Hand-Written Character Recognition Using Kohonen Network

Dr. Pankaj Agarwal
Dept. of Computer Sc, IMS Engineering College, Ghaziabad, U.P, India

Abstract
This paper presents a simple learning rule for recognition of mouse dragged character on our computer screen using artificial neural network. We use Kohonen self organization map for pattern classification which employs unsupervised learning algorithm. The results are quite encouraging in terms of percentage of characters being successfully recognized. One advantage of proposed scheme is that the system is quite tolerant to changing conditions and inputs. The system consistently learns. Moreover the recognition ratio is excellent in the proposed system.

Keywords
Artificial Neural networks, Kohonen network, paper, hand written character recognition, mouse dragging etc.

I. Introduction

Character recognition is the process to classify the input character according to the predefine character class. With increasing the interest of computer applications, modern society needs the input text into computer readable form. This research is a simple approach to implement that dream as the initial step to convert the input text into computer readable form. Some research for hand written characters are already done by researchers with artificial neural networks. In this paper we use Kohonen neural network. A net work, by its self organizing properties, is able to infer relationships and learn more as more inputs are presented to it [1].

The Kohonen Self-Organizing Map (SOM) designed by Tuevo Kohonen is a variation of the traditional Artificial Neural Network. It is a third generation neural network, meaning that many of its functional characteristics are thought to mirror those found in biological fact. An SOM consists of a collection of nodes of neurons that are each connected to every other node and each node has associated with it a set of input weights w. The SOM also has associated with it a metric for determining which nodes are in the neighborhood N of a given node.

When the network is presented with a vector xi at its input, it computes the neural response s_j of the node j using the formula:

$$S_j = w_j * x_i \quad (1)$$

Normalize both wj and xi before computing the dot product, sj, and refer to the node that produces the largest value of s as node k. Since the dot product of the normalized wk and xi vectors is the cosine of the angle between them, we can conclude that the winning node is the one with the weight vector closest to the input vector in its spatial orientation. We can then say that node k giving the largest s is closest to recognizing the input vector. We allow the nodes to learn by applying a Δw to their weights using the formula:

$$\Delta w_k = \alpha(x_i - w_k) \quad (2)$$

Where α is a constant in the range [0,1] called the learning constant. The learning process is applied to the maximum response neuron and neurons in its defined neighborhood.

This training process can be described by the following algorithm [1]:

1. A cycle: for every input vector x_i

[a] Apply vector input to the network and evaluate the dot products of the normalized weights on each node and a normalized input vector. Call these dot products s.

[b] Find the node k with the maximal response sk.

[c] Train node k, and all the nodes in some neighborhood of k, according to the learning equation above.

[d] Calculate a running average of the angular distance between the values of wk and their associated input vectors.

[e] Decrease the learning rate, α.

2. After every M cycles, called the period, decrease the size of the neighborhood N.

3. Repeat steps 1-2 for some finite period of time or until the average angular distance. One advantage to this scheme is that the system is quite tolerant to changing conditions and inputs. The system consistently learns. Moreover the recognition ratio is excellent in the proposed system.

II. The Proposed System

The overall method of the implemented system is illustrated in fig.1 :

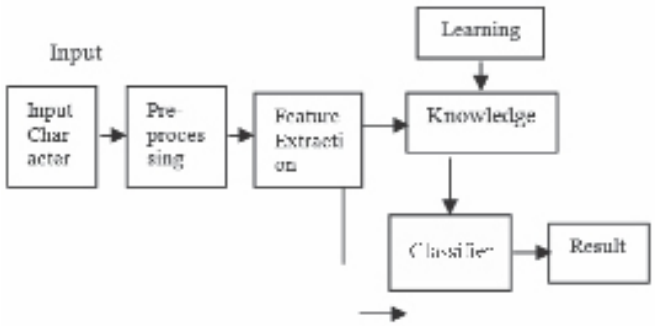

Fig.1: Overall model of implemented System

A. Input Image

The input to the recognition system is acquired by simply dragged the mouse on the text screen. Ignoring the concept of colored paper or character, the black part of the image is considered as the character and the white part is considered as the paper.

B. Feature Extraction and Preprocessing

Feature extraction is the process of extracting essential information contain from the image segment containing a character. It plays a vital role in the whole recognition process. This effectively reduces the number of computation and hence reduce the learning time in the training session of the neural network and faster the recognition process [1, 4].

C. Drawing Images

Though not directly related to neural networks, the process by which the user is allowed to draw the characters on the computer screen using mouse dragging is an important aspect this paper. Most of the actual drawing is handled by the process MouseMotionEvent.

If the mouse is being drug, then a line will be drawn from the last reported mouse drag position to the current mouse position. It is not enough to simply draw a dot. The mouse moves faster than the program has time to accept & process all values for. By drawing the line, we will cover any missed pixels as best we can. The line is drawn to the off-screen image, and then updated to the users screen. This is done with the following lines of code.

```
entryGraphics.setColor(Color.black);
entryGraphics.drawLine(lastX,lastY,e.getX(),e.getY());
getGraphics().drawImage(entryImage,0,0,this);
lastX = e.getX();
lastY = e.getY();
```

As the program runs, this method is called repeatedly. As a result whatever the user is drawing is saved in the off-screen image. Next section demonstrates how to down sample an image. The off-screen image is accessed as an array of integers, allowing us to work on the image data directly.

Down-sampling the Image: Every time a letter is drawn for either training or recognition, it must be down-sampled. In this section we will examine the process by which this down-sampling occurs. However, before we discuss the down-sampling process, we should discuss how these down-sampled images are stored.

When you draw an image, the first thing the program does; it draws a box around the boundary of your letter. This allows the program to eliminate all of the white space around your letter. This process is done inside of the down-sample method of the Entry. java class. As you drew a character this character was also drawn onto the entryImage instance variable of the entry object. In order to crop this image, and eventually down-sample it, we must grab the bit pattern of the image. This is done using a PixelGrabber class as shown here.

```
int w = entryImage.getWidth(this);
int h = entryImage.getHeight(this);
PixelGrabber grabber = new PixelGrabber (entryImage, 0, 0, w,
h,true);
grabber.grabPixels();
pixelMap = (int[])grabber.getPixels();
```

After this code completes, the pixelMap variable, which is an array of int datatypes, now contains the bit pattern of the image. The next step is to crop the image and remove any white space. Cropping is implemented by dragging four imaginary lines from the top, left, bottom and right sides of the image. These lines will stop as soon it crosses an actual pixel. By doing this, these four lines snap to the outer edges of the image. The hLineClear and vLineClear methods both accept a parameter that indicates the line to scan, and returns true if that line is clear.

Performing the Down-sample: Now that the cropping has taken place, the image must be actually down-sampled. This involves taking the image from a larger resolution to a 5X7 resolution. To see how to reduce an image to 5X7, think of an imaginary grid being drawn over top of the high-resolution image. This divides the image into quadrants, five across and seven down. If any pixel in a region is filled, then the corresponding pixel in the 5X7 down-sampled image is also filled it. Most of the work done by this process is accomplished inside of the "downSampleQuadrant" method. This method is shown here.

```
Protected boolean downSampleQuadrant(int x, int y)
{
  int w = entryImage.getWidth(this);
  int startX = (int)(downSampleLeft+(x*ratioX));
  int startY = (int)(downSampleTop+(y*ratioY));
  int endX = (int)(startX + ratioX);
  int endY = (int)(startY + ratioY);

  for ( int yy=startY;yy<=endY;yy++ )
  {for ( int xx=startX;xx<=endX;xx++ )
  {int loc = xx+(yy*w);
  if ( pixelMap[ loc  ]!= -1 )
  {return true;}
  {return false;}
```

The downSampleRegion method accepts the region number that should be calculated. First the starting and ending x and y coordinates must be calculated. To calculate the first x coordinate for the specified region first the downSampleLeft is used, this is the left side of the cropping rectangle. Then x is multiplied by "ratioX", which is the ratio of how many pixels make up each quadrant. This allows us to determine where to place startX. The starting y position, start Y, is calculated by similar means. Next the program loops through every x and y covered by the specified quadrant. If even one pixel is determined to be filled, then the method returns true, which indicates that this region should be considered filled. The downSample Region method is called in succession for each region in the image. This results in a sample of the image, stored in the SampleData class. The class is a wrapper class that contains a 5X7 array of Boolean values. It is this structure that forms the input to both training and character recognition.

III. Learning and Recognition (using Kohonen Self Organization Map)

The Kohenen network has two layers, an input layer and a Kohonen out layer. The input layer is a size determined by the user and much match the size of each row (pattern) in the input data file. A kohonen feature map may be used by it self or as a layer of another neural network. A kohonen layer is composed of neurons that compete with each other. The kohonen SOM use winner take all strategy. Inputs are feed into each of the neurons in the kohonen layer (from the input layer). Each neuron determines its out put according to a weighted sum formula: Output = $\sum$ wij xj The weights and the inputs are usually normalized which mean that the magnitude of the weight and input vectors are set equal to one. The neuron with the largest output is winner. The neuron has a final output of 1. All other neurons in the layer have an output of zero. Different input patterns end up with firing different wining neurons. Similar or identical input patterns classify to the same output neuron. Only winning neurons and their neighbor's par in learning for a given input pattern [3].

A. How a Kohonen Network Learns

There several steps involved in this learning process. Overall the process for training a Kohonen neural network involves stepping through several epochs until the error of the Kohonen neural network is below acceptable level. The training process for the Kohonen neural network is competitive. For each training set one neuron will "win". This winning neuron will have its weight adjusted so that it will react even more strongly to the input the next time. As different neurons win for different patterns, their ability to recognize that particular pattern will be increased [3].

Learning Rate: The learning rate is a constant that will be used by the learning algorithm. The learning rate must be a positive number less than 1. Typically the learning rate is a number such as .4 or .5. Generally setting the learning rate to a larger value will cause the training to progress faster. Though setting the learning rate to too large a number could cause the network to never converge. This

is because the oscillations of the weight vectors will be too great for the classification patterns to ever emerge. Another technique is to start with a relatively high learning rate and decrease this rate as training progresses. This allows initial rapid training of the neural network that will be "fine tuned" as training progresses. The learning rate is just a variable that is used as part of the algorithm used to adjust the weights of the neurons.

B. Adjusting Weights

The entire memory of the Kohonen neural network is stored inside of the weighted connections between the input and output layer. The weights are adjusted in each epoc. An epoch occurs when training data is presented to the Kohonen neural network and the weights are adjusted based on the results of this item of training data. The adjustments to the weights should produce a network that will yield more favorable results the next time the same training data is presented. Epochs continue as more and more data is presented to the network and the weights are adjusted. Eventually the return on these weight adjustments will diminish to the point that it is no longer values to continue.

IV. Proposed Algorithm

A. Initialize network

For each node I set the initial weight vector $W_i(0)$ to be random. Set the initial neighborhood $N_i(0)$ to a large value.

B. Present input

In this we fed input in the form of binary pixels of 1 for white and zero for black pixel. As a result, the program feeds it the value of 0.5 for a white pixel and -0.5 for a white pixel. This array of 35 values is fed to the input neurons. This is done by passing the input array to the Kohonen's "winner" method. This will return which of the 35 neurons won, this is stored in the "best" integer. Calculating winning node Calculating winning node c based on the maximum activation among all p neurons participating
In a competition $C= \max\sum W_{ij} X_i$
So the neuron with the largest activation is the winner. The neuron has the final output of 1 or this is the firing neuron. All other neurons in the layer have an output of zero.

C. Update weights

The original method for calculating the changes to weights, which was proposed by Kohonen, is often called the additive method. This method uses the following equation.

$$w^{t+1} = \frac{w^t + ax}{\left\| w^t + ax \right\|}$$

The variable x is the training vector that was presented to the network. The variable w^t is the weight of the winning neuron, and the variable w^{t+1} is the new weight. The double vertical bars represent the vector length.

D. Training the Sample Program to Recognize Letters

The program may not be able to recognize any one's handwriting because it is initially trained for particular handwriting only. Two choices have been provided as to how to train the neural network program. First, you can choose to start from a blank training set and enter all 26 letters for yourself. You can also choose to start from my training set. If you start from my training set you can replace individual letters. This would be a good approach if the network were recognizing most of your characters, but failing on a small set. You could retrain the neural network for the letters that the program was failing to understand. To delete a letter that the training set already has listed you should select that letter and press the "Delete" button on the OCR application. Not that this is the GUI's "Delete" button and not the delete button on your computer's keyboard. To add new letters to the training set you should draw your letter in the drawing input area. Once your letter is drawn you can click the "Add" button. This will prompt you for the actual letter that you just drew. What ever character you type for this prompt will be displayed to you when the OCR application recognizes the letter that you just drew. Now that you have your training set complete you should save it. This is done by clicking the "Save" button on the OCR application. This will save the training set to the file "sample.dat". If you already have a file named sample.dat, it will be overwritten. Because of this it is important to make a copy of your previous training file if you would like to keep it. If you exit the OCR application without saving your training data, it will be lost. When you launch the OCR application again you can now click "Load" to retrieve the data you previously stored to the sample.dat file.

V. Experimental Result

The complexity of the problem is greatly increased by noise in data and by an almost infinite variability of hand writing as a result of the writer and the nature of the writing .A single letter may be written & represented in many ways as shown below for character 'A', 'D' & 'E'. The level of complexity is more when the character is to be drawn on computer screen using mouse

.

Fig. 2:

Following table depicts the percentage/rate of success in recognition of above drawn characters in different styles.

Table 1:

Character Drawn	No of Patterns Given	#Recognized	#Not Recognized	Rate (%) of Recognition
A	5	4	1	80
D	5	4	1	80
E	5	4	1	80

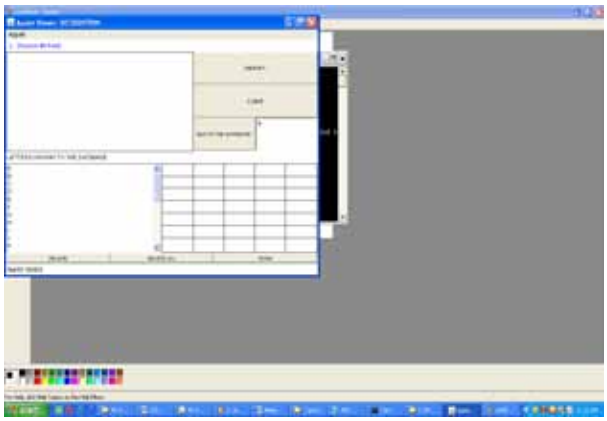

Fig. 3:

IV. Conclusion

Recognition of handwritten characters has been a considerable interest to researchers working on OCR. The complexity of the problem is greatly increased by noise in data and by an almost infinite variability of hand writing as a result of the writer and the nature of the writing. Here we have developed a generalized recognition system for hand written character. The rate of success has been found to be between 75% to 80% which is fairly significant. Some time humans can't even recognize their own handwritings and the handwritten character varies from man to man and depends on many factors i.e. emotion, pen pressure, and environment. This is why; it is too difficult to get accurate efficiency. Though it is problematic if a man follows standard writing rules, the filtering and feature extraction is done more accurately, and then it is possible to recognize the handwritten text into computer readable form in more precise manner.

References

[1] O, Trier, A.K. Jain, T. Taxt. "Feature extraction methods for character recognition", pattern recognition, 29(4): 641-662, 1996

[2] Vallu Rao, Hayagriva Rao, "C++ Neural Networks and Fuzzy logic". International Edition, 1996

[3] Vuokko Vuori, Erkki Oja, "Analysis of different writings styles with the self organizing map". In Proceedings of the 7th International Conference on neural information processing. Vol. 2., 2000, pp 1243-1247. November 2000

[4] Davis, R.H., Lyall, "Recognition of hand written character – A review", Image and vision computing 4,4, (1986) 208-218

Pankaj Agarwal received his doctorate from Jamia Millia Islamia Central University, New Delhi, India. His research interests include Soft Computing, Computational Algorithms & Bioinformatics. He has published more than 25 research papers in various international journals & conferences of repute & has 5 books to his credit. He is presently associated with IMS Engineering College, Ghaziabad, U.P, India in Department of Computer Science & Engineering as Professor & Head & Dean , Room No A-108