Handwritten Character Recognition using Neural Networks

Sunith Bandaru

Abstract

In this paper, we develop a multi-layered neural network based algorithm using which a computer can learn to identify handwritten characters. A GUI is designed in MATLAB which enables the user to either train or test the network on a 'one character at a time' basis. We have used a six element feature vector which is found to be sufficient for reliably identifying all characters that can be entered using a standard US English QWERTY keyboard.

1 Introduction

Optical Character Recognition (OCR) is a very well-studied problem in the vast area of pattern recognition. Its origins can be found as early as 1870 when an image transmission system was invented which used an array of photocells to recognize patterns. Until the middle of the 20th century OCR was primarily developed as an aid to the visually handicapped. With the advent of digital computers in the 1940s, OCR was realized as a data processing approach for the first time. The first commercial OCR systems began to appear in the early 1950s and soon they were being used by the US postal service to sort mail. According to Wikipedia "The accurate recognition of Latin-script, typewritten text is now considered largely a solved problem on applications where clear imaging is available such as scanning of printed documents. Typical accuracy rates on these exceed 99%; total accuracy can only be achieved by human review. Other areas including recognition of hand printing, cursive handwriting, and printed text in other scripts (especially those with a very large number of characters) are still the subject of active research."

The major schemes in OCR starting from the relatively easier to the most difficult are as follows [3]:

(i) *Fixed-font character recognition* is the recognition of specific fonts (Ariel, Courier, etc.) of typewritten characters.

(ii) *On-line character recognition* is the recognition of single hand-drawn characters where not only the character image is provided but also the timing information of each stroke.

(iii) *Handwritten character recognition* is the recognition of single hand-drawn characters of an alphabet which are unconnected and not written in calligraphy.

(iv) *Script recognition* is the recognition of unrestricted handwritten characters which may be connected and cursive.

Standard pattern recognition methodologies that have been successfully applied to OCR include point by point global comparisons, global transformations, extraction of local properties, template matching, analysis by means of curvatures and structural methods. However, the application of these methods for handwritten character recognition is discussable because of the infinite variations of character shapes resulting from writing habits, style, education, region of origin, social environment, mood, health and other conditions of the writer. Moreover, factors such as the writing instrument, writing surface, scanning methods, etc. also drastically affect the efficient of standard character recognition algorithms.

An obvious way to handle these variables is to design an algorithm which instead of relying on a pre-specified rule set for recognizing characters, 'learns' to identify them by example. This is known as supervised learning. Neural networks provide the simplest of platforms for realizing the supervised learning task. A thorough discussion on the application of neural networks for character recognition can be found in [2].

2 Multi-layered Neural Network

For the present work, we have used a multi-layered neural network with two hidden layers each containing ten neurons. In addition, we have an input layer containing six neurons (one for each input) and a single output neuron which represents the character. The network is fully connected and its weights are updated using the gradient descent method i.e,

$$w^{new} = w^{old} - \eta \left(\frac{\partial E}{\partial w} \right), \tag{1}$$

$$\text{where} \qquad E = \frac{1}{2} \sum_{i_3} (y^d_{i_3} - y_{i_3})^2.$$

The back propagated errors are calculated at each layer and using the *generalized delta rule*, the following weight update formulae are obtained for the network [1].

$$W_{i_3,i_2}(t+1) = W_{i_3,i_2}(t) + \eta \delta_{i_3} v_{i_2}, \tag{2}$$

$$\text{where} \qquad \delta_{i_3} = y_{i_3}(1 - y_{1_3})(y^d_{i_3} - y_{i_3}).$$

$$W_{i_2,i_1}(t+1) = W_{i_2,i_1}(t) + \eta \delta_{i_2} v_{i_1}, \tag{3}$$

$$\text{where} \qquad \delta_{i_2} = v_{i_2}(1 - v_{1_2}) \sum_{i_3} \delta_{i_3} W_{i_3,i_2}.$$

$$W_{i_1,i_0}(t+1) = W_{i_1,i_0}(t) + \eta \delta_{i_1} x_{i_0}, \tag{4}$$

$$\text{where} \qquad \delta_{i_1} = v_{i_1}(1 - v_{1_1}) \sum_{i_2} \delta_{i_2} W_{i_2,i_1}.$$

The output v at each neuron is calculated using the Sigmoidal activation function given by,

$$v = \frac{1}{(1 + e^{-h})}, \tag{5}$$

where h is the original output.

For our network, $i_0 = \{1, 2, 3, 4, 5, 6\}$, $i_1 = \{1, 2, \ldots, 9, 10\}$, $i_2 = \{1, 2, \ldots, 9, 10\}$ and $i_3 = \{1\}$. The learning rate η is taken as 0.5.

3 Implementation

The back propagation algorithm described above requires a numerical representation for the characters. As far as the desired output $y^d_{i_3}$ of the network is concerned, we use the ASCII value of the character for training. When a particular character is queried, the networks returns an ASCII value y_{i_3} which can then be converted into the corresponding character. However, the learning necessitates that for each training input, the shape of the character being entered be associated with the character (and hence the ASCII value) provided by the user. To this end, we also need to have a numerical representation for the shape of the character. A vector containing certain numerical features of the pattern (or character in our case) is known as the feature vector. In this paper, we have used six different features of the character as inputs to the neural network. Before discussing these features, we describe how an inputted character is represented within the MATLAB code.

In the GUI developed as part of this work, first a square drawing area is presented to the user. The user can now draw a single character using a mouse and close the drawing area. Next, a scaled binary image of the entered character is generated on a 100×100 pixel grid. This binary image contains 1s in the pixels over which the character was drawn and 0s in all other pixels. Thereafter, the binary image is used to create a gradient map of the inputted character as follows:

(i) The values of the pixels immediately surrounding a pixel with a value of 1 are incremented by 0.8.

(ii) The values of the pixels in next immediate proximity are incremented by 0.6.

(iii) This is carried on for two more layers of pixels and the values are incremented by 0.4 for the inner layer and by 0.2 for the outer one.

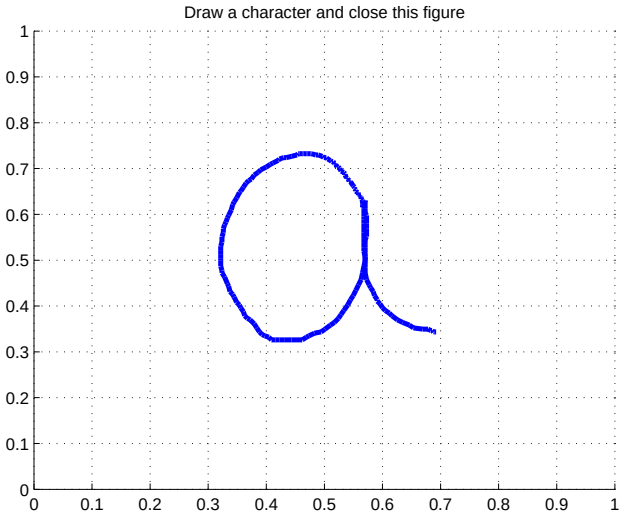

Figure 1: Drawing box

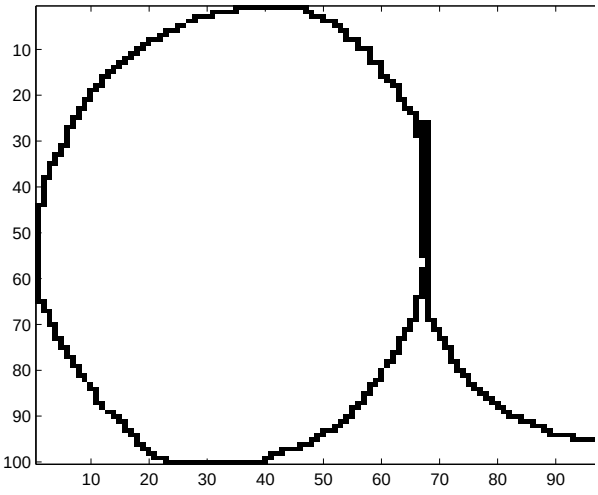

Figure 2: Binary image

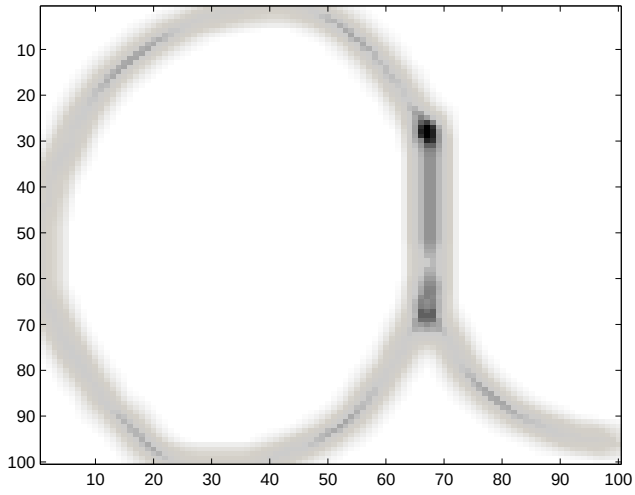

Figure 3: Gradient map

Thus, a pixel with value 1 surrounded entirely by pixels of value 0 would look as follows after the above given operations:

0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
0.2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.2
0.2	0.4	0.6	0.6	0.6	0.6	0.6	0.4	0.2
0.2	0.4	0.6	0.8	0.8	0.8	0.6	0.4	0.2
0.2	0.4	0.6	0.8	**1.0**	0.8	0.6	0.4	0.2
0.2	0.4	0.6	0.8	0.8	0.8	0.6	0.4	0.2
0.2	0.4	0.6	0.6	0.6	0.6	0.6	0.4	0.2
0.2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.2
0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2

Figure 1 shows the drawing box provided to the user. Figure 2 shows the binary image and Figure 3 shows the corresponding gradient map.

Once this gradient map is obtained, the following features are extracted from it.

(i) x_1: Sum of all the pixel values.

(ii) x_2: Sum of the values within pixels that are to the left of the vertical center line.

(iii) x_3: Sum of the values within pixels that are to the right of the vertical center line.

(iv) x_4: Sum of the values within pixels that are above the horizontal center line.

(v) x_5: Sum of the values within pixels that are below the horizontal center line.

(vi) x_6: The ratio of length to the width of the character.

Features x_2 and x_3 help distinguish between characters which are vertically symmetric. Eg: 'b' and 'd'. Similarly, features x_4 and x_5 help distinguish between characters that are horizontally symmetric. Eg: 'b' and 'p'. To differentiate between centrally symmetric characters like 'o' and '0', the last feature is used. x_1 serves as a general purpose feature.

After obtaining the feature vector $\mathbf{F} = [x_1\ x_2\ x_3\ x_4\ x_5\ x_6]$, the user is given a choice to either train the network or make a query. If the user chooses to train, a prompt appears and asks the user to enter the character using a keyboard. The ASCII value of this character becomes $y_{i_3}^d$ for the network. The network is now trained for the input $\mathbf{F}$ and the output $y_{i_3}^d$ using 100 epochs.

References

[1] L. Behera and I. Kar. *Intelligent systems and control: Principles and applications.* Oxford University Press, India, 2009.

[2] C.M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, USA, 1995.

[3] J. Mantas. An overview of character recognition methodologies. *Pattern recognition*, 19(6):425–430, 1986.