

Mathematical Handwriting Recognition with a Neural Network and Calculation

Author: Tyler Sondag

Date: 4/22/07

For Dr. Pokorny's CSI 490 Course

Abstract

The goal of this project was to create a software system that recognizes handwritten mathematical expressions and computes the answer. No special syntax or formatting was to be required for these expressions, since a major goal of this system was for users to be able to use the system without having to learn anything new. Support was desired for algebraic expressions, integrals, and summations. The Java programming language was chosen for this project because of its ability to be used on a number of different operating systems and architectures without re-compiling the source code. A graphical front end was also desired in order for the system to be more user friendly.

Table of Contents

1. Introduction.....	3
2. Images.....	4
3. Neural Network.....	7
4. Scanner.....	9
5. Parser.....	10
6. GUI.....	11
7. Future Work.....	11
8. Conclusion.....	12
References.....	13

1. Introduction

Handwriting recognition is done in two different ways. The first is on-line recognition which examines the characters as the user is drawing them. This method is the simpler of the two, since the system only deals with one character at a time. An example of this method is character recognition on a personal digital assistant (PDA). The second type is off-line recognition. In off-line recognition the system must look at an entire group of characters instead of just one at a time. An example of this is optical character recognition (OCR) software for scanners. This system will use off-line character recognition. Once the system has broken a picture into its individual characters, a neural network will be used to determine each individual character. Next these characters, as well as information regarding their locations, are sent to the scanner. The scanner then rebuilds the individual characters into numbers and also determines which symbol goes to the parser next. In some cases, the scanner must also insert additional characters. The parser then requests one character at a time from the scanner and calculates the expression. Finally, a pop-up is displayed with the calculated answer.

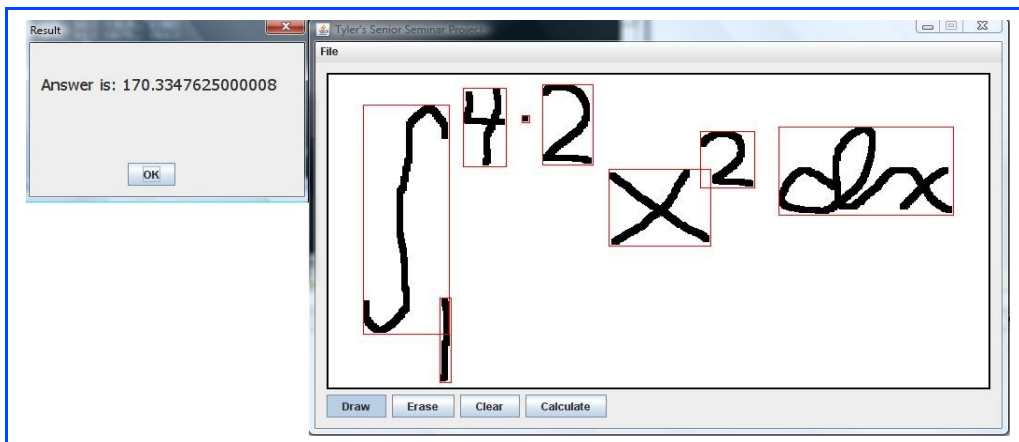


Figure 1: Example

2. Images

In this system, images are able to be input in two different ways. In either case, images are required to be gray scale. Support may eventually be added for non-gray scale images, but this was not considered important for the initial version of the system. The first method of picture input is with a bitmap file. The functionality for loading bitmap files was included for several reasons. First, since bitmap files do not compress the picture data no external libraries were required. Thus, converting the file into a data structure used by this system was much simpler. Second, for testing the system, it is much easier to send it a list of bitmap images to calculate rather than using the graphical user interface (GUI) of the system to draw test equations repeatedly. Finally, a future goal of the system is to allow users to load pictures in from a scanner, so being able to handle image files will allow this to work much more easily. The system currently does not support loading pictures from a scanner because scanned images typically have a lot of noise; in tests performed, this noise caused problems when breaking up the image into individual characters.

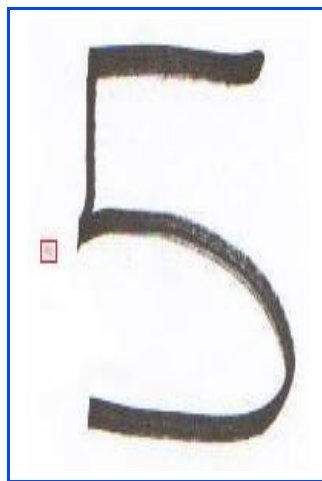


Figure 2: Noise in a scanned image

This functionality will be implemented at a later time. However, the system will have to filter these images and clean up the noise (most likely by using a Gaussian filter), and this was simply not feasible

given the limited time constraints. It is also a future plan to include support for other file formats of images (JPEG, GIF, PNG, etc.). The second method involved “drawing” the pictures on the screen through the program's GUI. This method is used in the current implementation, since it was considered the easiest and fastest for a user. Images are drawn by clicking and dragging the cursor around the draw panel of the GUI. Erasing is also allowed using the same method. The user may also clear the entire panel. When the user is finished writing an expression he or she simply clicks on the “Calculate” button. The system then draws an outline around each character it finds and displays a pop-up containing the calculated answer.

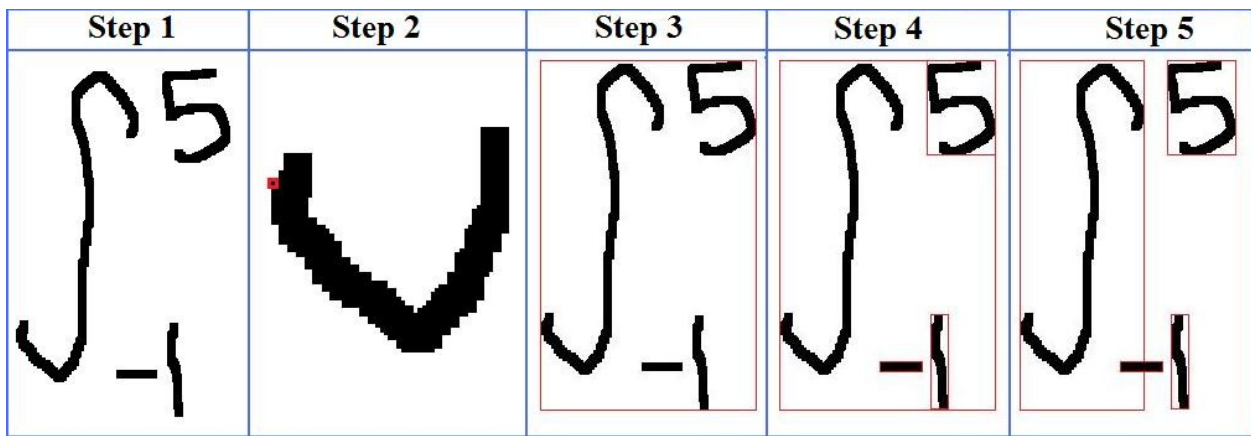


Figure 3: Character break up

For the following explanation, refer to Figure 3 for a graphical example of each step of the process. Once an image has been loaded in the system (Step 1), it must be broken up into individual characters. Currently, the system checks pixels from left to right until it finds a pixel value below some threshold (a black pixel has a value of 0, and a white pixel has a value of 255). The system then creates a small bounding box around this pixel (Step 2). Each of the four sides of this bounding box is checked to see if it crosses any pixel below this threshold value. If it does, the box is extended in that direction. This process is repeated for each side of the box until the edges of the bounding box cross no pixels below the threshold (Step 3). This method works in only some cases, since it is common that this

bounded box will contain multiple characters. Some examples of this situation include characters underneath a square root and bounds of an integral. To remove these extra characters, the bounded group of characters is scanned in the same fashion from different directions. After a character is removed from the bounded group of characters, the group is scanned again until no more characters are removed (Step 4). Finally, the bounding box of the original character is recreated since removal of characters may have affected its size (Step 5). This method has many flaws. It is very successful in breaking up characters that are not connected, but it is unable to break up characters that are connected (for example cursive writing). Fortunately in mathematical expressions connected characters are uncommon, especially when writing on a computer screen. Hence, for the current project this method was considered acceptable.

Once the image is broken up into its individual characters, each character's location information is stored along with the pixel values inside its bounding box. These pixel values are converted into a 10 pixel by 10 pixel representation of the character, since the neural network must be given a fixed number of input pixels for all characters. One problem that arose with this method was that some characters, when converted to a 10 pixel by 10 pixel representation all look the same. For example, a very straight 1 or minus (-) will turn into a block of all dark pixels, and the system will be unable to distinguish these from a multiplication sign or decimal point (\cdot). Also, a slightly slanted 1 will look a lot like a division sign ($/$). To deal with this problem, images that are very tall and narrow are padded on the sides with white pixels, and images that are very wide and short are padded on the top and bottom with white pixels.

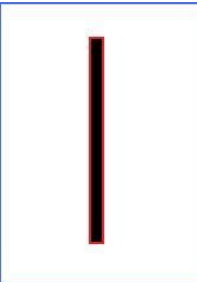
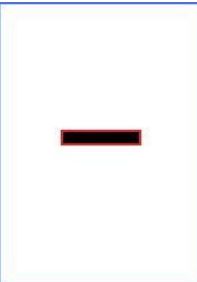
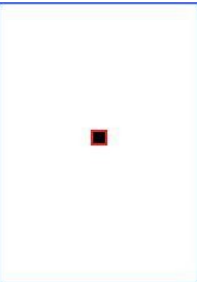
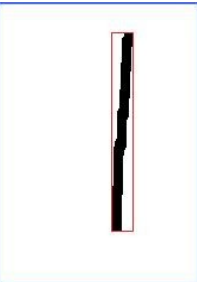

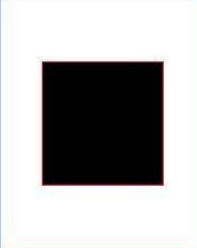
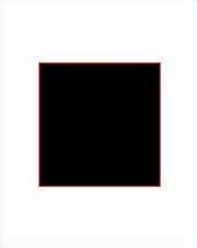
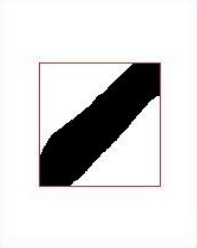
Bounded characters				
10x10 representation				

Figure 4: 10x10 problems

3. Neural Network

The neural network used for the recognition of individual characters is a feed-forward neural network with four layers. The first layer contains 100 inputs, that is, one for each input pixel. The output layer contains an output for each character that is to be recognizable by the system. Values for each input pixel are sent into a corresponding node in the first (input) layer. For each node in the first layer, its input value is sent to an activation function, in this case the logistic sigmoid function¹. The output of this function is sent to each node in the next layer. However, the output it is not sent directly; each output is multiplied by some weight before going to the nodes in the next layer. Each node in the next layer sums all of the signals it receives and sends this value to its activation function. This process repeats until the final output vector to the network is found.

¹ $\varphi(x) = \frac{1}{1 + e^{-x}}$

For example, for the neural network in Figure 5, to calculate the output of node $n+2$, each output for the previous layer (nodes 2 through $n+1$) must be calculated and multiplied by the corresponding connection weight. This calculation can be represented by the following equation:

$$o_{n+2} = \varphi_{n+2} \left(\sum_{k=2}^{n+1} w_{n+2,k} o_k \right)$$

o_k is the output of node k , φ_k is the activation function of node k , and $w_{k,j}$ is the weight going from node j to node k .

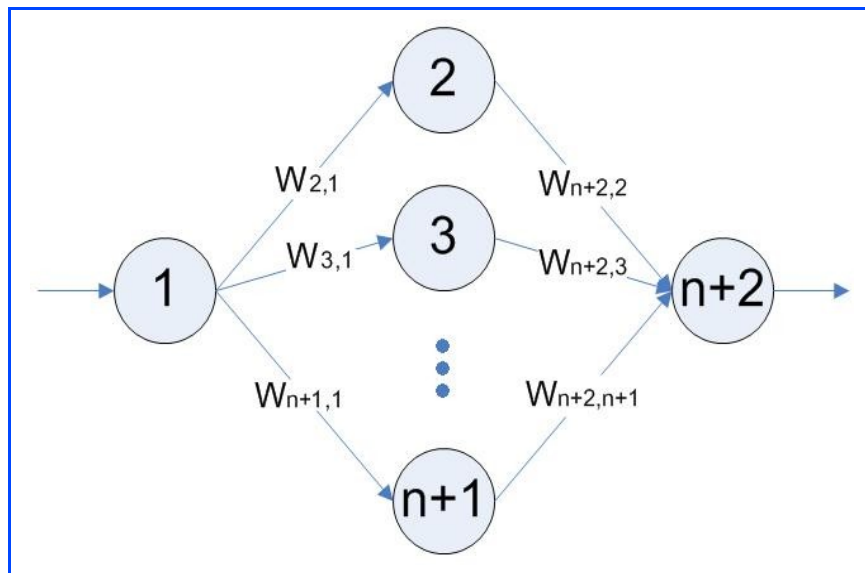


Figure 5: Sample artificial neural network

To train the neural network to recognize an individual's handwriting, a training set is created that contains a 10 images of each character the system is to recognize. The system can perform quite well when trained with fewer than 10 examples of each character; however, 10 was chosen to ensure a high level of accuracy. The user has the ability to train with more or less than 10 of each, but 10 is the default and the recommended amount of each. Each item in this training set is paired with a desired output vector. This is essentially a 0 vector except for the n^{th} element, which corresponds with this

character, contains a 1. Next, the gradient descent learning method is used to train the neural network. Training is done by adjusting the weights of the network until the total error for the training set is below $.005 a^2 b$, where a is the number of characters being recognized by the system (number of outputs for the neural network) and b is the number of each character in the training set. This can also be thought of as the total number of outputs when all inputs from the training set are sent into the neural network multiplied by $.005$. The total error is calculated by sending each member of the training set through the network and calculating the sum of the absolute values of the differences of the individual components of the output vector and desired output vector. Weight adjustments are calculated with the following equation:

$$w_{j,i} = w_{j,i} - \alpha \delta_j o_i^k$$

α is the learning rate (in this program this is simply 1), and δ_j is calculated as follows:

$$\delta_j = e_j^k \varphi'(net_j^k) \text{ if } j \text{ is in the output layer}$$

$$\delta_j = \varphi'(net_j^k) \sum_m w_{m,j} \delta_m \text{ if } j \text{ is in a hidden layer}$$

net_j^k is the sum of the inputs to node j for the k^{th} element of the training set.

Once the training is complete the weights are stored to a file which can then be loaded in by the user rather than having to retrain the system each time it is used.

4. Scanner

The scanner for this project works quite differently than a scanner for a programming language compiler. Normally, the next character in the sequence is the next character in the file; however, in this implementation the next character is not necessarily known. When a user draws an equation, the system breaks the image into individual characters and has the neural network recognize these individual characters. Once each individual character is recognized and its location information is

stored, this information is sent to the scanner. The scanner turns this information into tokens which are then sent to the parser. Numbers (0-9) and decimals (.) must be put together to form the number they constitute. For example, if the user writes the number 10.4, the system will see each character separately and must determine that these four individual characters make up the real number 10.4. Also, when two adjacent terms are multiplied, for example $3x$, the scanner must put a multiplication symbol between the terms. Similarly when the system encounters a power, it must insert the $^$ symbol, so that the parser knows it has reached a power. When series or integrals are found, the system must look for bounds instead of simply grabbing the next character. The system must also determine which character or group of characters is the next to be sent to the parser.

5. Parser

The parser takes the tokens given by the scanner and calculates the result. The parser must ensure that operations are calculated in the correct order and that operations that require numerical methods are calculated accurately. To calculate integrals, the trapezoidal method is used. The number of trapezoids used increases until the estimated error is below .00001 or until 100 trapezoids are used. Since calculation of multiple integrals requires intense computation, the stopping criteria are relaxed in order to give a timely result.

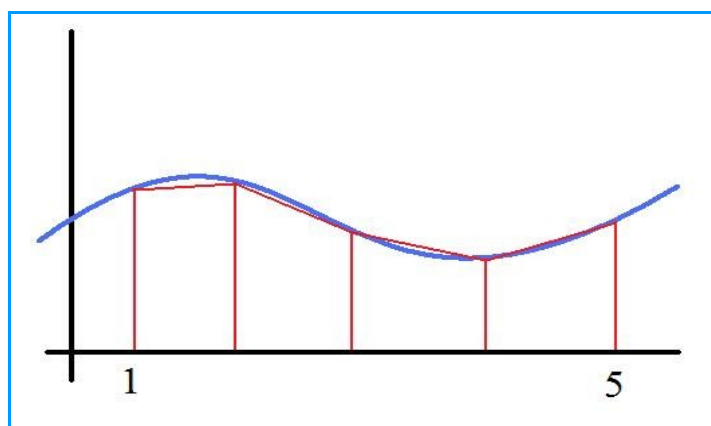


Figure 6: Example of trapezoidal method calculating an integral from 1 to 5 with four trapezoids

6. GUI

The graphical user interface has a canvas for the user to draw on and allows the user to draw, erase, and clear the entire canvas. The system has an option for creating a new user, which allows the user to choose which groups of characters he or she wants the system to be able to recognize. The system then asks the user to draw 10 examples of each character he or she asked the system to be able to recognize. Once all of these examples have been drawn, the training set is created and used to train the new user's neural network. Once the new user's neural network has been trained, the neural network weights are saved so that the user can load his or her profile at any time.

7. Future Work

In addition to the future goals mentioned previously in the paper, one future goal for the system is to include more error handling. Currently no message is displayed when an expression is written or recognized incorrectly. Eventually the system will explain why it could not calculate the expression and will suggest changes to the user. Similarly, the answer pop-up will eventually include a TeX

representation of the user's input expression. This representation will allow the user to ensure that the system recognized each character correctly. Another feature that will be implemented is the ability to add additional character sets to be recognizable by the user's neural network. That way, if the user suddenly decides he or she would like recognition for integrals but he or she has not trained the system for this, rather than having to go through the entire training process again, the user can simply draw the new characters. The system will retrain itself using this new data and the training set the user previously created. This retraining will require restructuring of the save file, since the original training set will need to be saved. Similarly, a guest user will eventually be added that will be trained with all of the other users' training sets. This will allow a user to use the system without training; however, it will not recognize his or her writing as well as it would if he or she were to go through the training process. Support is also planned for calculating derivatives, basic matrix operations, product series, and roots of equations.

8. Conclusion

For this project I created a software system that allows users to handwrite mathematical expressions which are then recognized and calculated by the system. This system is not yet complete. A working version has been created, but there are many features that have not been implemented yet. Additionally, as with most pieces of software there are many bugs that need to be found and corrected. I plan to continue to work on this project until all desired features are implemented.

References

1. Stevenson, Charles F., 1966. Neurophysiology: A Primer, John Wiley & Sons, Inc.
2. Gerald, Curtis F. and Wheatley, Patrick O., 1999. Applied Numerical Analysis, 6th Ed., Addison-Wesley
3. Russel, Stuart J. and Norvig, Peter, 2003. Artificial Intelligence: A Modern Approach, 2nd Ed., Prentice Hall
4. Schalko, Robert J., 1997. Artificial Neural Networks, McGraw-Hill
5. Li, Hongzong, Chen, Philip C.L. and Huang, Han-Pang, 2001. Fuzzy Neural Intelligent Systems, CRC Press LLC
6. Jang, J.-S. R., Sun, C.-T. and Mizutani, E., 1997. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence, Prentice Hall
7. Kosko, Bart, 1992. Neural Networks and Fuzzy Systems: A Dynamic Systems Approach to Machine Intelligence, Prentice Hall
8. Mammone, Richard J. and Zeevi, Yehoshua, 1991. Neural Networks: Theory and Applications, Academic Press, Inc.
9. Principe, José C., Euliano, Neil R., and Lefebvre, W. Curt, 2000. Neural and Adaptive Systems: Fundamentals Through Simulations, John Wiley & Sons, Inc.
10. Foley, James D., van Dam, Andries, Feiner, Steven K., Hughes, John F., 1996. Computer Graphics: Principles and Practice: Second Edition in C, Addison Wesley