Character Recognition

Using Neural Networks

Randy Dimmett
Semester Project
EE 368
May 9th, 2000

Purpose

The purpose of this project is to develop and test a neural network that will recognize letters of the alphabet in a scanned image.

Background

The need for character recognition software has increased much since the astounding growth of the Internet. With the Internet, the demand for online information has meant that what was once on paper is now desired in a digital format.

Character recognition, as defined in this project, consists of scanning in a page of typed information, and pulling out the recognizable characters of the alphabet and saving these characters to a file. In other words, this project is an attempt to take the image from a scanner and turn it back into the words that were scanned in to begin with. All training and testing inputs were in bitmap format (.bmp) because this is a very common way to save images that have been scanned.

For the purposes of this project, a fixed-point font was used for generating the sample data and for testing the final neural network. A fixed-point font is one in which all the characters are effectively the same width, regardless of the actual size of the letters, numbers or symbols in the font.

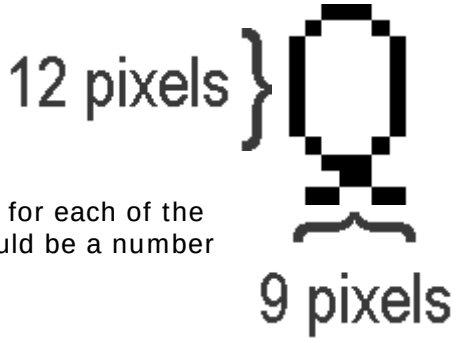
Pre-processing was needed to turn a scanned image into network-ready inputs. Matlab was used to read in an image that contained a sentence of text. Then Matlab reduced the image to a black and white image (binary image), and then to a matrix of ones and zeros, where ones indicated white pixels and zeros the black pixels.

Preparation of Data

The first part of the project consisted on gathering sample data and targets to train the neural network with. In this project, the 12 pt. Courier New font was used to generate the capital letters of the alphabet, and also an empty space. The character set in figure 1 was saved in .bmp format and given to the neural network to use for training.

Figure 1. Courier New Training Set

Each letter served as an input having 108 attributes. See figure 2 for a sample character from the Courier New font family having 12*9 attributes.



A normalized vector from 1 to 27 defined the targets for each of the 27 inputs. Therefore, the output for the network would be a number between 0 and 1, with 27 possible values.

Figure 2. Courier New font Sample

Next, an ideal word was created and saved in bitmap format for testing the network, just to make sure Matlab was simulating the network correctly and that the network was at least working with the training data.

The word 'PERCEPTRON' was used for testing the network to make sure the training was successful. Figure 3 shows how the bitmap looked that Matlab received.

Figure 3. Ideal test data

Then, non-ideal data from a scanner was used for testing the network. This non-ideal data was typed and printed out and then scanned back in to simulate the real-world process of scanning in a page of text. Figure 4 contains a close-up of a piece of scanned data.

Figure 4. Non-Ideal sample

After receiving a non-ideal input such as the one in figure 4, Matlab has to convert it to a black and white image. After conversion to a binary image, much information is lost and the letters also appear noisy. The scanned data looks like that in figure 5.

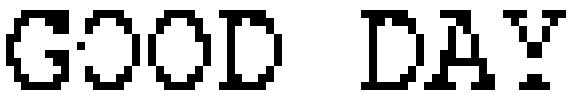
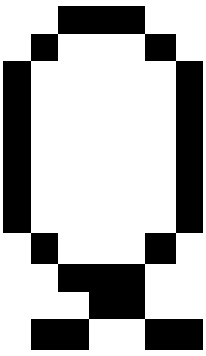


Figure 5. Non-Ideal black and white sample

Then, Matlab converts the black and white image to a matrix of ones and zeros. For example, the letter 'Q' can be spotted in the matrix after being converted:



```
1 1 1 0 0 0 1 1 1
1 1 0 1 1 1 0 1 1
1 0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0 1
1 0 1 1 1 1 1 0 1
1 1 0 1 1 1 0 1 1
1 1 1 0 0 0 1 1 1
1 1 1 1 0 0 1 1 1
1 1 0 0 1 1 0 0 1
```

Figure 6. Binary Matrix Representing Q

Architectures Tested

For all the architectures used, there were 27 input vectors each having 108 attributes.

Linear Associator With Pseudoinverse Rule

The first architecture that was used to attempt character recognition was the *Linear Associator* using the Pseudoinverse rule. The Pseudoinverse was used instead of the Hebb rule because the prototype patterns were not orthogonal. The Pseudoinverse rule was preferred over other learning rules because of its simplicity. The weight matrix for the *linear* associator using the Pseudoinverse rule can be found using the following matrix equation:

$$\mathbf{W = TP^{+}}$$

Where P^{+} *is the pseudoinverse defined by* $\mathbf{P^{+} = (P^{T}P)^{-1}P^{T}}$

After forming the input matrix P, and corresponding target matrix T, the weight matrix was easy to calculate. Because of the rule's simplicity, changing the weight matrix for a new set of fonts would be quick enough to do on-the-fly.

The Linear Associator gave better results than any other network tested, so this was the one chosen in the final version of the project.

4-Layer Networks With Backpropagation Algorithm

Several different architectures were experimented with, starting with a 4-layer network having 12 neurons in the first 2 layers, 2 neurons in the 3rd layer, and 1 neuron in the 4th layer. With all the transfer functions as tangent sigmoid, the ideal data was loaded and the network converged to a minimum error after about 50 epochs. The network was tested with the ideal data, and found to properly identify the letters, but with the non-ideal data, the network could not identify any of the characters.

The network was probably *over-learning* the prototype data set, so the number of neurons in each layer was changed a couple times. Even with mean squared errors (MSE) under .01, the network could not properly identify the non-ideal data.

5-Layer Network With Backpropagation Algorithm

Of the few 5-layer networks tested, the one with the best results had 2 neurons in the first layer and 5 neurons in the 2nd, 3rd, and 4th layers, and 1 neuron in the 5th layer. The tangent sigmoid function was used on the first 4 layers, and a pure linear function was used on the 5th layer.

Upon training, the network reached an MSE of virtually zero. When tested with non-ideal data, the performance was much better than with the 4-layer network, but still not as good as with the Linear Associator.

Results and Analysis

Using an ideal prototype data set, the best results for the 3 types of networks used are as follows:

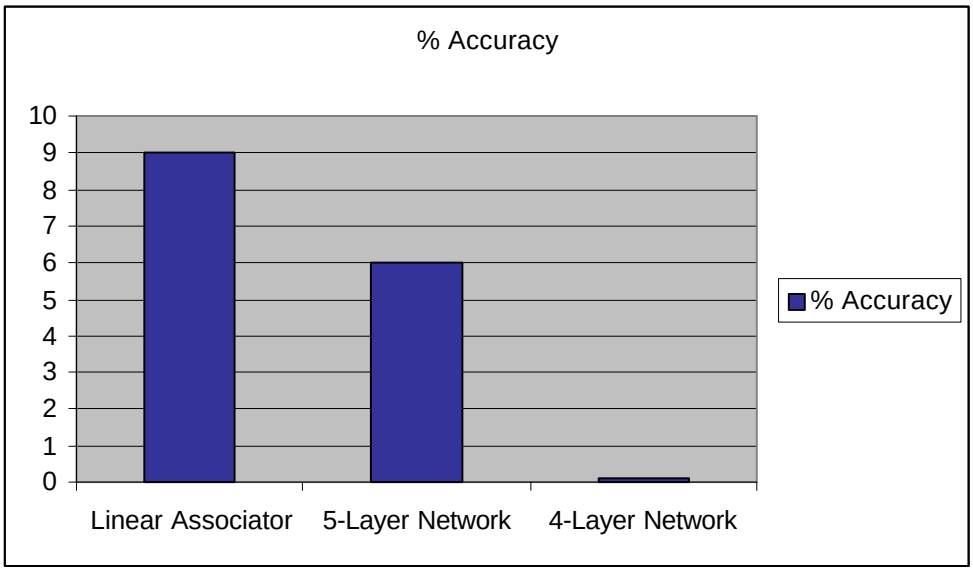


Figure 7. % Accuracy Using Ideal Prototypes

Note: These percentages do not include the spaces in the sentences that each network easily recognized. If taken into account, these percentages would be much higher.

Since the accuracy is obviously too poor, various measures were taken to try to improve performance. These included:

- Using edge detection on non-ideal data
- Using different schemes for the targets
- Sorting the prototype letters by similar shape and size

None of these attempts noticeably affected the performance.

The main reason the performance was so low was because of a character-offset effect that occurred when Matlab reduced the scanned image to black and white. See figure 8. The middle image is the ideal prototype, centered about its 9-pixel width, and the outer two images are what the scanned character may look like. Even though all the characters are identical, the offset makes it hardly possible for the neural network to identify it correctly.

Figure 8. Offset effect

Next, attempts to edit the prototype patterns were made because the prototype patterns should match (as well as possible) the non-ideal data that will be gathered.

For testing the effects, the Linear Associator was used because it had already been yielding better results than the other networks tested.

The first edit to the prototype patterns involved adding noise in places the scanned images looked noisy. For the scanned images in this project, most of the noise appeared at the top of the letters, so that is where the noise was added to the prototype patterns. This method increased the accuracy of the Linear Associator to 12%.

Then, non-ideal prototype patterns were created using the same method the non-ideal data was gathered. This greatly improved the performance. The Linear Associator gave an optimum accuracy of 21%.

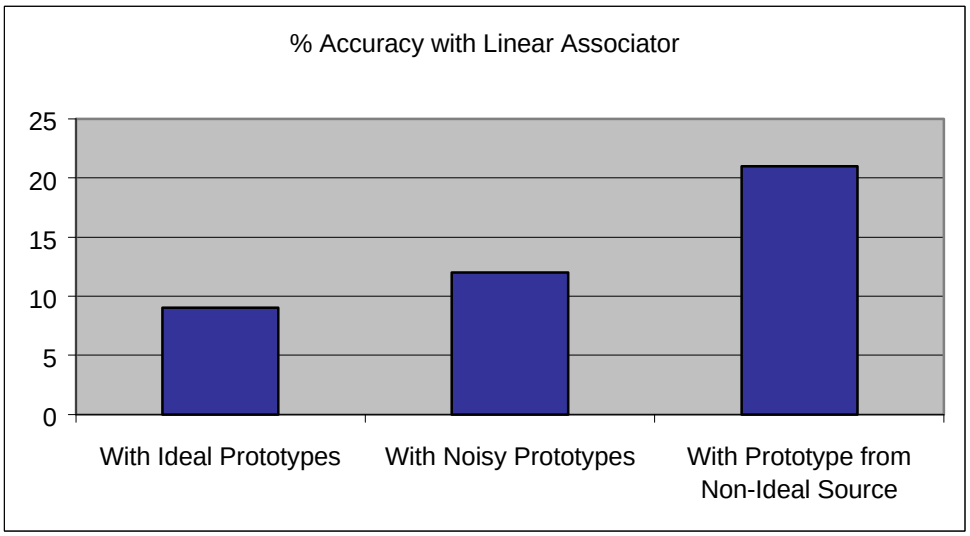


Figure 9. % Accuracy Using Different Prototypes

Conclusion

In this project, various networks were trained to recognize characters of the alphabet from a scanned image. The Linear Associator preformed the best *and* was also the simplest to implement.

After trying various methods to improve the performance, a character recognition accuracy of 21% was achieved when the prototype data was generated from the same source the test data was coming from. An accuracy of 21% means that out of every 100 letters, 21 will be correctly identified.

This accuracy is still very low, so other methods need to be approached for this type of character recognition, such as doing a more complicated edge detection algorithm, or using characteristic area ratios (for example, of black pixels to white pixels) of the characters to identify them.

Appendix – Matlab Code Explanation

An explanation of provided matlab files:

project.m - GUI for the character recognition project

getsamples.m – Gets prototype data in bitmap format

hebb.m – Simulation of Hebbian learning using Linear Associator

readline.m – Reads and simulates network on a line of image data (bitmap format)

projectresult.txt – File where resulting line of text is stored