# A Human-memory Based Learning Model and Hardware Prototyping in FPGA

Ali Ahmadi, M.Anwarul Abedin, Hans Jürgen Mattausch, Tetsushi Koide, Yoshinori Shirakawa, and M. Arifin Ritonga
Research Center for Nanodevices Systems, Hiroshima University, Higashi-Hiroshima, 739-8527, Japan

## 1. Introduction

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. 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 this report, we propose a novel learning model which is capable to learn from input samples constantly and adjust the reference pattern set whenever necessary. 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 of the large hardware resources.

## 2. Model Description

### 2.1 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 flowchart of Fig. 1 shows an outline of the learning procedure. Details of the two main blocks, *ranking* and *optimization*, are described below.

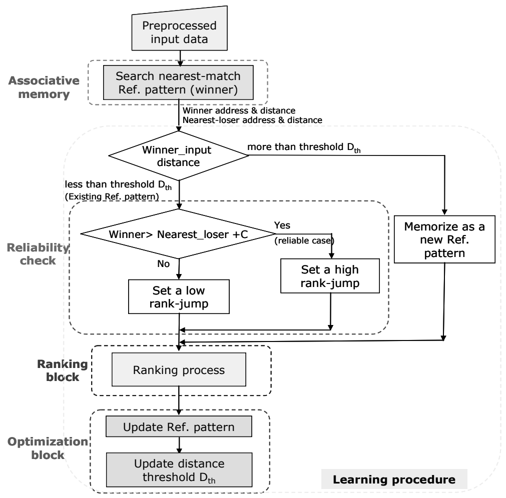

Fig. 1. Flowchart of the learning procedure. C is a constant value selected experimentally based on the data condition.

- **Ranking:** Each reference pattern in the system is given an “unique rank” showing the occurrence level of the pattern. The rank is increased by a predefined jump value in case of a repeated occurrence (when a new input is matched with the current reference pattern) , and reduced gradually when there is no occurrence of matching when and other reference patterns are 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. Figure 2 shows the flowchart of the ranking process. As can be seen from the flowchart, if $D_{w-i}<D_{th}$

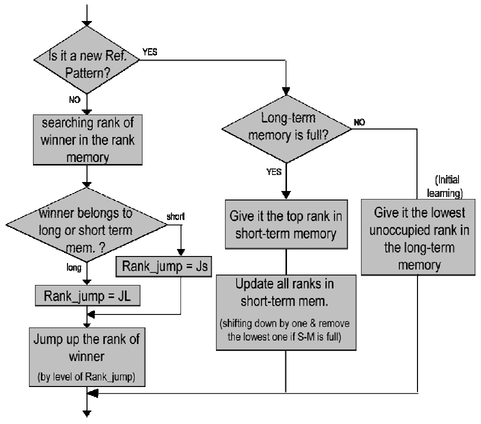
Is it a new Ref. Pattern?
YES
NO
searching rank of winner in the rank memory
Long-term memory is full?
NO
YES
(Initial learning)
winner belongs to long or short term mem. ?
short
long
Rank_jump = Js
Rank_jump = JL
Jump up the rank of winner (by level of Rank_jump)
Give it the top rank in short-term memory
Update all ranks in short-term mem. (shifting down by one & remove the lowest one if S-M is full)
Give it the lowest unoccupied rank in the long-term memory

Fig. 2. Flowchart of the ranking process block.

(i.e. a known Ref. pattern case) then we first search for the existing rank of the winner in the ranking memory.

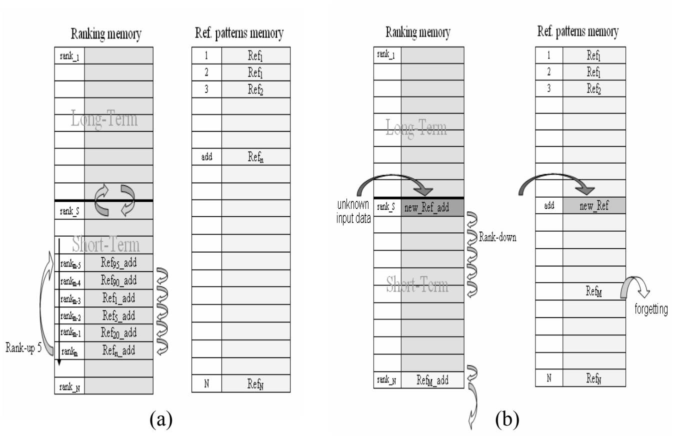

Fig. 3. (a) Rank advancement for a currently existing reference pattern. (b) Giving the top-rank of short-term memory to a new input sample considered as a new reference pattern.

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(a)). The transition between short and long-term memory happens by these changes in rank.

In the case of $D_{w-i} \geq D_{th}$, 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 (Fig. 3(b)).

**Optimization:** 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 generation, 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.

## 2.2 Preprocessing and Feature Extraction

The preprocessing steps include following blocks: *reading, noise removal, binarizing, labeling, segmentation, feature extraction..* We have described the preprocessing steps in detail in our pervious presentations [6]. In the data reading step the data of each text line are scanned continuously as a sequence of thin frames by moving a reading device (scanner sensor) on the text. The frames between each two word spaces are collected and form a larger frame as a gray-scale bitmap array which contains all the word's characters. As most noises appearing in the texts are of the pepper and salt type, we apply a median filter with a neighborhood size of 3×3 for noise removing. The method is hardware-friendly and can be implemented in a pipelined structure. Next, the obtained frame is binarized to a simple black-white bitmap by taking a local threshold value extracted via a mean filter. In the labeling process which comes as the next step, 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. Once the labeling process is terminated, the bitmap memory is scanned once again for the segmentation task. We scan the labeled frame N times each time searching label Li (i=1…N). To have an accurate classification the size of each segmented character is normalized to a fixed size of 16×16 pixels before classification. A bilinear interpolation algorithm is used for resizing the character bitmap. In order to have a robust classification, a number of characteristic features of the input pattern are extracted and grouped in a feature vector. 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*.

## 2.3 Classification

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). We use weighting factors as follows.

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

where in our experiments we found $\alpha$=0.25 and $\beta$=1 as the most effective values.

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 [4] and has a mixed analog-digital fully-parallel architecture for nearest Hamming/ Manhattan-distance search.

## 3. Hardware Realization

As for prototyping of the model including preprocessing, classification, and learning blocks, we have chosen an FPGA platform. 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 block is reported in Table 1. The DSP blocks listed in the table are used for the noise removal block in the preprocessing steps. From Table 1 we can see that only a reasonable amount of logic cells (19%) and memory (38%) are needed for the whole system implementation. An ASIC full-custom design for a simple model of the Ranking block is also achieved in our lab [7,8].

Table 1: 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 % ) |

## 4. Experimental Results

In order to evaluate the learning capability of the system, it was applied for recognition of handwritten English characters. 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 [5] and the results will be presented in our later works.

The simulation results for classification during and after the learning process are reported in Table 2 and 3. 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 2, 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 the system gets to a more stable condition by adjusting continuously the reference pattern memory and distance thresholds, the misclassification rate reduces dramatically. In Table 3 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.

Table 2: Classification results for 4 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 |
| **Total %** | 19.7 | 73.6 | 19.7 | 43.8 | 24 | 38.9 | 22.1 | 37 |

Table 3: Classification results after a period of learning for 832 samples.

| *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 |

Figure 4 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).

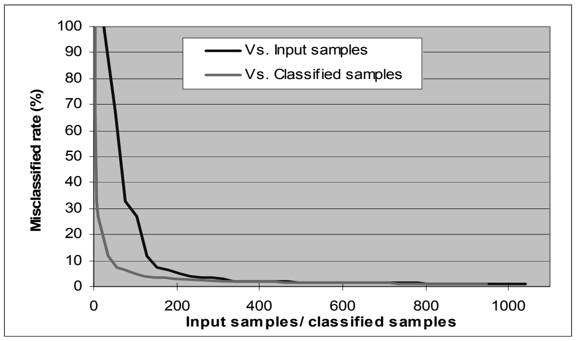

Fig. 4. Changes in the misclassification rate during the learning process.

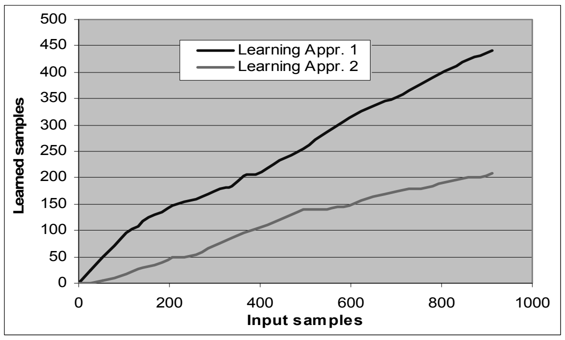

Fig. 5. Learning speed in the experimental tests on handwritten data.

The plot of Fig. 5 shows the learning progress for two approaches, and is based on the handwritten data listed in Table 2. In the first approach, a 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.

## 5. Conclusion

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

## 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] 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.

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

[6] A. Ahmadi, M.A. Ritonga, M.A. Abedin, H.J. Mattausch, and T. Koide, "A Learning OCR System Using Short/Long-term Memory Approach and Hardware Implementation in FPGA," *Proc. of WCCI'2006,* pp. 2702-2708, Canada, 2006.

[7] 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.

[8] 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.