# HANDWRITTEN EQUATION INTELLIGENT CHARACTER RECOGNITION WITH NEURAL NETWORKS

STEVEN SLOSS

Abstract. We seek to implement intelligent character recognition of handwritten equations in a manner that allows for a computer program to translate a user's hand-drawn equation into formatted LaTeX. We use a Kohonen neural network fed by downsampled images to accomplish this task. While we were able to achieve single-character recognition with a very low error rate, multiple character recognition and whole equation recognition have proven elusive.

## Contents

1. Introduction 1
2. Neural Networks Background 2
2.1. Biological to Artificial Neural Networks 2
2.2. The Neuron 3
2.3. Neuron Connection Weights 4
2.4. Neuron Firing Rules 4
2.5. Training Neural Networks 4
2.6. Validating Neural Networks 4
3. Kohonen Neural Networks 5
3.1. Training Kohonen Networks 6
3.2. Assigning Weights 6
4. Character Recognition with Kohonen Neural Networks 7
4.1. Single Character Recognition 8
5. Future Work 8
5.1. Multi-Character Recognition 9
5.2. Equation Recognition 9
6. Conclusion and Outlook 9
7. Acknowledgements 10
References 10

## 1. Introduction

While computers can generally perform many tasks far faster than humans, humans are still vastly superior at other tasks, like categorizing, finding patterns, and image recognition. (As a thought experiment, imagine teaching a two-year-old how to identify what traffic lights mean, and then imagine teaching a computer to do the same task.) Under the assumption that part of this ability is due to the

*Date*: 30 April 2007.

structure of the human brains, artificial intelligence researchers created artificial neural networks .

Biological neurons , as found in a brain, are interconnected through very complex networks. They function by accepting a signal as an input and only transmitting the signal to other neurons if the signal is sufficiently strong. Artificial neural networks use an extremely similar structure (though often on a smaller scale), and we can leverage this structure to "learn" pattern recognition in much the same way as an animal brain. That is, artificial neural networks, like people, learn by example. They are configured to a specific application through an involved learning process which, as in their biological analogues, involves adjustments to the synaptic connections between neurons.

In the case of this paper, the specific application is an attempt to use neural networks to perform intelligent character recognition on human handwritten equations, automating the transformation of handwritten equations to LaTeX.

## 2. Neural Networks Background

We will begin our discussion of handwriting recognition with a brief introduction of the background and mechanics of artificial neural networks .

Neural networks, while apparently a fairly recent development, were actually conceptualized before the advent of computers as we now know them. In fact, McCulloch and Pitts, in 1943, developed models of neural networks based on the understanding of neuroscience of the day. They thus made several simplifying assumptions, but nevertheless produced neural networks that were capable of making logical decisions (like $A \wedge B$ or $A \vee B$). It wasn't until 1974, however, that Werbos developed the currently popular notion of a backpropogating neural network and thus popularized and revolutionized the field.

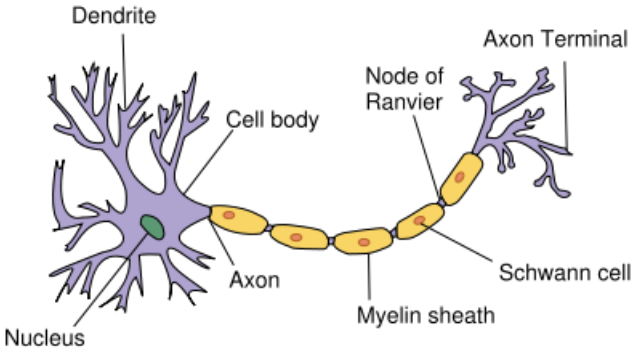


Figure 1. A single neuron in an biological neural network.

2.1. **Biological to Artificial Neural Networks.** There are numerous parallels between artificial and biological neural networks . Much is still unknown about how human brains train themselves to be able to process information, but current theories state that both artificial and biological networks use neurons that are parts of vast interconnected networks. A typical biological neuron accepts signals from other neurons over a cluster of very fine structures known as dentrites . After accepting the signal, that neuron may fire if its excitatory input is sufficiently large

compared to the background electrical activity. Upon firing, a signal is transmitted over the axon terminals, over a structure known as a synapse , which converts the excitation of the axon into electrical activity in neighboring neurons. Learning takes place when the effectiveness of synapses in the brain changes, thereby changing the influence of one neuron on another.

Computers, being digitial rather than analogue, function in an analagous though distinct manner. We may then define what it means to be an artificial neuron.

Simply, a neuron is a straightforward device with multiple inputs and outputs (see Figure 2). We define two modes of operation for the neuron - training mode and operational mode. In training, the neuron is taught to fire or not fire for a particular pattern of inputs. In the operational mode, the neuron is activated when the current input matches the input for which it was trained. If the current input is not exactly the input for which it was taught, a firing rule (discussed later) is used to determine whether to fire.

Artificial neural networks (hereafter neural networks) have the advantages of being able to learn adaptively (i.e. they can learn how to perform tasks based on some initial experience), being self-organizing, and being exceptionally good at finding solutions when there is no clear set of steps to find the answer.

Neural networks are, however, not particularly suited to solving deterministic problems, problems whose solutions can be easily diagrammed on a flowchart, where the logic of the program is likely to change, or where it is necessary to know exactly how the solution was derived. They are, however, extremely adept at solving problems that cannot be solved as a finite series of steps, or that deal with pattern recognition or classification.

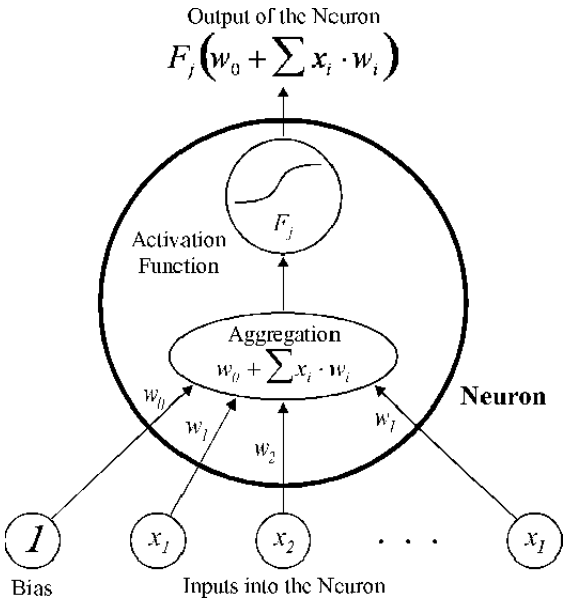


FIGURE 2. A single neuron in an artificial neural network. From [1].

2.2. **The Neuron.** An artificial neuron is a very simple computational device with many inputs and outputs. We can represent a single neuron as in Figure 2. It

recieves input either from other neurons or the program's input. When the sum of the inputs reaches some level, it is said to "fire," that is, it sends signals to other neurons.

2.3. **Neuron Connection Weights.** As we mentioned before, neurons are connected together by "synapses." Note, however, that not all of these connections are created equal – each connection is assigned a weight. It is these weights that allow the neural network to recognize certain patterns. Adjusting the weights of a neural network will result in it recognizing a different pattern. Thus, when we train a neural network, we are merely adjusting the weights between neurons.

2.4. **Neuron Firing Rules.** Firing rules, the set of rules that determine whether a particular neuron fires, are an extremely important component of a neural network, and much research in the field has focused on finding firing rules that apply to generalized neural network problems. (Note that firing rules may also be known as activation functions).

Consider a single neuron, connected to a number of other neurons. Then when the sum of the inputs

$$i = \sum_k w_k x_k$$

where $w$ represents the weights between this neuron and the other $k$ neurons and $x$ is the input to this neuron from other neurons, exceeds some threshold value, the neuron "fires." There are several popular activation functions, like Sigmoid, but we use hyperbolic tangent in our application, as we experimentally found this to have the lowest error rate.

2.5. **Training Neural Networks.** Recall that individual neurons in the neural network are connected through the synapses, which allows neurons to signal each other as an input is passed back and forth. Each of these connections has a weight, and that "training" a neural network is merely the process of adjusting these weights so that the proper neurons fire given a specific input to produce the correct output. In fact, this segment of neural network design is analogous to an enormous optimization problem and thus draws on lessons learned in other mathematics and computer science optimization research.

There are two main paradigms in neural network training – supervised learning and unsupervised learning. In supervised learning, we give the neural network training data in pairs $(x, y)$, where $x$ is an input and $y$ is the corresponding correct solution. Our aim is then to minimize the error (usually given as the mean squared error) of the neural network.

Unsupervised learning takes place when we have no corresponding correct solution for each input that the neural network is trained on, and we wish for the neural network to classify its inputs into a specific number of groups. Our application uses unsupervised learning.

2.6. **Validating Neural Networks.** It is extremely easy for a neural network's training to have found a corner case. That is, our training may have caused the neural network to return the correct output solely for the input we trained it on, which is obviously of very limited utility. Therefore, it it necessary to *validate* a neural network, to check that it is fit for use in a more general setting. This step

is generally also used to determine whether we may cease training, or if the neural network requires further epochs of training.

In general, the process for this requires separation of the training data and validation data. Generally, half the training data trains the network to recognize a valid input, and we use the other half to verify that the neural network produces generally correct solutions (i.e. that it is valid).

## 3. Kohonen Neural Networks

Pattern recognition with neural networks is normally done with a *feed-forward* neural network , which were the first types of neural networks devised, and remain among the simplest and most useful. The name refers to the requirement that information move in only one direction through the neural network – forward from the input nodes, through the hidden nodes (if they exist in the particular network), and to the output nodes. A feedforward network can thus be equivalently defined by requiring that there be no cycles in the network.

We, however, use a Kohonen neural network, named after Tuevo Kohonen, which contains no hidden layers and handles output slightly differently than the generalized neural networks we described above. It also does not use an firing rule, and does not include bias weights along neuron pathways.

The input to our neural network consists of floating point numbers, normalized to a range of $[-1, 1]$. Whereas the output of a normal neural network consists of output from all of the output neurons, in our version of a Kohonen network, a "winner" neuron is chosen from the set of output neurons that represents the output of the network. For example, if we were to train the program to recognize all capital letters from the English alphabet, we would have a Kohonen neural network with 26 output neurons (which map to the letters of the alphabet), from which a winner is chosen when we recognize a character.

We chose a Kohonen network because it is a particularly simple network whose training and recognition progress very rapidly, which is a benefit in a real-time application like the one we are attempting to solve.

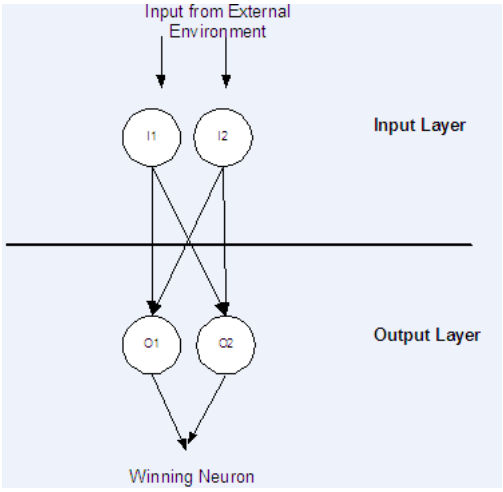


Figure 3. A simple example of a Kohonen Neural Network. From [3].

For clarity, we will proceed with an example of a very simple Kohonen neural network. Consider the network in Figure 3. Let the inputs be defined as in Table 1 and the connection weights between neurons be defined as in 2 (recall that these values would have been set during training).

TABLE 1. Example Kohonen Neural Network Inputs

| | |
|---|---|
| Input Neuron 1 ($I_1$) | 0.5 |
| Input Neuron 2 ($I_2$) | 0.75 |

Recall that the input needs to be normalized between $[-1, 1]$. We calculate the "vector length" of the input vector by summing the squares of the input:

$$L = (0.5 \cdot 0.5) + (0.75 \cdot 0.75) = 0.8125$$

which then gives us a normalization factor of $\frac{1}{\sqrt{0.8125}} = 1.1094$.

To calculate the output of each neuron, we first take the dot product of the input neuron vector and their connection weights, multiplied by the normalization factor

$$\begin{pmatrix} 0.5 & 0.75 \end{pmatrix} \cdot \begin{pmatrix} 0.1 & 0.2 \end{pmatrix} \cdot \frac{1}{\sqrt{0.8125}} = 0.395 \cdot 1.1094 = 0.438213$$

TABLE 2. Example Kohonen Neural Network Connection Weights

| | |
|---|---|
| $I_1 \rightarrow O_1$ | 0.1 |
| $I_2 \rightarrow O_1$ | 0.2 |
| $I_1 \rightarrow O_2$ | 0.3 |
| $I_2 \rightarrow O_2$ | 0.4 |

(note that the calculation for the second output neuron follows in precisely the same way, and will have a value of 0.0465948). In this case, we choose a winner by choosing the output which has the highest value, which is clearly the first neuron.

3.1. **Training Kohonen Networks.** Like most neural networks, training a Kohonen neural networkinvolves iterating through several epochs until the the overall error is below some defined threshold. Kohonen neural networks use unsupervised training.

The difference lies in the way weights are adjusted – in training a Kohonen neural network, one neuron will "win," resulting in its weight being adjusted to react even more strongly to that particular input. Thus, as different neurons are chosen as the winner for different patterns, their ability to recognize their specific patterns will be increased. We can summarize this process as an activity diagram in Figure 4.

3.2. **Assigning Weights.** There are two traditional methods for adjusting the weights within a Kohonen neural network– the additive method and the subtractive method. Kohonen proposed the additive method, which is defined by

$$w^{i+1} = \frac{w_i + \alpha x}{|w^i + \alpha x|}$$

where $w^i$ is the weight of the winning neuron in a particular training cycle, $w^{i+1}$ is the new weight, $x$ is the training vector presented to the network, and $\alpha$ is some weight.

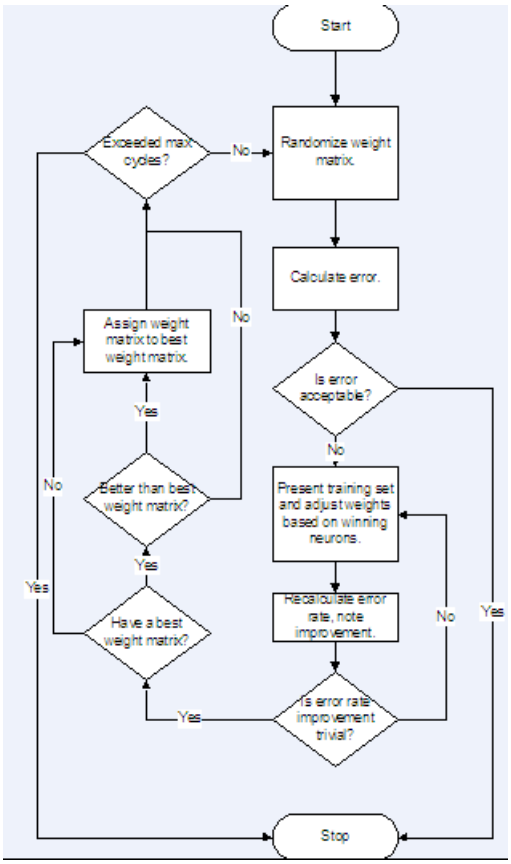


FIGURE 4. The training process for a Kohonen neural network. From [3].

Alternatively, if the additive method fails to converge, we may use the subtractive method,

$$w^{i+1} = w^i + \alpha \left( x - w^i \right).$$

For the purposes of this project, the additive method suffices.

## 4. Character Recognition with Kohonen Neural Networks

With that background in neural networks and Kohonen neural networks, we may begin to implement a neural network application to perform intelligent character recognition, that is, transforming human handwriting into a format that a computer can recognize. The framework consists of approximately 3,000 lines of Java that implement a drawing area, a Kohonen neural network, a generic neural network superclass, functionality to downsample user-drawn images, and support for loading and saving of training sets for neural networks.

We may generally break up the problem of transforming handwritten equations into LaTeX into three parts: single character recognition with a neural network, multiple character recognition (e.g. sentence recognition), and positional recognition (to recognize such things as exponents).

4.1. **Single Character Recognition.** We first approach the most complex of the three sub-problems, single character recognition. Given a user-drawn image representing a character, we must convert this image into a format a neural network can understand, feed this data into a neural network constructed and trained such that it can recognize characters, and then transform the output of the neural network into something useful.

We have used Java to create an application and an extensible framework to be used for character recognition. We construct an extension of the Java SWING JPanel to support user drawing, and then store the user-drawn character as a Java Image object. We then construct a bounding box around the character to do away with excess white space, and then we downsample the image to a double array of boolean values (here "true" represents a pixel that has been drawn in) to feed to the neural network. Experimentally, we found that a 5×7 grid has the lowest error rate. Actual implementation issues, which comprised the vast majority of time spent on the project, are beyond the scope of this discussion.

This boolean double-array is passed to a Kohonen neural network, which was trained on similar boolean arrays to recognize characters of the alphabet. As described in the previous section, the winning output neuron maps to the character that the neural network believes most closely resembles what the user drew.

This backend was attached to a Java SWING graphical user interface for the purposes of demonstrating the capabilities of the neural network at this stage in the project. It is shown in Figure 4.1.

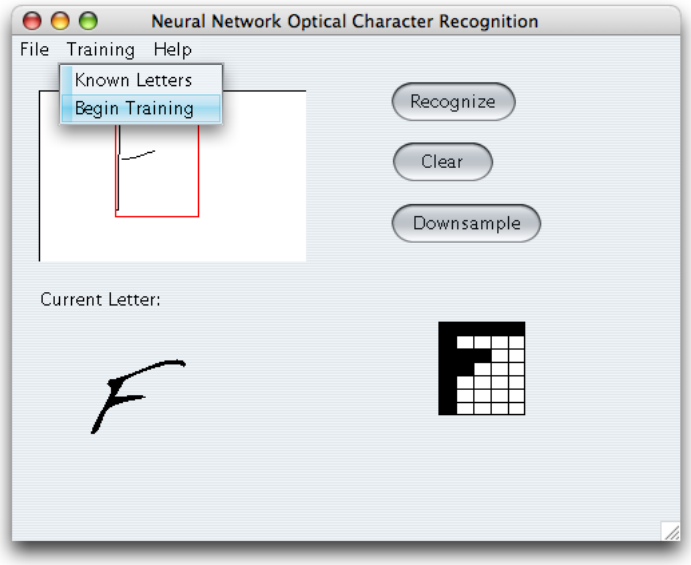


FIGURE 5. The graphical user interface to the single-character recognition component of the project.

## 5. Future Work

It should be a tractable task to extend the knowledge gleaned from this research and transform it into an equation recognition system. Though the implementation

has not been completed due to difficulties with Java, we may outline the approach for the remaining two steps that must be completed in order to create a handwriting to LaTeX system.

5.1. **Multi-Character Recognition.** Multiple character (e.g. sentence) recognition will not require the use of a neural network, and is merely a problem of image processing. In general, instead of finding a bounding box for a single character, we find a bounding box around each of the multiple characters that a user draws. Each of these characters is then fed into the neural network character recognition framework separately. In finding bounding boxes for each of the drawn characters, we make the assumption that no character is connected to another. We, however, believe this is a valid assumption for the scope of this project, as through our user testing, we never observed a person who writes mathematics in script.

We visualize the multiple bounding boxes approach in Figure 6.

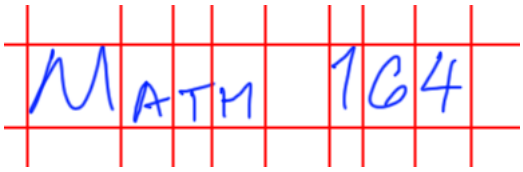


FIGURE 6. A rendering of the process of finding a bounding box for each drawn character. Each of these characters would then be recognized by a Kohonen neural network.

5.2. **Equation Recognition.** Recognizing entire equations will be a much more complex process than recognizing simple English phrases. There are two primary problems that need to be solved when performing whole-equation recognition: tokenization of the equation and positional character recognition.

With positional character recognition, when we find bounding boxes for each character the user writes, we also record the size and position of each character. Thus, for $2^4$, along with the characters 2 and 4, we would record that the 4 is slightly above and smaller than its neighboring characters. This position and size data would be fed into a second neural network, which would be trained to map character positions and sizes into exponents, subscripts, fractions, and the like. In trying to develop such a system, fractions have proved to be a problem with this method, and we know of no solutions.

Also, since math as it is hand-written and math as it is represented in LaTeX does not exactly correspond, we must tokenize and then parse the results of the neural network runs such that we may translate them into LaTeX source. This is an extremely solvable problem, as written mathematics already has very well defined parse trees.

## 6. Conclusion and Outlook

From experiments with users, our single-character recognition system was able to achieve a 94.3% success rate with generalized handwriting training data and 96.2% when trained with a particular user's handwriting.

The goal of this project was to push neural networks into a territory that they had thus far not been used to explore. While we have, as of writing, not been

able to complete a translation from handwriting to LaTeX, we believe that given the approaches outlined to solve the multiple character recognition and positional recognition problems, allowing a neural network system to translate between It is our belief that, while this has been accomplished with other domain-specific intelligent character recognition techniques, this would be the first time anyone has accomplished handwritten equation recognition with neural networks.

## 7. Acknowledgements

I would first like to thank Professor Yong for his wonderful Scientific Computing class that gave me the confidence and skills to tackle this project.

I would also like to extend my gratitude to the residents of East Dorm, who provided me with invaluable assistance in training the neural network to recognize a variety of handwriting styles.

## References

[1] `http://www.xplore-stat.de/tutorials/xlghtmlimg565.gif`
[2] `http://commons.wikimedia.org/wiki/Image:Neuron.svg`
[3] Heaton, Jeff T. *Introduction to Neural Networks with Java*. New York: Heaton Research, Inc. 2005.
[4] Zomaya, Albert Y., ed. *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*. New York: Springer Science+Business Media, 2006.