A Design Method for Multiclass Classifiers

Tsutomu Sasao Yuto Horikawa Yukihiro Iguchi
Meiji University
Kawasaki 214-8571, Japan.

***Abstract*—Logic minimization is used to design multiclass classifiers for machine learning. This can be an alternative to a neural network. A partially defined classification function f is derived from the training set. Our multiclass classifier correctly classifies not only all the samples in the training set, but also much of samples in the unseen test set. To improve the test accuracy, 1) minimization of variables in f; 2) minimization of the number of products in a ternary SOP for f; and 3) maximization of the number of literals in a ternary SOP for f, are performed. Experimental results using MNIST and fashion MNIST data set show that logic minimization improves the test accuracy. Our classifiers can be easily implemented by LUTs and glue logic.**

***Index Terms*—partially defined function, support minimization, classification, digit recognition, MNIST, machine learning, neural network, ternary logic, generalization ability**

I. Introduction

MNIST [14] is a well known data set for machine learning. In [9], LeCun et al. showed that LeNet-5, a convolutional neural network, can recognize the MNIST images with the test accuracy higher than 0.99. After that, various binarized neural networks [5] [12] have been developed. Now, accuracies of the binarized neural networks are approaching to analog ones [15]. Unfortunately, development of such a network requires two steps: 1) design of an analog neural net, and 2) conversion of it into binary one. **If we can generate binary circuits directly without using neural nets, it would be more convenient.**

In this paper, we use a logic synthesis technique to develop multiclass classifiers. **We derive a partially defined function f from the training set, and design a simple circuit for f.** Such circuits satisfy the specification 100%, but have poor **generalization ability**. Usually, the classifiers need to predict correct results for unseen test data. To improve the generalization ability, we use three techniques: 1) minimization of variables for f, 2) minimization of products in a ternary sum-of-products expression (SOP) for f, and 3) maximization of literals in a ternary SOP for f.

This paper shows that a **reduction of variables and minimization of ternary SOPs improve the generalization ability of the circuits**. This is the main contribution[1]. Similar, but different approaches were presented in [11], [4].

II. Definitions and Problems

Definition 2.1: Consider a set of k distinct vectors of n bits. These vectors are **registered vectors**. For each registered vector, assign an integer between 1 and m, where $2 \leq m \leq k$, and k is the **weight** of the function. A **registered vector table** shows the **function value** for each registered vector. A **partially defined classification function** produces the corresponding function value when the input vector matches to a registered vector, and produces undefined function value when the input vector mismatches to all the registered vectors.

TABLE 2.1
Registered Vector Table with $k = 6$ and $m = 2$.

x_1	x_2	x_3	x_4	x_5	x_6	f
1	1	0	0	1	1	1
0	1	1	0	1	1	1
0	0	0	0	1	0	1
1	1	0	1	1	1	2
1	0	0	0	1	1	2
0	1	0	1	0	0	2

Definition 2.2: A partially defined classification function shows a mapping $f : D \rightarrow \{1, 2, \ldots, m\}$, where $D \subset B^n$ is the set of registered vectors and $B = \{0, 1\}$. The set of vectors $\vec{a} \in D$ such that $f(\vec{a}) = i$ is represented by F_i, where $D = \bigcup_{i=1}^{m} F_i$. We assume that $F_i \neq \varnothing$ $(i = 1, 2, \ldots, m)$.

Example 2.1: The registered vector table in Table 2.1 shows a classification function with weight $k = 6$ and $m = 2$. ■

Definition 2.3: Let F_1 and F_2 be subsets of B^n, where $B = \{0, 1\}$, and $F_1 \cap F_2 = \varnothing$. Consider the function f such that

$$\vec{a} \in F_1 \quad \Rightarrow \quad f(\vec{a}) = 1,$$
$$\vec{a} \in F_2 \quad \Rightarrow \quad f(\vec{a}) = 2.$$

f is **partially defined** if $F_1 \cup F_2 \subset B^n$. A **classifier** realizes the function f.

To evaluate the performance of a classifier, we use:

Definition 2.4:

$$Accuracy = \frac{\#\text{ of Correctly recognized samples}}{\text{Total }\#\text{ of samples.}}$$

The **training accuracy** is calculated by using data in the training set, while the **test accuracy** is calculated by using data in the test set.

III. Strategy for Multiclass Problems

Before showing the detail of our method, we survey existing methods for multiclass problems [2],[10],[3].

A. Direct Method

This method uses a single classifier to solve the m-class problem. Since it treats all the training data at a time, the design time tends to be large. Also, the size of hardware tends to be large, and the test accuracy tends to be low.

[1]Supported in part by a Grant-in-Aid for Scientific Research of the JSPS.

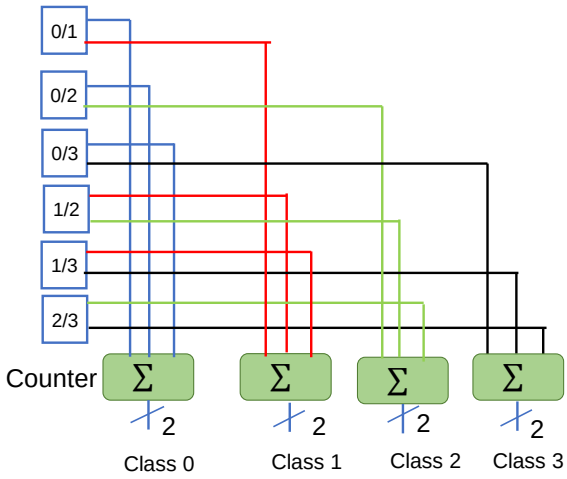

Fig. 4.1. $\binom{m}{2}$-Unit Realization ($m = 4$).

B. One-vs-Rest Method

This method uses m **base classifiers** to solve the m-class problem. To design a base classifier, for each class, we use the samples of that class as the positive samples, and the remaining samples of $(m-1)$ classes as the negative samples.

If a classifier is **binary**, i.e. it produces only a positive or a negative decision, then the total decision would be ambiguous for unseen data.

C. One-vs-one Method

This method uses $\binom{m}{2}$ base classifiers to solve an m-class problem. Each base classifier distinguishes a pair of classes. To design a base classifier, for each pair of classes, only the pairs of samples from the original data set are used.

When the test data is applied, a voting scheme is used. The outputs of all $\binom{m}{2}$ classifiers are used to find the class that received the highest number of positive predictions. However, for unseen data, erroneous prediction may occur.

D. Our Method

We use LUTs as basic elements, and use the one-vs-one approach. We use **ternary classifiers** instead of binary ones. This decreases the chance of ambiguity, and increases the test accuracy. To further improve the test accuracy, we use a **simple ensemble method**. This method partitions the training data into r disjoint sets, and uses r $\binom{m}{2}$-units to implement a classifier. It is called a $\binom{m}{2}$**-unit** $\times r$ **realization**.

IV. Multi-Class Classifier

In this section, we show the details of our m-class classifier. Fig. 4.1 shows the $\binom{m}{2}$**-unit realization**. It consists of $\binom{m}{2}$ ternary classifiers, counters, and a max selector. Note that the figure illustrates the case of $m = 4$, and the max selector is omitted.

A **ternary classifier**, shown by a square symbol, decides if the input data belongs to the class i, or the class j, or another class or unknown. The classifier i/j has two outputs: The output $(1, 0)$ denotes that the input data belongs to the class i; the output $(0, 1)$ denotes that the input data belongs to the class j; and the output $(0, 0)$ denotes that the input data belongs to another class or unknown. We assume that the ternary classifier is implemented by an LUT. The detail of the design method is shown in Section V.

A **counter**, shown by $\sum$ symbol, counts the number of 1's in the inputs, and represents it by a binary number. It has $m-1$ inputs and $\lceil \log_2 m \rceil$ outputs. The **max selector** (not shown in Fig. 4.1) selects the class with the largest count.

Example 4.1: Assume that a training sample in Class 0 is applied to the circuit in Fig. 4.1. Then, top three units recognize Class 0, and all the upper (blue) lines become 1. Thus, all the inputs to the counter for Class 0 become 1, and the counter receives three votes. On the other hand, all the lower (red, green, and black) lines of top three units become 0. So, the first inputs of the counters for Classes 1, 2, and 3 become 0. Thus, they receives votes less than 3. Since, Class 0 has the largest vote, Class 0 is detected by the max selector. ■

To design the classifier i/j, we use the training data for class 'i' and class 'j', only. In this case, only the input variables that are necessary to distinguish class i from class j are selected, and other variables are removed. For example, in the case of MNIST, the total number of input variables is 784, but only 13 variables are used to distinguish class 0 from class 1. This not only reduces the size of the LUT to implement the classifier, but also improves the generalization ability.

When a registered vector for class i or j is applied, then the classifier i/j produces non-zero output, which is always correct. When a non-registered vector is applied, it often produces non-zero output. This is because only the variables to distinguish a pair of classes are used. In this way, the classifier i/j *guesses* the class. In Section V, we show a simple example where a reduction of redundant variables improves generalization ability.

The $\binom{m}{2}$-unit realization produces much higher test accuracy than the single-unit realization. The next theorem shows that the training accuracy of the $\binom{m}{2}$-unit realization is 1.00.

Theorem 4.1: The $\binom{m}{2}$-unit realization always produces correct results for the data in the training set.

(Proof) The number of inputs to each counter is $m-1$. For an input data representing the class 0, the value of the counter for the class 0 is $m-1$. On the other hand, the values of the other counters are less than $m-1$. Thus, any training data for class 0 produces correct result. This is true for other classes. □

Fig. 4.2 illustrates the $\binom{m}{2}$**-unit** $\times r$ **realization**, where $m = 10$ and $r = 4$. The max selector is omitted. In this case, the registered vectors are partitioned into r groups of similar sizes. Note that, the number of input lines to each counter is $(m-1)r$. With this **ensemble method**, the test accuracy is improved, and at the same time the total memory size is reduced drastically.

Since there always exists a counter with the maximum value, input data is always recognized either correctly or incorrectly.

V. Design of Ternary Classifier

Generalization ability is power to adapt properly to new,

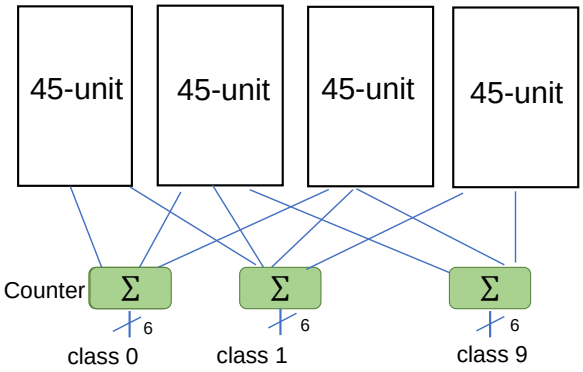

Fig. 4.2. $\binom{m}{2}$-unit $\times r$ realization ($m = 10, r = 4$).

TABLE 5.1
EXAMPLE FUNCTION WITH $k = 10$ AND $m = 2$.

		x_1	x_2	x_3	x_4	x_5	f
F_1	$\vec{a}_1$	1	1	0	1	1	1
	$\vec{a}_2$	1	0	1	1	1	1
	$\vec{a}_3$	1	0	1	0	0	1
	$\vec{a}_4$	1	0	0	1	1	1
	$\vec{a}_5$	0	0	0	1	0	1
F_2	$\vec{b}_1$	1	0	1	1	0	2
	$\vec{b}_2$	0	1	1	1	1	2
	$\vec{b}_3$	0	1	0	1	0	2
	$\vec{b}_4$	0	0	1	0	1	2
	$\vec{b}_5$	0	0	0	1	1	2

previously unseen data, drawn from the training data. Design of networks with a high generalization ability is a major research topic in machine learning. Networks with a high test accuracy have a good generalization ability. Occam's razor recommends to use as simple rule as possible [1].

In this part, we show a design method for a ternary classifier. Also, we show that logic optimization can improve the generalization ability.

Example 5.1: Consider the function f in Table 5.1. The number of variables in f is five. The upper five vectors form the F_1 set, while the lower five vectors form the F_2 set. Since the variable x_3 is redundant[2] in f, x_3 can be removed from f to obtain the function $\hat{f}$ shown in Table 5.2. A method to reduce variables is shown in Section VI. Next, consider the generalization ability of the functions. Fig. 5.1 shows the map for the example function, where the blank cells denote undefined. The number of undefined cells is 22. This function has no generalization ability, but only stores the data in the

TABLE 5.2
EXAMPLE FUNCTION AFTER REMOVAL OF x_3.

x_1	x_2	x_4	x_5	$\hat{f}$
1	1	1	1	1
1	0	1	1	1
1	0	0	0	1
1	0	1	1	1
0	0	1	0	1
1	0	1	0	2
0	1	1	1	2
0	1	1	0	2
0	0	0	1	2
0	0	1	1	2

[2]This can be verified by the fact that $f(|x_3 = 0)$ and $f(|x_3 = 1)$ are compatible [13].

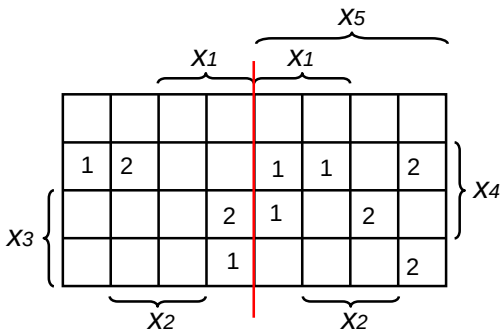

Fig. 5.1. Example function (Original)

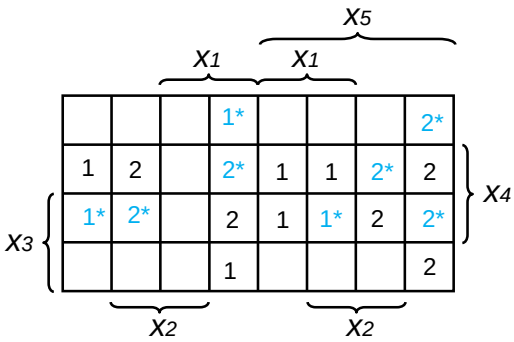

Fig. 5.2. Example function (After removal of x_3)

training set.

Fig. 5.2 shows the map for $\hat{f}$, the example function after removal of x_3. Note that 8 cells with 1*, and 2* denote newly introduced minterms by the reduction of x_3. The values of the function for these 8 minterms are *guessed* by the removal of x_3. This means that the reduction of variables improves generalization ability. However, there still remain 14 undefined cells.

Fig. 5.3 shows the map for the reduced function $\hat{f}$, after minimization of the **ternary** SOP. Note that $\hat{f}$ takes three values 0, 1, 2, where 0 corresponds to undefined. Also, cells for 1's and 2's cannot be combined together. The minimized ternary SOP for $\hat{f}$ is

$$\begin{aligned}\mathcal{F} &= 1 \cdot (x_1x_4x_5 \vee x_1\bar{x}_2\bar{x}_4\bar{x}_5 \vee \bar{x}_1\bar{x}_2x_4\bar{x}_5) \\ &\quad 2 \cdot (\bar{x}_1x_5 \vee x_1\bar{x}_2x_4\bar{x}_5 \vee \bar{x}_1x_2x_4\bar{x}_5).\end{aligned}$$

Two cells with + marks denote newly introduced minterms that are *guessed* by the minimization of the ternary SOP for $\hat{f}$. Note that there still remain 12 undefined cells. ■

The above example shows the influence of logic minimization on the prediction of values in a classifier. When the training set and test set are completely random, such a guess is useless. However, for functions for image recognition, experimental results show that such a guess is really useful.

Note that the logic minimization of **ternary classification for machine learning** is different from that of **binary function for circuit optimization**. If we use a **binary** minimizer for circuit design, more *undefined* cells are assigned values. In this case, the minimized SOP lost the information on the confidence.

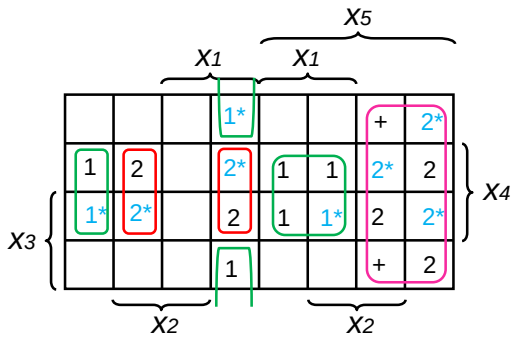

Fig. 5.3. Example function (After ternary SOP minimization)

In the circuit minimization, all the true minterms are covered by the loops, where the **loops must be as large possible**. This is to reduce the interconnection (literals in the SOP). However in the optimization for machine learning, the **loops must be as small as possible**. With ternary minimization of SOP, only the minterms with high confidence are assigned values.

In the case of $\binom{m}{2}$-unit realization, many units guess the classes, and the counters and the max selector find the most probable class.

VI. Optimization Algorithms

A. Reduction of Variables

This part shows a method to reduce the number of variables in the classification function. Since the number of the original input variables are very large, we cannot use the exact method [8], but have to use a heuristic method.

To reduce the variables, we introduce the impurity measure μ. The reduction of μ tends to make the decision tree as balanced as possible.

Definition 6.1: Let $\vec{a}$ be a vector showing the selected variables. Let $Size(j)$ be the number of vectors that belong to the partition j generated by $\vec{a}$. Let $Hist(j, Value)$ be the number of vectors that belong to the partition j and whose value corresponds to $Value$. In this case, the following relations hold:

$$\sum_{Value=1}^{m} Hist(j, Value) = Size(j), \quad \sum_{j=0}^{2^t-1} Size(j) = k,$$

where k denotes the total number of vectors, and t denotes the number of variables specified by $\vec{a}$. In this case, the **impurity measure** of the function is

$$\mu = \sum_{j=0}^{2^t-1} \left[Size(j)^2 - \sum_{Value=1}^{m} Hist(j, Value)^2 \right]$$

When $\mu = 0$, the classification function f can be represented by the variables specified by $\vec{a}$.

Algorithm 6.1: (A heuristic method to reduce the number of variables)

1) Given a classification function f.
2) Compute the impurity measures μ, where each variable x_i is fixed, for $i = 1, 2, \ldots n$.
3) Select a variable x_i that minimizes the value of μ. Let $\vec{a}$ denote the set of selected variables. Let $\vec{a} \leftarrow \vec{e}_i$, where $\vec{e}_i$ is the unit vector whose i-th element is 1, and other elements are 0's.
4) Among the remaining variables, select a variable x_j that minimizes μ. Let $\vec{a} \leftarrow \vec{a} \vee \vec{e}_j$.
5) If $\mu > 0$, then go to step 4.
6) If $\mu = 0$, then stop.

TABLE 6.1
When the function is expanded by x_1.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	1	1	0	1	1	1
0	0	0	0	1	0	1
0	1	0	1	0	0	2
1	1	0	0	1	1	1
1	1	0	1	1	1	2
1	0	0	0	1	1	2

TABLE 6.2
When the function is expanded by x_2.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	**0**	0	0	1	0	1
1	**0**	0	0	1	1	2
1	**1**	0	0	1	1	1
0	**1**	1	0	1	1	1
0	**1**	0	1	0	0	2
1	**1**	0	1	1	1	2

Example 6.1: Consider the 6-variable function shown in Table 2.1. When the function is expanded by x_1, we have Table 6.1. Note that $Size(0) = 3$ and $Size(1) = 3$.
For the partition $x_1 = 0$, $Hist(0, 1) = 2, Hist(0, 2) = 1$.
For the partition $x_1 = 1$, $Hist(1, 1) = 1, Hist(1, 2) = 2$.
Thus, the impurity measure is $\mu = [3^2 - (2^2 + 1^2)] + [3^2 - (1^1 + 2^2)] = 8$.

When the function is expanded by x_2, we have Table 6.2. Note that $Size(0) = 2$ and $Size(1) = 4$.
For the partition $x_2 = 0$, $Hist(0, 1) = 1, Hist(0, 2) = 1$.
For the partition $x_2 = 1$, $Hist(1, 1) = 2, Hist(1, 2) = 2$.
Thus, the impurity measure is $\mu = [2^2 - (1^2 + 1^2)] + [4^2 - (2^2 + 2^2)] = 10$.

When the function is expanded by x_4, we have Table 6.3. Note that $Size(0) = 4$ and $Size(1) = 2$.
For the partition $x_4 = 0$, $Hist(0, 1) = 3, Hist(0, 2) = 1$.
For the partition $x_4 = 1$, $Hist(1, 1) = 0, Hist(1, 2) = 2$.
Thus, the measure is $\mu = [4^2 - (3^2 + 1^2)] + [2^2 - (0^2 + 2^2)] = 6$.

In this way, for each variable, we compute the measure. In summary, when the function is expanded by x_i, the measures are $\mu(x_1) = 8$, $\mu(x_2) = 10$, $\mu(x_3) = 12$, $\mu(x_4) = 6$, $\mu(x_5) = 12$, $\mu(x_6) = 10$. Since x_4 yields the smallest measure, we use x_4 to expand the function.

In the next step, we select the second variable in a similar way, and find that when the function is expanded with x_1 and x_4, the measure becomes minimum.

In this case, Table 6.4 shows the partition. Note that

TABLE 6.3
When the function is expanded by x_4.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	1	1	**0**	1	1	1
0	0	0	**0**	1	0	1
1	1	0	**0**	1	1	1
1	0	0	**0**	1	1	2
0	1	0	**1**	0	0	2
1	1	0	**1**	1	1	2

TABLE 6.4
WHEN THE FUNCTION IS EXPANDED BY x_1 AND x_4.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	1	1	**0**	1	1	1
0	0	0	**0**	1	0	1
0	1	0	**1**	0	0	2
1	1	0	**0**	1	1	1
1	0	0	**0**	1	1	2
1	1	0	**1**	1	1	2

$Size(00) = 2$, $Size(01) = 1$, $Size(10) = 2$, $Size(11) = 1$.
For $(x_1, x_4) = 00$, $Hist(00, 1) = 2, Hist(00, 2) = 0$.
For $(x_1, x_4) = 01$, $Hist(01, 1) = 0, Hist(01, 2) = 1$.
For $(x_1, x_4) = 10$, $Hist(10, 1) = 1, Hist(10, 2) = 1$.
For $(x_1, x_4) = 11$, $Hist(11, 1) = 0, Hist(11, 2) = 1$.

Thus, the measure is $\mu = [2^2 - (2^2 + 0^2)] + [1^2 - (0^2 + 1^2)] + [2^2 - (1^2 + 1^2)] + [1^2 - (1^2 + 0^2)] = 2$.

In the next step, we select the third variable in a similar way, and find that when the function is expanded with x_1, x_2 and x_4, the measure μ becomes zero. Table 6.5 shows the partition. Thus, the function is represented by (x_1, x_2, x_4). ■

TABLE 6.5
WHEN THE FUNCTION IS EXPANDED BY x_1, x_2 AND x_4.

x_1	x_2	x_3	x_4	x_5	x_6	f
0	**0**	0	**0**	1	0	1
0	**1**	1	**0**	1	1	1
0	**1**	0	**1**	0	0	2
1	**0**	0	**0**	1	1	2
1	**1**	0	**0**	1	1	1
1	**1**	0	**1**	1	1	2

Since this algorithm is a heuristic one, it does not always produce a minimum solution.

B. Simplification of Ternary SOP

Algorithm 6.2: (Minimization of a ternary SOP)

1) Assume that F_1 and F_2 are given as SOPs, where $F_1 \cap F_2 = \emptyset$.
2) Merge the cubes of F_1 and F_2, respectively.
3) $DC \leftarrow \overline{F_1 \cup F_2}$.
4) Simplify F_1 using DC as *don't cares*.
5) Simplify F_2 using DC as *don't cares*.
6) $F_2 \leftarrow F_2 \cap \overline{F_1}$.

In the simplifications of ternary SOPs in Steps 4) and 5), we try to **minimize** the number of produces, while trying to reduce the volume of cubes [7]. This corresponds to **maximize** the number of literals in an SOP. Thus, values are assigned only to the minterms that are near to the minterms for registered vectors. Since minimizations of Steps 4) and 5) are done independently, some undefined minterms are assigned to both F_1 and F_2. Thus, Step 6) is necessary to remove inconsistency.

Example 6.2: Consider the function in Example 6.1. After the removal of redundant variables, we have

$$\begin{aligned} \mathcal{G}_1 &= 1 \cdot (\bar{x}_1\bar{x}_2\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_4 \vee x_1x_2\bar{x}_4) \\ &\quad 2 \cdot (\bar{x}_1x_2x_4 \vee x_1\bar{x}_2\bar{x}_4 \vee x_1x_2x_4). \end{aligned}$$

In Step 2) of Algorithm 6.2, products of $\mathcal{G}_1$ are merged to

$$\mathcal{G}_2 = 1 \cdot (\bar{x}_1\bar{x}_4 \vee x_2\bar{x}_4) \vee 2 \cdot (x_2x_4 \vee x_1\bar{x}_2\bar{x}_4).$$

TABLE 7.1
RESULT FOR 45-UNIT REALIZATION (MNIST).

Result	Variable minimization	ternary SOP minimization
Correctly recognized	8773.5	9053.6
Incorrectly recognized	1219.5	939.4
Test Accuracy	0.878	0.906

TABLE 7.2
RESULT FOR 45-UNIT×8 REALIZATION (MNIST).

Result	Variable minimization	ternary SOP minimization
Correctly recognized	9024.8	9291.3
Incorrectly recognized	968.2	701.7
Test Accuracy	0.903	0.930

For this particular example, Steps $4) \sim 6)$ of Algorithm 6.2 does not change the function.

Table 6.1 shows a 6-variable function, and specifies the outputs for 6 combinations, while $\mathcal{G}_2$ shows a 3-variable function, and specifies the outputs for 6 combinations. Each minterm of $\mathcal{G}_2$ corresponds to $2^3 = 8$ minterms of Table 6.1. So, $\mathcal{G}_2$ specifies $6 \times 8 = 48$ combinations in Table 6.1. Thus, $48 - 6 = 42$ combinations are guessed by the simplification . ■

VII. EXPERIMENTAL RESULTS

A. MNIST

The MNIST [14] data set consists of bit maps of 28×28 handwritten digits images. The training set consist of 6×10^4 images, while the test set consists of 10^4 images. They are grayscale images, but we converted them into binary ones, by setting the threshold to 96. In this way, we had an n-variable binary-input m-valued function, where $n = 28 \times 28 = 784$, and $m = 10$. Also in this process, we removed duplicated data. The number of samples in the training set is 59981, while the number of samples in the test set is 9993. Since the number of different classes is $m = 10$, $\binom{m}{2} = 45$ ternary classifiers are used. Such realization is called **45-unit realization**.

Table 7.1 shows recognition results of the 45-unit realization. In this process, we had 45 component functions, each of which has about 1.25×10^4 samples. The number of variables for each component function is reduced to $13 \sim 25$ by Algorithm 6.1. The second column shows the results when only the minimization of variables was used. The last column shows the results when the ternary SOP minimization was applied after variable minimization. Note that the ternary SOP minimization (Algorithm 6.2) improved the test accuracy.

Table 7.2 shows recognition results of the **45-unit**$\times 8$ **realization**. In this case, each component function has 1.5×10^3 samples, on the average, and the number of variables for each component function is reduced to $6 \sim 17$. Both the ensemble method and the ternary SOP minimization improved the test accuracy. For example, consider the classifier 0/1, the unit that distinguishes digit 0 from digit 1. The number of training images (minterms) was 12646, and the number of variables was 784. Algorithm 6.1 reduced the number of variables to 13, and the number of minterms to 895. Algorithm 6.2 increased the number of minterms to 6216. Thus, the classifier 0/1 recognizes

TABLE 7.3
RESULT FOR 45-UNIT REALIZATIONS (FASHION-MNIST).

Result	Variable minimization	ternary SOP minimization
Correctly recognized	6934.8	7719.1
Incorrectly recognized	3051.2	2266.9
Test Accuracy	0.694	0.773

TABLE 7.4
RESULT FOR 45-UNIT$\times 8$ REALIZATIONS (FASHION-MNIST).

Result	Variable minimization	ternary SOP minimization
Correctly recognized	8079.3	8310.3
Incorrectly recognized	1906.7	1675.7
Test Accuracy	0.809	0.832

$6216 \times 2^{\{784-13\}} \simeq 7.72 \times 10^{235}$ images. This shows the power of generalization ability of logic minimizations.

B. Fashion-MNIST

Fashion-MNIST [6] is a dataset of Zalando's article images. Similarly to MNIST, each sample is a 28×28 grayscale image, associated with a label from 10 classes. Each training and test example is assigned to one of the following labels: 0 (T-shirt/top); 1 (Trouser); 2 (Pullover); 3 (Dress); 4 (Coat); 5 (Sandal); 6 (Shirt); 7 (Sneaker); 8 (Bag); 9 (Ankle boot). In this case, the threshold of the grayscale was set to 32. The number of samples in the training set is 59954, while the number of samples in the test set is 9986.

Table 7.3 shows the results for fashion-MNIST. In this case, linear transformations [13] of degree two was used to reduce the number of variables. Again, both the ensemble method and the ternary SOP minimization improved the test accuracy.

Table 7.4 shows recognition results of the 45-unit$\times 8$ realizations. Again, ternary SOP minimization improved the test accuracy. The test accuracies are not so good as neural nets, but the circuits are simpler.

C. Comparison of Total Memory Sizes

We assume that all the units are realized by LUTs. Table 7.5 compares the average number of input variables for each unit. The 45-unit realization requires LUTs with $\sum_{i=0}^{8} \sum_{j=i+1}^{9} 2 \times 2^{p_{ij}}$ bits, where p_{ij} denotes the number of the variables for the unit i/j. Note that each base classifier has two outputs. Table 7.6 compares the total memory size for various classifiers. 45-unit $\times 8$ realizations require smaller amount of memory, because the number of input variables for each unit is smaller than that of 45-unit realizations. Note that they do not contain the costs for counters and the max selector.

VIII. CONCLUDING REMARKS

In this paper, we showed a design method for an m-class classifier. It consists of $\binom{m}{2}$ ternary classifiers, m counters, and a max selector. A partially defined function f is derived from the training set, and a classifier is designed for f.

TABLE 7.5
AVERAGE NUMBER OF INPUT VARIABLES FOR EACH UNIT.

Architecture	MNIST	Fashion MNIST
45-unit	19.42	14.17
45-unit$\times 8$	12.34	7.70

TABLE 7.6
COMPARISON OF TOTAL MEMORY SIZES (MEGA BITS).

Architecture	MNIST	Fashion MNIST
45-unit	197.94	1,692.34
45-unit$\times 8$	7.09	4.10

Our contributions of this paper are:

- Showed a new design method for a multiclass classifier.
- Showed methods to improve the test accuracy. 1) Minimization of variables, 2) Minimization of products in a SOP, and 3) Maximization of literals in a SOP.
- Showed that ternay SOP minimization improves the test accuracy for MNIST and Fashion MNIST data sets.

To the best of authors' knowledge, we are the first to have successfully minimized the ternary SOPs for the MNIST/fashion MNIST data set, and to show that logic minimization improves the test accuracy.

REFERENCES

[1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, Vol. 24, Issue 6, 1987, pp. 377-380.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, CRC Press, New York, 1984.

[4] S. Chatterjee,"Learning and memorization," *International Conference on Machine Learning (ICML 2018)*, Stockholm, Sweden, July 10-15, 2018, pp. 754-762.

[5] M. Courbariaux, Y. Bengio, and J.P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," *Advances in Neural Information Processing Systems*, pp. 3123-3131, 2015.

[6] https://www.kaggle.com/zalando-research/fashionmnist

[7] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. and Develop.*, pp. 443-458, Sept. 1974.

[8] J. Kuntzmann, *Algèbre de Boole*, Dunod, Paris, 1965. English translation: *Fundamental Boolean Algebra*, Blackie and Son Limited, London and Glasgow, 1967.

[9] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, " Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, Vol. 86, No. 11, pp.2278-2324, November 1998.

[10] A. Mohamed,"Survey on multiclass classification methods," *Technical Report, Caltech*, Nov. 2005.

[11] A. L. Oliveira and A. Sangiovanni-Vincentelli, "Learning complex boolean functions:Algorithms and applications," *Advances in Neural Information Processing Systems*, No. 6, pp. 911-918. Morgan-Kaufmann, 1994.

[12] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," *European Conference on Computer Vision*, pp.525-542, 2016.

[13] T. Sasao, *Index Generation Functions*, Morgan & Claypool, Oct. 2019.

[14] http://yann.lecun.com/exdb/mnist/

[15] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "LogicNets: Co-designed neural networks and circuit for extreme-throughput applications,", *Int. Conf. on Field-Programmable Logic and Applications* (FPL-2020), pp. 291-297. 31 Aug.- 4 Sept. 2020.