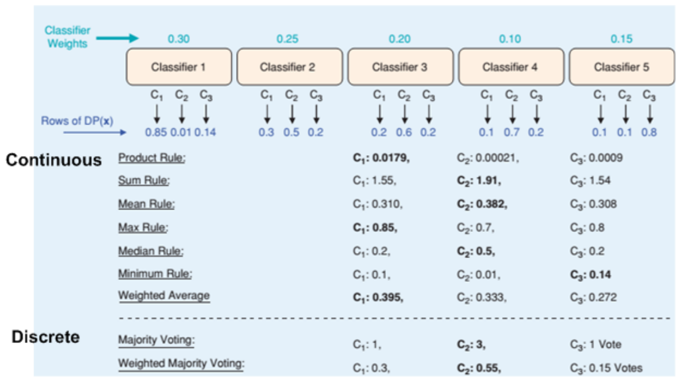

Classifier Weights
0.30
0.25
0.20
0.10
0.15
Classifier 1
Classifier 2
Classifier 3
Classifier 4
Classifier 5
Rows of DP(x)
0.85 0.01 0.14
0.3 0.5 0.2
0.2 0.6 0.2
0.1 0.7 0.2
0.1 0.1 0.8
Continuous
Product Rule: C1: 0.0179, C2: 0.00021, C3: 0.0009
Sum Rule: C1: 1.55, C2: 1.91, C3: 1.54
Mean Rule: C1: 0.310, C2: 0.382, C3: 0.308
Max Rule: C1: 0.85, C2: 0.7, C3: 0.8
Median Rule: C1: 0.2, C2: 0.5, C3: 0.2
Minimum Rule: C1: 0.1, C2: 0.01, C3: 0.14
Weighted Average C1: 0.395, C2: 0.333, C3: 0.272
Discrete
Majority Voting: C1: 1, C2: 3, C3: 1 Vote
Weighted Majority Voting: C1: 0.3, C2: 0.55, C3: 0.15 Votes

Fig. 6. Decision combination methods

requirements: 1) the mapping from input space to output error space need to have sufficient randomness, other wise it will be trivial to remove the error by adding some bias to the network output. 2) for same input pattern, it will always generate the same 'random' error pattern. To solve this issue, we use a input pattern dependent random number generator for error injection, as shown in Fig. 7. When presented with a feature vector, the random generator uses each dimension as a seed to generate a Gaussian random number; the random number from different dimension are then added and scaled to provide an output $err \sim N(0,1)$. This method ensures that for different feature vector, the generated random errors are different, while at the same time same feature vector always results in same error value.

The err is then used to inject hardware errors into trees and ANNs. For decisions tree, we generate two threshold $Threshold_1$ and $Threshold_2$ for each node, when err falls into the interval $[Threshold_1, Threshold_2]$,the decision of the node is flipped. For ANNs, similar thresholds are generated for each weight, and if the err falls into $[Threshold_1, Threshold_2]$, the weight matrix is perturbed to simulate hardware defects. We can control the error rate by controlling the thresholds, i.e. higher error rate can be achieved if $Threshold_2 - Threshold_1$ is increased.

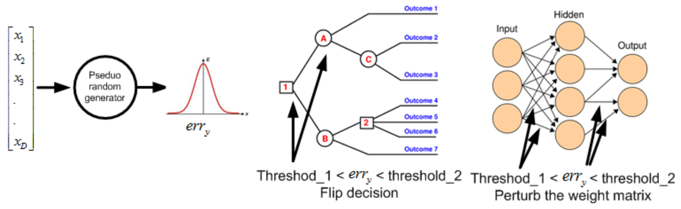


Fig. 7. Error injection methodology

B. Experiment Setup

Table I shows the experiment setup. In this project, we explore Bagging and Adaboost with ANN and C4.5 decision tree and decision stump (a decision tree with depth 1) as base classifiers. When training ANN, random initial weights are used to further introduce diversity into the ensemble. For Adaboost, we use weighted voting as shown in Fig. 5, and for

TABLE I
EXPERIMENTAL SETUP

Base Classifier	Ensemble Methods	Combination
ANN	Bagging	Majority voting
C4.5	Boosting	Weighed voting
Dec. stump	Initial weights	

Bagging, simple majority voting scheme is employed. We use 4 data sets from UCI machine learning repository. For each data set, 10 fold cross-validation methods are used to evaluate their performances.

IV. Results

Fig. 8 shows the test error rate vs. ensemble size for all evaluated ensemble methods. For Bagging with decision trees, test error rate decreases as ensemble size increases for both hardware error free (red) and hardware error injected (blue) case. However, hardware error degrade the performance of the ensemble and this degradation cannot be effectively compensated for with increased ensemble size. For Adaboost with decision tree, Fig. 8 indicates that the ensemble suffer from over-fitting. As ensemble size grows, the error rate first decreases, then starts to increase at ensemble size of 3 and 2 for error free and erroneous case. Adaboost with decision stump gives best performance in the family of tree ensembles, the error injected ensemble almost achieves the same error reduction as the error free ensemble. For ANN with bagging, both error injected and error free ensemble achieve effective reduction of test error rate with increasing ensemble size. However, for ANN with Adaboost, over-fitting starts to occur at ensemble size of 4 and 3 for error free and erroneous ensemble, respectively.

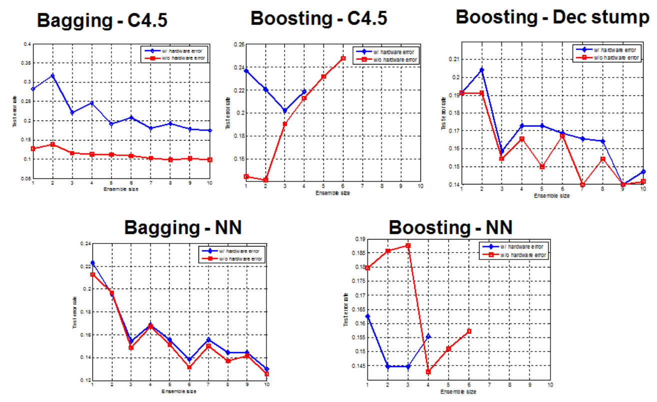


Fig. 8. Results: test error rate vs. ensemble size

Fig. 9 shows the result for 4 data set from UCI machine learning repository at different injected hardware error rate. From the figure several observation can be made:

1) In terms of inherent error tolerance, ANN is better than decision trees. This can be seen from Fig. 9 by noting that at 3% error rate, erroneous C4.5 already have severely degraded performance at data set 1 and 2, but ANN is able to maintain similar performance regardless of error injection

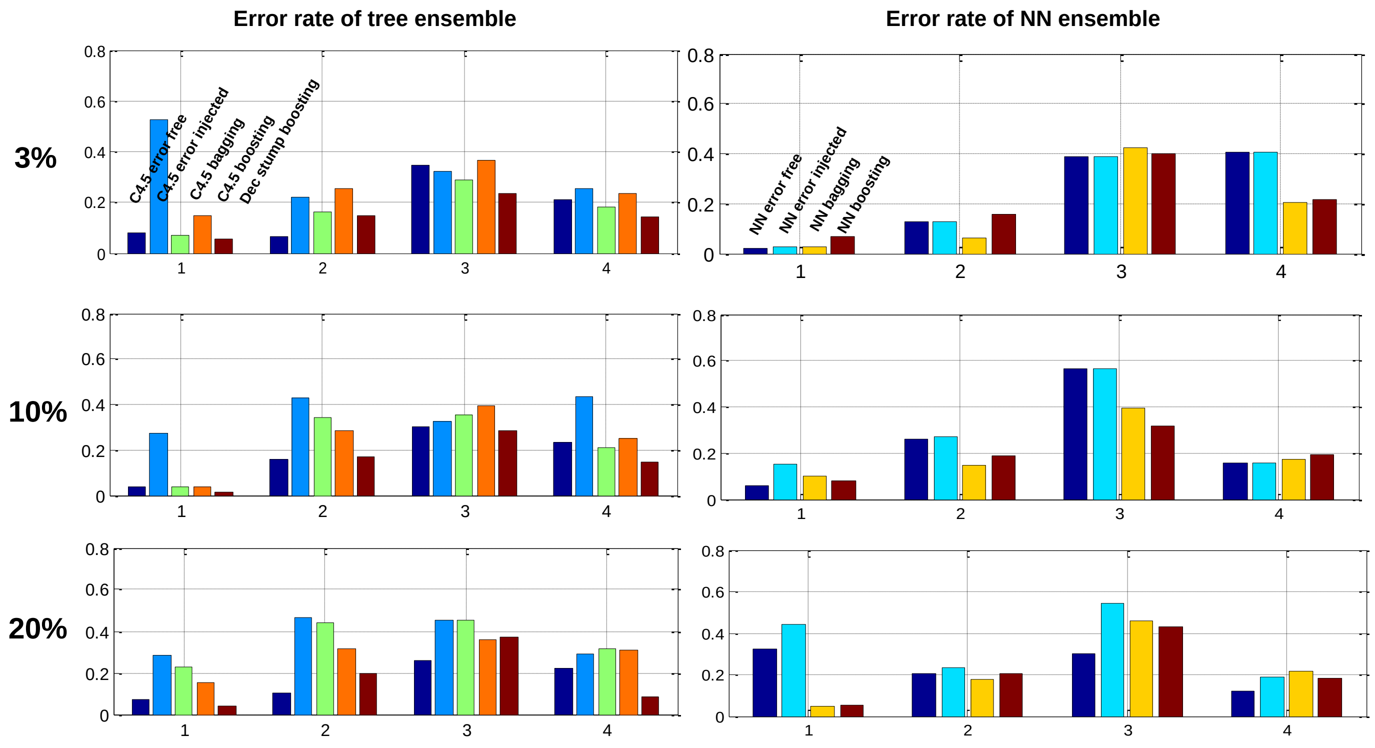


Fig. 9. Results: test error rate for different data set at error rate of 3%, 10% and 20%

till 10% percent error rate. The difference can be partially explained by noting that for decision trees, at each node a hard decision is made, so error have high chance to propagate to output, while for ANN, each layer does not make a hard decision but use a sigmoid function to suppress the output, which might help reduce the effect of node errors.

2) For tree ensemble, Adaboost with decision stump always give best performance, consistent with Fig. 8. Boosting decision tree tends to suffer from over-fitting at low error rate. This problem is mitigated at high error rate due to the fact that at high error rate, the ensemble size tends to be reduced.

3) For ANN ensemble, Adaboost and Bagging can out perform each other depending on data set, and there is no consistent behavior across different error rate. The over-fitting problem of ANN is less severe compared with trees, and similarly the over-fitting gets mitigated at high error rate due to the reduction of ensemble size.

V. Conclusion and Future Work

In this paper, we show that CE is an effective method to enhance the robustness of ML hardware. Bagging and Adaboost are explored with decision tree and ANN as base classifiers. Simulation results show that ANN is inherently tolerant to hardware errors with up to 10% hardware error rate. With simple majority voting scheme, CE is able to effectively reduce the classification error rate for almost all tested data sets, with maximum test error reduction of 48%. For tree ensemble, Adaboost with decision stump as weak learner gives best results; while for ANN, bagging and boosting outperform each other depending on data set. In the future, we can extend the framework to include more ensemble methods, to handle multi-class problems, to mitigate the over-fitting problem by using regularization. Also it will be helpful to implement part of the algorithm in hardware to evaluate the performance of CE on ML kernels.

Acknowledgment

The author would like to thank Prof. Hasegawa-Johnson and Sujeeth Subramanya Bharadwaj for their help and guidance.

References

[1] N. Verma, K. H. Lee, K. J. Jang, and A. Shoeb, "Enabling system-level platform resilience through embedded data-driven inference capabilities in electronic devices," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, 2012, pp. 5285–5288.

[2] J. Choi, E. P. Kim, R. A. Rutenbar, and N. R. Shanbhag, "Error resilient mrf message passing architecture for stereo matching," in *Signal Processing Systems (SiPS), 2013 IEEE Workshop on*, 2013, pp. 348–353.

[3] L. ying Yang, Z. Qin, and R. Huang, "Design of a multiple classifier system," in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 5, 2004, pp. 3272–3276 vol.5.

[4] G. Giacinto, F. Roli, and G. Fumera, "Design of effective multiple classifier systems by clustering of classifiers," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 2, 2000, pp. 160–163 vol.2.

[5] T. G. Dietterich, "Ensemble methods in machine learning," in *MULTIPLE CLASSIFIER SYSTEMS, LBCS-1857*. Springer, 2000, pp. 1–15.

[6] J. Rodriguez, L. Kuncheva, and C. Alonso, "Rotation forest: A new classifier ensemble method," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 10, pp. 1619–1630, 2006.

[7] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[8] L. B. Statistics and L. Breiman, "Random forests," in *Machine Learning*, 2001, pp. 5–32.

[9] L. Xu, A. Krzyzak, and C. Suen, “Methods of combining multiple classifiers and their applications to handwriting recognition,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, no. 3, pp. 418–435, 1992.

[10] T. K. Ho, J. Hull, and S. Srihari, “Decision combination in multiple classifier systems,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, no. 1, pp. 66–75, 1994.