Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP 2007)

Developing a Reliable Learning Model for Cognitive Classification Tasks Using an Associative Memory

Ali Ahmadi, Hans Jürgen Mattausch, M. Anwarul Abedin, Tetsushi Koide, Yoshinori Shirakawa, and M. Arifin Ritonga

Research Center for Nanodevices and Systems, Hiroshima University, Higashi-Hiroshima, Japan
Email: ahmadi@sxsys.hiroshima-u.ac.jp

***Abstract*—An associative memory based learning model is proposed which uses a short and long-term memory and a ranking mechanism to manage the transition of reference vectors between two memories. The memorizing process is similar to that in human memory. In addition, an optimization algorithm is used to adjust the reference vectors components as well as their distribution, continuously. Comparing to other learning models like neural networks, the main advantage of the proposed model is no need to pre-training phase as well as its hardware-friendly structure which makes it implementable by an efficient LSI architecture without requiring a large amount of resources. The system was implemented on an FPGA platform and tested with real data of handwritten and printed English characters and the classification results found satisfactory.**

I. Introduction

The machine learning in artificial systems has attracted a great amount of attention in recent years. Many different approaches have been developed for improvement of learning systems in the literature [1-3]. As for a learning model, one of the main issues is the feasibility of training online in a real-time application. However, in most of currently existing models a pre-training process becomes necessary. For example in case of neural networks, as a well-known connectionist learning model, it is known that in most of practical case the system can only be learned perfectly when the entire data set is made available to the network in a prior training procedure [4-5]. If a neural network is just given a subset of the data, it often fails to learn the correct generalization and remains stuck in a local error minimum. The next issue in a learning model is a hardware-friendly structure. A specialized hardware for a learning system offers applicable advantages such as higher speed in processing of repetitive calculations, lower cost by lowering total component counts, and increased reliability in the sense of reduced probability of equipment failure. In case of neural networks, generally the large number of weight parameters as well as the complicated algorithm of learning which basically needs a large training data set and a long time of training, makes it difficult to implement it in an integrated hardware structure with limited amount of resources.

In this paper, we propose a novel learning model which is capable to learn from input samples constantly and adjust the reference pattern set whenever necessary. The underlaying concept of the learning algorithm is based on taking a short and long-term memory and a ranking mechanism which manages inclusion and elimination of reference patterns as well as their transition between the two memories. Also, the reference vectors magnitude as well as their distribution are adjusted continuously by means of an optimization algorithm. The main advantage of the proposed algorithm comparing to other learning methods, like neural networks, is that it can be easily implemented in the lower level hardware as an LSI architecture with no need to the large hardware resources.

In order to enhance the pattern matching speed, the classification process is designed on the basis of using a parallel associative memory. The prototype of associative memory we use here has been already designed [6] and has a mixed analog-digital fully-parallel architecture for nearest Hamming/ Manhattan-distance search.

The prototype of the hardware system was first designed in a higher level programming language (MATLAB) and then transferred to HDL code and after synthesis and simulation was programmed in an FPGA platform of the Altera Stratix family, successfully. In order to evaluate the system performance, it was used in the real application of character recognition. A number of handwritten data samples were used for testing the system and the results approved the efficiency of classification with a high speed of pattern matching as well as the learning functionality.

II. Model Description

As it is described in the Introduction, the main feature of the proposed model is a dynamic learning function which makes it useful in the real-time applications like video detection, online text recognition, intelligent systems, etc. With its hardware-friendly structure, the model can be easily implemented in different hardware platforms. When compared with other learning models like neural networks, the main advantage of the proposed model is that a pre-training phase is unnecessary and the model has a hardware-friendly structure. The core part of the model is the learning procedure but it also includes two further blocks of preprocessing and classification, prior to the learning. More explanations on the performance of each block are given in the following.

A. Learning Procedure

The core concept of learning in the model is based on a short/long term memory which is very similar to the memorizing procedure in the human brain. For this, the memorized reference patterns are classified into two areas according to their learning ranks. One is a short-term storage

area where new information is temporarily memorized, and the other is a long-term storage area where a reference pattern can be memorized for a longer time without receiving the direct influence of incoming input patterns. The transition of reference patterns between short-term and long-term storages is carried out by means of a ranking algorithm. Besides, an optimization algorithm is applied to update the reference patterns and optimize their distribution as well as the threshold values used for classification and ranking. The block flowchart of Fig. 1 shows an outline of the learning procedure.

As can be seen from the flowchart, by taking a nearest-matching over the reference patterns memory, we get the winner address and winner-input distance ($D_{w\text{-}i}$). The winner-input distance is then compared with a local threshold value corresponding to each reference pattern to find if the

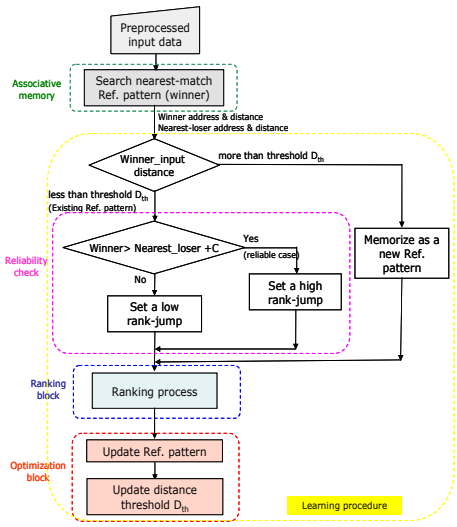

Fig. 1. Flowchart of the learning procedure. C is a constant value selected experimentally based on the data condition.

classification is acceptable or a new reference pattern to be added. In case that the winner-input distance is under threshold D_{th} and also there is a reasonable margin between winner and nearest-loser, the classification is considered as a reliable case and we give a high rank-jump to the winner pattern in the ranking memory, otherwise the winner will get a low rank-jump. Details of the two main blocks, *ranking* and *optimization*, are described below.

- **Ranking block**: Each reference pattern in the system is given an "unique rank" showing the occurrence level of the pattern. A ranking memory is considered for the ranks where the rank is as the index of the memory and the reference patterns address (its address in the reference memory) are saved as the content. The rank is increased by a predefined jump value in case of a new occurrence (when a new input is matched with the current reference pattern) , and reduced gradually when there is no occurrence of matching and other reference patterns getting higher ranks and shifting up to higher positions. The reference patterns are classified into the long-term and short-term memory according to their rank

whereas a specific rank level of *s_rank* is defined as the border of short-term and long-term memory. If the rank of each pattern gets higher then *s_rank,* it enters into the long-term memory, and conversely if its rank gets less then *s_rank* falls into the short-term memory. Figure 2 shows the flowchart of ranking process. As can be seen from the flowchart, if $D_{w\text{-}i}<D_{th}$ (i.e. a known Ref. pattern case) then we first search for the existing rank of the winner in the ranking memory.

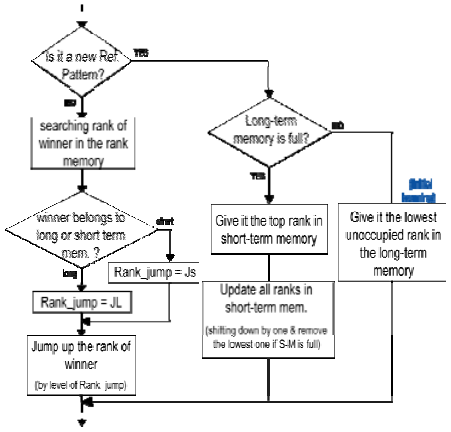

Fig. 2. Flowchart of the ranking process block.

If the winner belongs to the short-term memory (rank<*s_rank*) the rank advancement is J_S, and if the case of winner is belonging to the long-term memory (rank>*s_rank*), the rank advancement becomes J_L ($J_L>J_S$). Then each of the patterns having rank between the old and the new winner rank are reduced in rank by one (Fig. 3). The transition between short and long-term memory happens by these changes in rank.

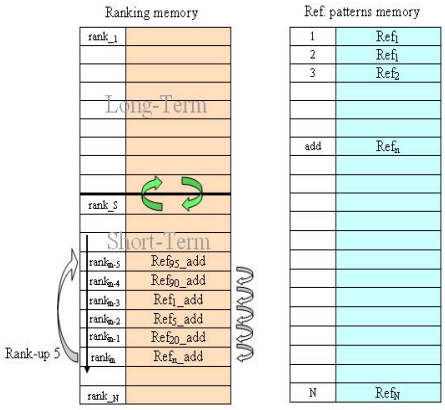

Fig. 3. Rank advancement for a currently existing reference pattern.

In the case of $D_{w\text{-}i}\geq D_{th}$, the system considers input and winner pattern to be different and takes the input pattern as a new reference pattern. The top rank of the short-term memory is given to this new reference pattern (if the long-term memory is not yet full, then the lowest rank in the long-term memory will be assigned) and subsequently the rank of each

of the other reference patterns existing in the short-term memory are moved down by one, and the reference pattern with the lowest rank will be erased from the memory.

- **Optimization block:** This block is used for renewing the reference patterns continuously according to the input data variation. The main purpose is to improve the reliability of the classification. We take two main updating steps including reference pattern magnitudes and distance thresholds. The first step is to decide whether a new reference pattern is generated or the nearest-matching pattern can be considered as the winner. The decision maker here is the local distance threshold D_{th} corresponding to each reference pattern, which itself is renewed during the optimization process. If the winner-input distance is greater than D_{th}, that is the case of new reference pattern generating, initial values are given to the new D_{th} and Ref pattern. Also, $D_{th(mean)}$ and $Ref_{(mean)}$ as well as the input counter memories are set with initial values. $D_{th(mean)}$ and $Ref_{(mean)}$ memories are used later for updating of D_{th} and Ref patterns. In case of a known reference pattern, i.e. winner-input distance is smaller or equal to D_{th}, we update the mean values of D_{th} and Ref pattern magnitude ($D_{th(mean)}$ & $Ref_{(mean)}$) for the current winner. The updating process is explained in the following. If the counter number, i.e. the number of inputs already assigned to the current winner, is larger than a predefined threshold N_{th}, then the D_{th} and Ref memory are updated with the last mean values and the counter is set to 1.

Updating process

- *Reference pattern magnitude*: A reference pattern vector is a combination of a reference image and its feature vector as described in the classification section. For each ref. vector we take a mean vector of input patterns matched with it as

$$\overline{\mathrm{Ref}} = \frac{1}{n}\sum \mathbf{x}_i \tag{1}$$

where $\mathbf{x}_i$ is the i^{th} input vector and $\overline{\mathrm{Ref}}$ is the mean-vector of the last *n* input assigned to the specific reference pattern. This mean-vector is updated at each input incoming as follows

$$\overline{\mathrm{Ref}}^{\,n} = ((n-1)\overline{\mathrm{Ref}}^{\,n-1} + \mathbf{x}_n)/n \tag{2}$$

where $\overline{\mathrm{Ref}}^{\,n-1}$ is the pervious value of mean-vector before input $\mathbf{x}_n$ entered. In order to simplify the instructions for hardware implementation, relation (2) can be written as

$$\overline{\mathrm{Ref}}^{\,n} = \overline{\mathrm{Ref}}^{\,n-1} - (\overline{\mathrm{Ref}}^{\,n-1} - \mathbf{x}_n)/n \tag{3}$$

where the division by *n* is performed only when *n* is a multiple of 8, using a simple right shift operator. The replacement of the reference vector with this updated mean-vector is achieved after a specific number of incoming inputs (N_{th}).

- *Distance thresholds*: In addition to reference patterns, the threshold values for the winner-input distance are updated in this block. For each reference vector we take a local distance threshold based on the distribution of local data. Similarly to relation (1) we have

$$\overline{D_{\mathrm{th}}} = \frac{1}{n}\sum D_{wi}^{k} \tag{4}$$

where D_{wi}^{k} is the winner-input distance of the k^{th} input sample and $\overline{D_{\mathrm{th}}}$ is the mean value of last *n* D_{wi}. Similarly to (3) we use following relation for updating this mean value.

$$\overline{D_{\mathrm{th}}}^{\,n} = \overline{D_{th}}^{\,n-1} - (\overline{D_{\mathrm{th}}}^{\,n-1} - D_{wi}^{n})/n \tag{5}$$

It should be noted that in relation (5) the D_{wi}^{n} is not only the winner-input distance of the inputs falling inside the current D_{th} but also of the input samples that are out of D_{th} but have been matched to the current reference pattern (however they are considered as new reference patterns later). This prevents the distance threshold D_{th} to become continuously smaller.

The replacement of the threshold D_{th} with this updated mean value is achieved after a specific number of incoming inputs (N_{th}).

B. Preprocessing Steps

Preprocessing steps are intended to read the input data and prepare it for the classification task. Since in the current work, the proposed learning model is applied for a character recognition task, the preprocessing steps are designed to provide the necessary preparations of the character input data prior to the classification step. The preprocessing steps include following blocks: *reading, noise removal, binarizing, labeling, segmentation, feature extraction, and normalizing.*

- **Reading:** Contains a reading device (line-scan sensor) moving on each line of the text with an appropriate speed and scans the data continuously as a sequence of thin frames. The frames between each two word spaces are collected and form a larger frame as a gray-scale bitmap array which holds all the word characters.

- **Noise removal:** As the most noises appearing in the texts are of the pepper & salt type, we apply a median filter with a neighborhood size of 3×3 for noise removing. A tree-diagram is used for finding the median of each 9 neighbor pixels [7]. The method is hardware-friendly and can be implemented in a pipelined structure.

- **Binarizing:** In this block the input image frame is binarized to a simple black-white bitmap by taking a local threshold value extracted via a mean filter with neighborhood size of 7×7. The structure of the mean filter is likely similar to the median filter explained above in terms of the registering manner of neighborhood pixels.

- **Labeling:** In the labeling process we scan all the input frame pixels sequentially while keeping the labels of preceding neighboring pixels (4 pixels' labels) in 4 registers and decide if the current pixel belongs to one of the preceding labels or gets a new label. The labels for all image pixels are then extracted in this way and saved in a label memory. In case of facing with equivalent labels for the same character, like in the

case of character W, the label equivalences are recorded in two buffers SLB1 and SLB2.

- **Segmentation:** Once the labeling process is terminated, the image memory is scanned once again for the segmentation task. We scan the labeled image N times each time searching label L_i (i=1…N). The label read from memory is then searched within a lookup-table (SLBF) which is already created based on data of SLB1 and SLB2 buffers, and is replaced with the final equivalent label. The addresses of pixels with label L_i are written in a new memory (SM) as a distinct segment L_i. Next, the boundaries of segment L_i are identified and a new 2-dimensional segment vector with binary values 0 and 1 is generated based on the addresses already saved in the SM memory.

- **Feature extraction:** In order to have a robust character classification, some characteristic features of the input pattern are extracted and grouped in a feature vector. The features are selected so that they have the minimum dependency on size and variation of data. Given a segmented (isolated) character we extract its moment based features as follows: *Total mass* (number of pixels in a binarized character), *Centroid,* Elliptical parameters (i.e. *Eccentricity* (ratio of major to minor axis) and *Orientation* (angle of major axis)), and *Skewness*.

In principle skewness is defined as the third standardized moment of a distribution as

$$\gamma = \mu_3 / \sigma^3 \qquad (6)$$

but to simplify the calculations we take a simpler measure of Karl Pearson [9] defined as

$$\gamma = 3\,(mean - median) / standard\ deviation \qquad (7)$$

and calculate horizontal and vertical skewness, separately.

All the above six features, i.e. total mass, centroid, eccentricity, orientation, horizontal skewness, and vertical skewness are then normalized as is described in the next section to generate the feature vector.

- **Normalizing:** Each segmented character as well as the feature vector are normalized prior to the classification. The segmented character bitmap is rescaled to 16×16 pixels using a bilinear interpolation technique. As for the feature vector, each feature value is normalized using the minimum and maximum of the feature in the memory, and gets a value ranging between 0 and 1.

C. Classification block

The classification task is carried out by using a nearest-matching method taking a hybrid distance measure: a Hamming distance for the main image vector D_H (image) which comes as a 256 bits vector after size normalizing, and an Euclidean distance for the feature vector D_E(feature). Because of the variations of the magnitudes of two distance measures, the values should be weighted. We use weighting factors as follows.

$$D = \alpha D_H\,(\text{image}) + \beta D_E(\text{feature}) \qquad (8)$$

where in our experiments we found α=0.25 and β=1 as the most effective values.

III. HARDWARE IMPLEMENTATION

The proposed learning algorithm is intended to be implementable in the lower level of the hardware structure with higher speed and minimum resources. As for prototyping of the whole model including preprocessing and classification blocks, we have chosen an FPGA platform. Figure 4 shows the block diagram of the system implemented in the FPGA. A pipelined architecture is used for the main data flow conducting between processing blocks. Furthermore, each block itself is designed in a pipelined structure. An Altera Stratix family FPGA device (EP1S80) with a resource capacity of 79K logic cells and 7.4 Mbits of RAM memory is applied. As for reading device we use the line-scan sensor ELIS-1024 of Panavision Co. with a resolution level of 1024×1 pixels equipped with a 16mm optical lens. The system first was designed in Verilog HDL and after synthesis and functional simulation, we performed route & placement for FPGA programming. The route & placement is done with the Altera QuartusII software. An average clock frequency of 20 MHz is selected for all processing blocks.

The resource usage for the whole system excepting the feature extraction part is reported in Table I. The DSP blocks listed in the table are used for the noise removal block in the preprocessing steps. From Table I we can see that only a reasonable amount of logic cells (19%) and memory block (38%) are needed for implementation of major system blocks. The main system clock can also be increases to higher values if faster I/O devices are used.

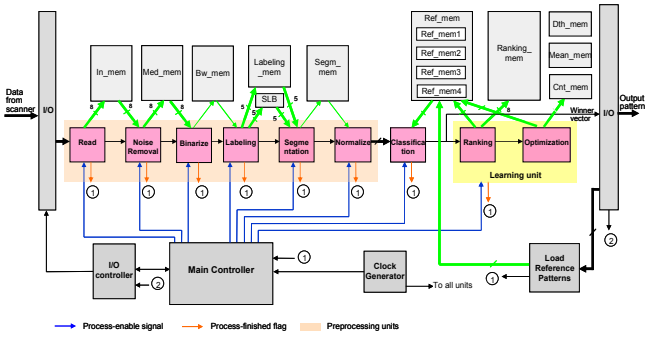

Fig.4. Block diagram of the system implemented in the FPGA.

TABLE I
ROUTE & PLACEMENT RESULTS FOR THE WHOLE SYSTEM IMPLEMENTED IN AN ALTERA STRATIX FPGA.

Fitter Status	Successful
Quartus II Version	4.2 Full Version
Top-level Entity Name	OCR
Family	Stratix
Device	EP1S80B956C7
Timing Models	Final
Total logic elements	15,451 / 79,040 (19 %)
Total pins	281 / 692 (40 %)
Total virtual pins	0
Total memory bits	2,854,174 / 7,427,520 (38 %)
DSP block 9-bit elements	28 / 176 (15 %)
Total PLLs	1 / 12 (8 %)
Total DLLs	0 / 2 (0 %)

We have also achieved an ASIC full-custom design for a simple model of the Ranking block only, suitable for implementation in an LSI architecture [8,11].

IV. Experimental Results and Evaluation

We examined the system performance in a real application for character recognition. The system was applied for recognition of both printed and handwritten English characters. As for printed characters a total number of 25 datasets including data of five different fonts (Times, Arial, Monotype, Symbols, Comic), noisy data, color background data, slightly rotated data, and data with different resolution were gathered and tested. Each set contained 26 characters. The classification results were very satisfactory and comparing to the results for handwritten data, with much less misclassifications. Here, we only report the results of experiments on handwritten data. A number of 35 datasets of English characters written by four different writers were used for the experiments. To have a variation limited data for this step of the test, the writers were asked to adhere to standard writing style and not to use a complicated writing manner. Additionally, we have also made some tests on more generalized datasets like CEDAR [10] and the results will be presented in our later works.

We achieved the evaluation of system performance by a Matlab simulation program in the three levels of: *classification*, *learning*, and *hardware efficiency*. Details of each level are described in the following.

- **Classification results:** The classification results during and after the learning process are reported in Table II and III. It is worth-noting that the system has started learning without any initial reference patterns or a predefined dataset. The data are given to the system in a random order. As can be realized from the Table II, the number of patterns added as new references as well as the misclassification rate is high in the beginning of the process but when the learning goes on and system gets to a more stable condition by adjusting continuously the reference pattern memory and distance thresholds, the misclassification rate reduces dramatically.

TABLE II
CLASSIFICATION RESULTS FOR FOUR DATASETS FROM DIFFERENT WRITERS DURING THE LEARNING PERIOD (TAKING NO INITIAL REF. PATTERNS).

Writer	Dataset A (52 samples)		Dataset B (52 samples)		Dataset C (52 samples)		Dataset D (52 samples)	
	Mis classify	New Ref. added	Mis classify	New Ref. added	Mis classify	New Ref. added	Mis classify	New Ref. added
1	9	42	12	25	11	19	19	14
2	11	36	9	26	16	19	10	31
3	10	36	6	20	10	21	12	12
4	11	39	14	20	13	22	5	20
otal (%)	19.7	73.6	19.7	43.8	24	38.9	22.1	37

In Table III we can see the classification results for two different datasets after a short period of learning (832 samples). The average misclassification rate is around 5% which is reasonable for this type of application.

The histogram of the winner-input distance for the classification of handwritten datasets is shown in the Fig. 5. From this histogram it can be found that the number of accurate classifications, that is classifications with lower winner-input distance, is very large which could imply high reliability of the classification.

TABLE III
CLASSIFICATION RESULTS FOR TWO TEST DATASETS AFTER A PERIOD OF LEARNING.

Writer	Test Set 1 (26 samples)		Test Set 2 (26 samples)	
	Mis classify	New Ref. added	Mis classify	New Ref. added
1	0	0	2	5
2	1	0	1	6
3	1	1	3	7
4	1	0	2	4
Total (%)	2.8	0.9	7.4	20.4

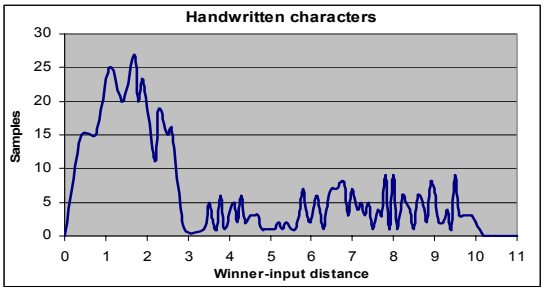

Fig. 5. Histogram of winner-input distance for classification of handwritten characters.

Figure 6 depicts the misclassification rate during the learning process. The first plot shows the misclassification rate versus the number of input samples and the second one versus the number of classified samples (i.e. input samples excluding the samples added as new reference patterns).

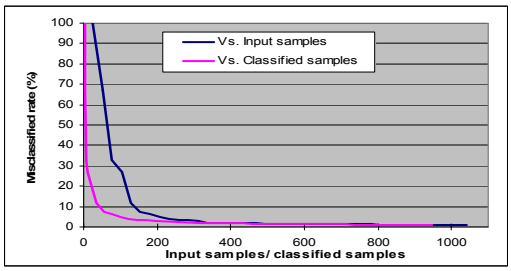

Fig. 6. Changes in the misclassification rate during the learning process.

- **Learning results:** As it is mentioned above, the model was applied in a learning task without initial reference patterns and we let the new patterns to be learned over the time. By a "learned" pattern we refer to a pattern that after a rank-jump process has entered into the long-term memory and can be

considered as a stable reference pattern. To investigate the efficiency of learning, we tested two different approaches for ranking a new reference pattern. In the first approach, the new reference pattern is initially given the highest unoccupied rank in the long-term memory as long as unoccupied ranks are available, whereas in the second approach it gets the top rank in the short-term memory from the beginning. The plot of Fig. 7 shows the learning progress for both approaches and is based on handwritten data listed in Table II. As can be seen the learning process is faster in the first approach, however, the approach two is more reliable.

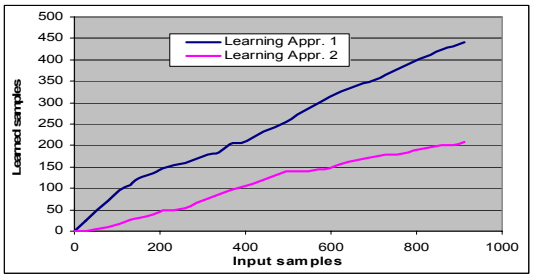

Fig. 7. Learning speed in the experimental tests on handwritten data.

It is described earlier in the description of ranking algorithm that the key parameters for ranking up a reference pattern and shifting it up to the higher locations in the long-term memory, are J_L and J_S which are currently taken as 7 and 5 for a reliable classification case and 5 and 3 for other cases. By taking other values for J_L and J_S, depending on the data distribution, the learning speed will be changed significantly. Also, the border rank (s_rank) separating short-term and long-term memory, now is predefined as 180, but can be defined dynamically based on a statistical calculation. In that case, the learning speed will be varying during the learning process.

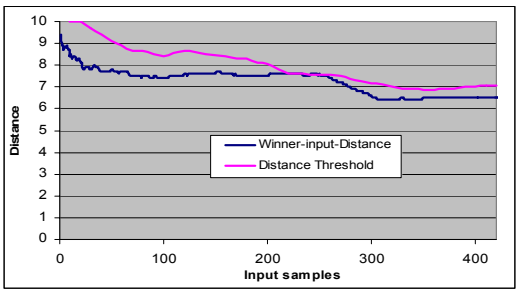

Fig. 8. Distribution of the average of winner-input distance and threshold values (D_{th}).

Figure 8 displays the changes in the average of winner-input distance as well as the distance threshold values (D_{th}) during the learning process. The distance thresholds take initial value of 10 and become averagely smaller during the optimization process which implies a more accurate classification.

- **Hardware efficiency:** Using a pipelined structure in the hardware, we could save hardware resources as well as processing time. As we use a pipelined architecture also between the main processing blocks, the block with the longest processing time is the bottleneck of the system and its processing time can be considered as the overall pipeline time. We realized that the segmentation block with processing time of roughly $n(P+p)\tau$ is the system bottleneck where P is the pixels number of input frame read by the sensor, p is pixels of each segmented character, τ is the time period ($1/f$), and n is the number of characters in a word. Given a main clock cycle of f=20 MHz and taking the average values of 15000, 1500, and 5 for parameters P, p, and n, respectively, we will get a pipeline process time of 82.5 μs per character and 4.12 ms per word, which are very satisfactory for this application. The main system clock and consequently the overall processing speed can still be increased to higher values if faster I/O devices are used.

V. Conclusion

In this paper we have proposed a learning model based on a short/long-term memory and an optimization algorithm for constantly adjusting the reference patterns. The system was implemented in an FPGA platform and tested with real data samples of handwritten and printed English characters. The classification results obtained from a simulation program showed an acceptable performance of classification and learning. We intend to improve the learning performance by taking more dynamic parameters in the ranking process.

Acknowledgment

This work has been supported by the 21st century COE program, Ministry of Education, Culture, Sports, Science and Technology, Japanese government. Authors would like to express their thanks for this support.

References

[1] T.G. Dietterich, "Machine Learning Research: Four Current Directions," *AI Magazine*, Vol. 18, No. 4, pp. 97-136, 1997.

[2] T. Mitchell, *Machine Learning,* McGraw Hill, USA, 1997.

[3] T.G. Dietterich, "Ensemble methods in machine learning," *Proc. of the First Int'l Workshop on Multiple Classifier Systems, Lecture Notes in Computer Science*, pp. 1-15, Springer, Cagliari, Italy, 2000.

[4] C. Harris, *Parallel distributed processing models and metaphors for language and development.* Ph.D. Dissertation, University of California, San Diego, 1991.

[5] S. Haykin, *Neural Networsk.* New Jersey: Prentice Hall, 1999.

[6] H.J. Mattausch, T. Gyohten, Y. Soda, and T. Koide, "Compact Associative-Memory Architecture with Fully-Parallel Search Capability for the Minimum Hamming Distance," *IEEE Journal of Solid-State Circuits,* Vol. 37, pp. 218-227, 2002.

[7] J. Smith, Xilinx Design Hints and Issues, Address on the Web: http://www.xilinx.com/xcell/xl23/xl23_16.pdf.

[8] Y. Shirakawa, M. Mizokami, T. Koide, and H.J. Mattausch, "Automatic Pattern-Learning Architecture Based on Associative Memory and Short/Long Term Storage Concept," *Proc. of SSDM'2004*, pp. 362-363, Japan, 2004.

[9] Wikipedia: http://en.wikipedia.org/wiki/Skewness.

[10] A database of handwritten texts generated by CEDAR research group in university of Buffalo. Web address: http://www.cedar.buffalo.edu/.

[11] Y. Shirakawa, H.J. Mattausch, and T. Koide, "Reference-Pattern Learning and Optimization from an Input-Pattern Stream for Associative-Memory-Based Pattern-Recognition System", *Proc. of MWSCAS'2004 (IEEE In'l Midwest Symposium on Circuits and Systems)*, Vol. I, pp. 561-564, Japan, 2004.