Learning Vector Quantization algorithm as classifier for Arabic handwritten characters recognition

MOHAMED A. ALI[1], KASMIRAN BIN JUMARI[2], SALINA ABD. SAMAD[2]
Computer department[1], Elec., Electronics & System Engineering department [2]
Faculty of Science, Fakulti Kejuruteraan
Sebha University, Universiti Kebangsaan Malaysia
LIBYA, MALAYSIA

Abstract: - In this module, Learning Vector Quantization LVQ neural network is first time introduced as a classifier for Arabic handwriting. Classification has been performed in two different strategies, in first strategy, we use one classifier for all 53 Arabic Character Basic Shapes CBSs in training and testing phases, in second strategy we use three classifiers and three subsets of 53 Arabic CBSs, the three subsets of Arabic CBSs are; ascending CBSs, descending CBSs and embedded CBSs. Three training algorithms; OLVQ1, LVQ2 and LVQ3 were examined and OLVQ1 found as the best learning algorithm.

Key-Words: - Classification, Neural Network, Arabic handwritten recognition, Character Recognition

1 Introduction

Arabic Off-line handwriting character recognition has been a difficult problem to machine learning. It is hard to mimic human classification where specific writing features are utilized. Recent surveys have shown that present technology has still a long way to catch up in terms of robustness and accuracy [1]. Compared with machine-printed character recognition, the prime difficulty in the research and development of handwritten character recognition systems is in the variety of shape deformations [3].

Classification module is one of stages in an Arabic optical character recognition system that we are developing. Classification stage came after preprocessing, segmentation and features extraction. The classification problem can be stated as finding functions which map feature vectors to classes. Preferably these functions should map clusters of same class objects in the feature space to the class space.

2 Classification approaches

Once a feature selection or classification procedure finds a proper representation, a classifier can be designed using a number of possible approaches [4]. In practice, the choice of a classifier is a difficult problem and it is often based on which classifier(s) happen to be available, or best known, to the user. Three different types of classification techniques have been identified.

The first type of pattern classification technique is based on the probabilistic approach. Optimal Bayes decision rule is one example of this approach. The optimal Bayes decision rule assigns a pattern to the class with the maximum posterior probability.

The second type of classification techniques constructs decision boundaries (geometric approach) directly by optimizing certain error criterion. The driving force of the training procedure is, however, the minimization of a criterion such as the apparent classification error or the mean squared error (MSE) between the classifier output and some preset target value.

The third, simplest and the most spontaneous types are based on the concept of similarity by which patterns can be classified by minimum distance classifier using a few prototypes per class, some time called nearest neighbor (NN) classifier. In other words, this type of classifiers models the classes using prototypes and classify according to the shortest distance, defined in an appropriate logic, to a prototype (Lasse et al 1996). The choice of prototypes is crucial for the performance of these types of classifiers. Another version of this type of classifiers is nearest mean classifier. In the nearest mean classifier, selecting prototypes is very simple and reliable; each pattern class is substituted by a single prototype which is the mean vector of all training patterns in that class. Another advanced technique for computing prototypes is unsupervised Neural Network called "Self-Organizing Maps" (SOM), and its supervised version network "Learning Vector Quantization (LVQ) [1].

2.1 Classifier candidate

The main objective in most character recognition applications is to minimize the classification errors, one or combined of the aforementioned techniques is used for this purpose. Now we need to nominate a classifier that fulfills our needs. Basically, the nominated classifier is expected to be compatible with what we have got from the features extraction stage of our OCR system. In addition nominated classifier is supposed to be able to deal with high dimensionality inputs in very fast and accurate manner, and it uses its inputs for learning locally without depending on prior knowledge

In this paper a Learning Vector Quantization (LVQ) network has been chosen to be implemented as classifier for Arabic OCR system due to the aforementioned reasons.

Brief description of Self-Organizing Maps network and detailed architecture and implementation of Learning Vector Quantization (LVQ) network as well as algorithms are given in the following sections.

2.2 Self-organizing Map

Self-Organizing Map (SOM) network is one of the most interesting topics in the neural network field. Such networks are capable of detecting correlations and regularities in their input and adapt their upcoming responses to that input accordingly. The main aim of using a Self Organizing Map (SOM) is to encode a large set of input vectors {x} by finding a smaller set of “representatives” or “prototypes” or “codebook vectors” {wI(x)} that provide a good approximation to the original input space.

2.3 Learning vector quantization

Learning Vector Quantization (LVQ) is a nearest-neighbor method operating essentially in the input domain [2]. It consists of a preset number of processing units, each unit having a d-element reference vector, and each unit being associated with one of the classes of the input samples.

Learning vector quantization (LVQ) is an algorithm for training competitive layers of SOM networks in a supervised manner. A competitive layer automatically learns to classify input vectors. However, the classes that found by the competitive layer are dependent only on the distance between input vectors. If two input vectors are very identical, the competitive layer most likely will put them in the same class. LVQ networks, on the other hand, learn to classify input vectors into target classes chosen by the user. The basic architecture of LVQ network that has a first competitive layer and a second linear layer is shown in Figure 1

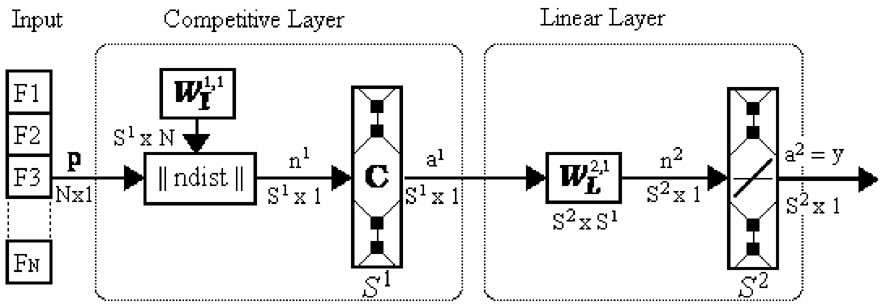

FIGURE 1. Learning Vector Quantization Network

Where: $\mathbf{S}^1$ and $\mathbf{S}^2$ are numbers of competitive and linear neurons respectively.

N: is number of elements in input vector **P** (features: F1, F2 …and FN)

$W_I^{1,1}, W_L^{2,1}$ are competitive and linear weights respectively.

ndist: is a function for calculating near-distance

The competitive layer learns to classify input vectors in much the same way as the competitive layers of SOM networks. The linear layer transforms the competitive layer's classes into target classifications defined by the user. We refer to the classes learned by the competitive layer as *subclasses* and the classes of the linear layer as *target classes*.

Both the competitive and linear layers have one neuron per sub/target class. Thus, the competitive layer can learn up to S1 subclasses. These, in turn, are combined by the linear layer to form S2 target classes. It is worth mentioning here that S1 is always larger than S2.

To conclude the objective of LVQ is to encode a large set of input vectors (x) by finding a smaller set of "representatives" or "prototypes" or "codebook vectors" ($w_i(x)$) that provide a superior approximation to the original input space.

2.3.1 Learning Vector Quantization Algorithms

Learning Vector Quantization utilizes three different learning techniques namely; LVQ1, LVQ2, LVQ3. LVQ1 algorithm has optimized version called OLVQ on which we will give brief description here, further details on other algorithms can be found in Kohonen [3]. OLVQ1 was implemented as a classifier in this research

I. The LVQ1

In this algorithm various domains of the input vector x is approximated by placing a number of 'codebook vectors' m_i (free parameter vectors) in the input space. The quantized values of codebook vectors are used for this approximation. Typically a number of codebook vectors are specified to each class of x values, and x is then decided to belong to the same class to which the nearest m_i belongs. Let the following equation define the nearest m_i to x,

$$c = \arg\min_i \{ \| x - m_i \| \} \qquad (1)$$

The following equations illustrate the mechanism of the basic LVQ1 process:

$$m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$$

if x and m_c belong to the same class,

$$m_c(t+1) = m_c(t) - \alpha(t)[x(t) - m_c(t)] \qquad (2)$$

if x and m_c belong to different classes,

$$m_i(t+1) = m_i(t) \quad \text{for } i \neq c.$$

Where $\alpha(t)$ is individual learning rate. Here $0 < \alpha(t) < 1$, and $\alpha(t)$ may be constant or decrease monotonically with time. In the above basic LVQ1 it is recommended that α should initially be smaller than 0.1 Kohonen [2]; linear decrease in time is used in this research when we used optimized LVQ (OLVQ) as we shall see in the following section.

II. The optimized-learning-rate LVQ1 (OLVQ1)

The basic LVQl algorithm is now modified in such a way that an individual learning rate $\alpha_i(t)$ is assigned to each m_i. and that can approximately be optimized for quick convergence. Then the following discrete-time learning process is obtained. If c is defined by Eq. (1). Then

$$m_c(t+1) = m_c(t) + \alpha_c(t)[x(t) - m_c(t)]$$

if x is classified correctly,

$$m_c(t+1) = m_c(t) - \alpha_c(t)[x(t) - m_c(t)], \qquad (3)$$

if x is classified incorrectly,

$$m_i(t+1) = m_i(t) \quad \text{for } i \neq c.$$

Next, the problem of whether the $\alpha_i(t)$ can be determined optimally for fastest possible convergence of (3) was addressed. If equation (3) is expressed in form of;

$$m_c(t+1) = [1 - s(t)\alpha_c(t)]m_c(t) + s(t)\alpha_c(t)x(t) \qquad (4)$$

where:
$s(t)$= +1 if the classification is correct and
$s(t)$= -1 if the classification is incorrect,
it is noted here that $m_c(t)$ is statistically independent of $x(t)$. It might also be noted how optimal the statistical accuracy of the learned codebook vector values, if the effects of the corrections made at different times are of equal weight. Notice that $m_c(t + 1)$ contains a "trace" from $x(t)$ through the last term in Eq. (4), and "traces" from the earlier $x(t')$, where $t' = 1, 2, \ldots, t-1$ through $m_c(t)$. The (absolute) magnitude of the last "trace" from $x(t)$ is scaled down by the factor $\alpha_c(t)$, and, for instance, the "trace" from $x(t-1)$ is scaled down by: $[1-s(t)\alpha_c(t)]\bullet\alpha_c(t-1)$.
It was made certain that these two scaling factors are identical:

$$\alpha_c(t) = [1 - s(t)\alpha_c(t)]\alpha_c(t-1) \qquad (5)$$

Now if the condition is fulfilled to hold for all t, it can be shown that the "traces" accumulated up to time t from all the earlier x will be scaled down by an equal amount at the end, and thus the "optimal" values of $\alpha_i(t)$ are determined by the recursion

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 + s(t)\alpha_c(t-1)} \qquad (6)$$

After we implemented the optimal version of LVQ in our system, it became obvious how equation (6) really provides fast convergence. However, we noticed that precaution must be made when choosing $\alpha_c(t)$ so that it does not rise above the value 1, the interesting about learning program OLVQ1 that it never allows any α_i to rise above its initial value. It is proved that the initial values of the α_i can be selected rather high, say, 0.3, whereby learning is considerably speeded up, especially in the beginning, and the m_i quickly finds their approximate asymptotic values.

3 Algorithm parameters and implementation

After the brief theoretical background of LVQ algorithm discussed in previous section we need to initialize the network and implement the algorithm utilizing all parameters come with algorithm package provided by Kohonen [1,2]. The LVQ algorithm has been implemented in our Arabic OCR system as illustrated in Figure 2. Classification is performed in two major steps; training and classifying.
In training stage number of parameters like; input and output files, learning rate $\alpha_i(t)$, number of codebook vectors, code-vectors for each class (basic-shape character), number of iterations, number of k-NN and learning rate function, need to be tuned for LVQ network so that optimal recognition rate is obtained.

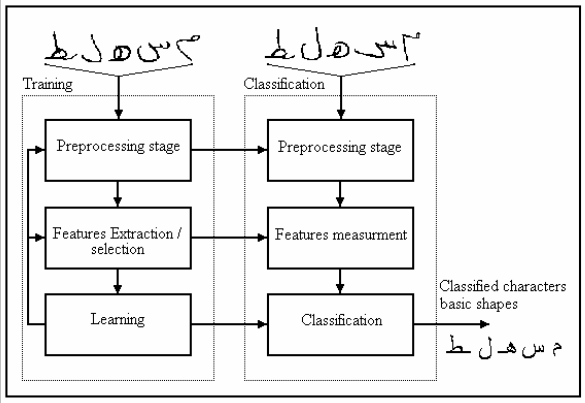

FIGURE 2. Classifier Implementation, training and testing

4. Classifier assessment

The codebook file contains codebook vectors that approximate to the samples of the input vectors, one codebook vector being assigned to each sample. A set of fourteen Arabic character basic-shapes is used to test the classifier, the characters used are; (ی و صـ سـ ر م ل کـ حـ هـ مـ د ب ع). Number of samples for each character is 21 samples. Two different date sets of these samples are prepared, one for training the codebook and the other for testing. Each character image has gone under preprocessing steps we mentioned earlier in section 3.1, before it is fed into feature extraction stage where it is translated into number of features as has been explained in section 3.4.5. The set of features of each character is referred to as a *vector*. Each input vector has a dimensionality of 12 floating point numbers followed by the class label (that can be any string), the label here is representing one of Arabic characters basic-shapes. A number of 21 samples of each character (total of 294 input vectors) are stored in a training data file called *Features1.dat*, and the same numbers are stored in a testing data file called *Features2.dat*. A codebook is initialized with a total number of 200 codevectors. The number of entries selected for each class (character) is check and the medians of the shortest distances are calculated.

It has been noticed that the recognition accuracy depends on the number of codebook entries allocated to each class, and the best distribution of the codebook vectors is not easily predicted. In this experiment we used the method of iteratively balancing the medians of the shortest distances in all classes. Balancing of the medians is achieved by calculating the medians of the shortest distances for each class first, then correcting the distribution so some of those classes which have distances greater than the average entries are added, and some of those classes in which the distance is smaller than the average are deleted, and one learning cycle of the optimized LVQ is automatically run. After this the medians of the shortest distances are computed again.
Now the codebook has been initialized and ready for learning. The codebook is trained by optimized-LVQ, which has been described by Kohonen [3] as the fastest and most robust of all the Learning Vector Quantization algorithms. Number of training run length (training iterations) are tried, and the best found to be 7000 times which can achieve optimal accuracy as shown in Figure 3, it can also be noticed that higher iterations may lead to higher accuracy but the problem it takes more time for convergence. KNN number has been chosen to be 3 which proved to be adequate for accuracy assessment as well as in real practical task of recognition. Figure 4 shows the effect of medians balancing on overall accuracy.

4.1 Stopping rule

It is frequently occurred that the neural network algorithms 'overlearnt'; i.e., when learning and test phases are alternated, the recognition accuracy is initially improved until an optimum is obtained;

later, if further learning is continued, the accuracy starts to decline slowly. A possible explanation in the present case is that when the codebook vectors become very specifically tuned to the training data, the algorithm will not be able enough to generalize for new data. It is hence necessary to halt the learning process after some 'optimal' number of steps, it is empirically found as 50 to 200 times the total number of the codebook vectors (depending on particular). Such a stopping rule can only be found by experience, and it depends on the input data, algorithm being used and learning

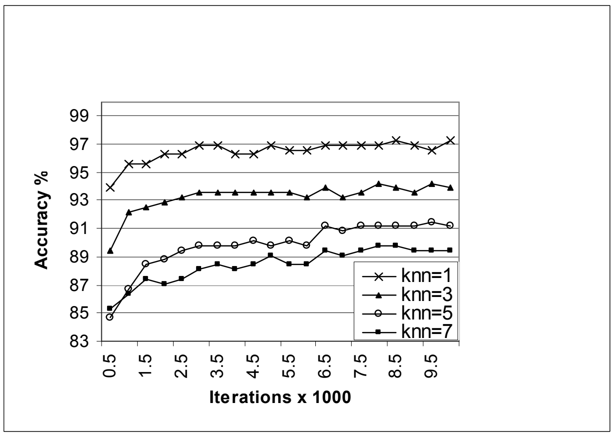

FIGURE 3. The Effect of Iterations and knn on Recognition Accuracy

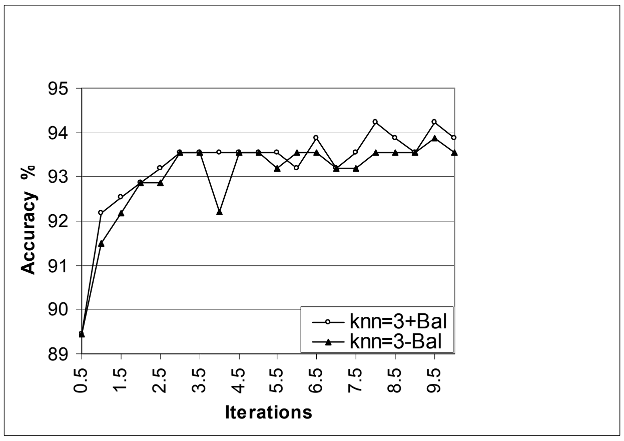

FIGURE 4. The Effect of Medians Balancing on Overall Accuracy,
± means with or without Balancing

It can be clearly noticed that characters like 'Ain', 'Kaf' and 'Hha' are 100% recognized whereas character like 'Waw' is only 80% recognized, other character have satisfactory recognition. Some measures have been taken in next section to solve some of these misrecognition cases.

Further investigations will be carried out when we implement the classifier in our complete system where the complete set of Arabic characters basic-shapes is divided into three categories; Ascenders, Descenders and Embedded, and the classifier is trained for each of them individually.

5 Classification strategies

5.1 First strategy

In this strategy, as we mentioned earlier, all 53 Arabic CBSs are used as one sets for training and testing phases for one LVQ network. A total of 21 samples of each CBS are used in learning and testing steps, 11 samples of each CBS are taken as training set whereas the rest 10 samples of each CBS are used as testing set. Each CBS of training and testing sets are translated into code-vector by features extraction module and stored into two different data file, one data file contains code-vectors of all CBSs belong to training set, the other data file contains code-vectors of all CBSs belong to testing set. We started by initialization of codebook-vectors, then the medians of the shortest distances between the initial codebook vectors of each class are computed (balancing), then we trained the LVQ network by specifying different combinations of learning parameters like; input and output files, learning rate α i(t), number of codebook vectors, code-vectors for each class (CBS), number of iterations and number of k-NN. Finally we tested the network and monitor the recognition accuracies for each individual class (CBS) as well as the classifier overall recognition rate. The overall recognition accuracy of this classifier is 83.2 %.

5.2 Second strategy

This strategy is based on numbers of hypothesis and notions we started with. First, if we take the basic shapes (53 CBSs) the possibility of finding two or more of these basic shapes being morphologically close to each other is not ignorable. Second, the more the number of classes to be classified the more the complication of network design and programming are claimed. Third, tasks subdivision facilitates parallel programming, fast debugging and system development. Based on what we mentioned here we decided to use three classifiers instead of one, we divided the 53 CBSs into three groups based on how each CBS located in a word with respect to base-area in the text-line, the three groups are ascenders, descenders and embedded, and accordingly the three classifiers are; ascenders classifier, descender classifier and embedded classifier. The same training and testing procedures followed in case of first strategy is applied for each individual classifier in this strategy, the only difference here is that the database set used in the first strategy is divided into three groups as we explained in this section, and each group is fed into features extraction module (so each CBS is translated into set of codes 'codevector') before it is fed into its corresponding classifier. During training phase the data is fed manually into features extraction module and then the output of this module is fed into classification module where a vectors codebook is created to be used in testing and real recognition phases will takeover controlling of different processes flow. In this stage, in particular, CBSs are sequentially retrieved from word/subword-to-characters module and fed into features extraction module first, then according to knowledge source CP-KS it is fed to either ascenders, descenders or embedded classifier. The classified CBS is then fed to the next stage (composer) for further processing. Each individual classifier is trained and tested with its correspondent dataset. The overall recognition accuracies of ascenders, descenders and embedded classifier are 92.21 %, 88.24 % and 83.2 % respectively. The average recognition rate of the three classifiers is 89.1 % which is higher than the recognition rate of the first strategy classifier (which is 83.83 %). It was concluded that as the number of CBS classes' increases the recognition accuracy decreases, and this reassure our presumption of dividing the CBSs set into three categories and using three classifiers, one classifier for each category. In our opinion, that there are two fundamental sources of misclassification for our classifier. The first one is the low number of available CBSs samples for training and testing. The second is the intrinsic ambiguity in Arabic cursive characters. Characters in a handwritten word often have flourish and ligature strokes which generally do not appear in independently written characters. These extra strokes are hard to be generalized in common prototypes, since they are usually unique to the writing style. The training character set can not cover all the variations of these strokes even if pre-isolated characters from word images are used for training because of the uniqueness of these extra strokes.

6 Conclusion

We have presented a new method for Arabic handwritten character recognition using a reliable features extraction module and LVQ neural network technique as classifier. LVQ is one of the best clustering techniques. It has been shown that the proposed method is more effective than the conventional matching methods used in OCR systems. This method is robust with regard to geometrical variation, but very sensitive to topological variations such as the presence of spurious small branches, loops opening or strokes closing etc

References:

[1] Cao, J. Ahmadi, M. & Shirdhar, M 1995. Recognition of handwritten numerals with multiple feature and multistage classifier. *Pattern Recognition*. 28(2): 153-160.

[2] Kohonen, T. 1996. New developments and applications of self-organizing maps Neural Networks for Identification. Control, *Robotics, and Signal/Image Processing. Proceedings., International Workshop on*. pp. 164-172.

[3] Kohonen, T. Barna, G. Chrisley, R. 1988. Statistical pattern recognition with neural networks: benchmarking studies. *Neural Networks. IEEE International Conference on*. 1: pp. 61-68

[4] Yu, D. Yan, H., Separation of single-touching handwritten numeral strings based on structural features, *Pattern Recognition*, Vol. 31, No. 12 1998, pp. 1835-1847.

[5] Liwei W., Xiao W. and Jufu F., On image matrix based feature extraction algorithms, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 36, No. 1, 2006, pp. 194-197

[6] Kenneth E. H., Deniz E., Kari T. and Jose C. P., Feature Extraction Using Information-Theoretic Learning, *IEEE Transactions on Pattern Analysis and Machine Intelligence* Vol. 28, No. 9, 2006, pp.1385-1392.