FREEFORM CURSIVE HANDWRITING RECOGNITION USING A CLUSTERED NEURAL NETWORK

Kelly H. Bristow

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2015

APPROVED:

Kathleen Swigger, Major Professor
Song Fu, Committee Member
Xiaohui Yuan, Committee Member
Barrett Bryant, Chair and Professor of
Computer Science and Engineering
Costas Tsatsoulis, Interim Dean of the
Toulouse Graduate School

Bristow, Kelly H. *Freeform Cursive Handwriting Recognition Using a Clustered Neural Network*. Master of Science (Computer Science), August 2015, 44 pp., 3 tables, 8 figures, 13 numbered references.

Optical character recognition (OCR) software has advanced greatly in recent years. Machine-printed text can be scanned and converted to searchable text with word accuracy rates around 98%. Reasonably neat hand-printed text can be recognized with about 85% word accuracy. However, cursive handwriting still remains a challenge, with state-of-the-art performance still around 75%. Algorithms based on hidden Markov models have been only moderately successful, while recurrent neural networks have delivered the best results to date.

This thesis explored the feasibility of using a special type of feedforward neural network to convert freeform cursive handwriting to searchable text. The hidden nodes in this network were grouped into clusters, with each cluster being trained to recognize a unique character bigram. The network was trained on writing samples that were pre-segmented and annotated. Post-processing was facilitated in part by using the network to identify overlapping bigrams that were then linked together to form words and sentences. With dictionary assisted post-processing, the network achieved word accuracy of 66.5% on a small, proprietary corpus.

The contributions in this thesis are threefold: 1) the novel clustered architecture of the feed-forward neural network, 2) the development of an expanded set of observers combining image masks, modifiers, and feature characterizations, and 3) the use of overlapping bigrams as the textual working unit to assist in context analysis and reconstruction.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Optical character recognition, or OCR, is the process of automatically converting text that has been printed on paper into a format, such as ASCII or Unicode, suitable for digital storage and manipulation. Historically, OCR has been applied to such tasks as electronic processing of business receipts, converting printed texts to speech for the visually impaired, digitizing historic documents for online reference, and sorting mail. The goal of OCR is to convert text quickly and accurately, with as little human intervention as possible.

1.1 Optical Character Recognition (OCR)

Optical character recognition, in the general sense, refers to any technology that automatically converts printed text to a digital format. In a more restrictive sense, OCR refers to the recognition, by means of template matching, of text printed in a specific font. One of the most successful OCR systems is based on the OCR-A font developed by American Type Founders and released in 1968 [1]. The OCR-A font was designed to be easily recognized by computers of that era, while being readable by humans.

1.2 Intelligent Character Recognition (ICR)

While template matching works well for situations involving a font such as OCR-A, a more flexible approach is needed for recognizing the wide variety of fonts found in the world at large. Recognizing handwritten text, with its significant variation even within a single document,

presents a greater challenge, far beyond the reach of traditional OCR methods. Cursive handwriting, with its continuously flowing script, is more difficult, still. Intelligent character recognition, or ICR, is the application of advanced machine learning techniques to these difficult OCR problems.

1.3 Defining Accuracy

Before the accuracy of a character recognition system can be measured, it must first be decided what type of accuracy is to be measured. The two primary performance measures are character accuracy and word accuracy. Character accuracy is defined as the percentage of characters that are correctly identified in a body of text, while word accuracy is the percentage of words that are correctly identified [2]. It is quite possible for these two measures to diverge greatly, for example, if the system correctly discerns most individual characters, but misidentifies a single character in most words of more than four or five characters. Since the objective is to convert a handwritten document into searchable text, the preferred performance measure is word accuracy. In this thesis, performance is measured by word accuracy. A word is considered to be correctly identified if it is spelled as written by the handwriter without regard to case.

1.4 Sayre's Paradox

The single greatest obstacle to cursive handwriting recognition was described in 1973 by Kenneth M. Sayre, and has come to be known as Sayre's paradox [3]. This research states that a cursive handwriting sample cannot be recognized without first being segmented, and it cannot

be segmented without first being recognized. The problem arises from the fact that there is typically no separation between individual characters in cursive handwriting. In other words, there is no apparent segmentation of the text. Instead, the letters that form a word flow continuously from one to the next, and it is often not clear where the segmentation points are that separate the characters. An example of this ambiguity is illustrated in Figure 1.1.

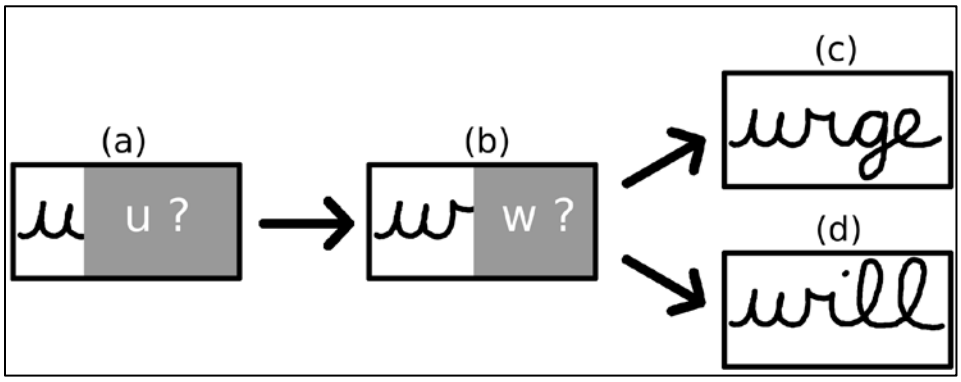


Figure 1.1 This diagram illustrates how Sayre's paradox complicates cursive handwriting recognition. In (a) a handwriting sample has been masked at a reasonable segmentation point. The exposed portion appears to be the letter "u." In (b) the mask has been moved to another reasonable segmentation point to the right of the first one. Now the exposed text appears to be a "w." Exposing the entire word shows how it could be either a "u" or a "w," in (c) and (d), respectively.

Sayre's paradox affects the ability of any ICR program that depends on identifying segmentation points to recognize cursive handwriting. Modest success has been achieved by using classifiers based on hidden Markov models, especially in on-line recognition systems that follow pen movement [4]. HMMs depend only on the current state and the previous state of the classifier, and have been widely used in voice recognition systems [4]. Speech recognition faces similar challenges with regards to segmentation, thus making HMMs a natural choice for

cursive handwriting recognition [a][5]. Character recognition systems based on recurrent neural networks have been more successful. A team led by Dr. Jürgen Schmidhuber [5] developed a unique type of recurrent network called a long short-term memory (LSTM) and were able to achieve a word accuracy of 74.1% on a corpus of 13,040 lines of handwritten text.

Central to this thesis is a three-part strategy for confronting Sayre's Paradox. The first part of this strategy comes into play during the training phase of the network. Each training sample is marked with segmentation points denoting start- and end-boundaries for the individual characters comprising the sample. The sample is then read into the network as bigrams of adjacent characters, with each successive bigram sharing a common character. The second part of the strategy is to simply ignore Sayre's Paradox while scanning text during the test phase. The text is scanned in a roughly continuous manner, with only rudimentary pre-segmentation to eliminate loops and vertical strokes as possible character boundaries. This pre-segmentation is done to save processing time and to reduce the number of erroneous candidate solutions available in post-processing. The third and final part of the strategy is to use bigrams identified by the network to assist during post-processing by helping to string together likely sequences of letters to form words.

CHAPTER 2

RELATED WORK

The problem of segmentation in recognizing cursive handwriting also exists in voice recognition [5]. Hidden Markov models, or HMMs, have been used for voice recognition systems with great success. This has inspired their application to cursive ICR problems [4]. Much of this work has been focused on recognizing text in on-line systems, such as those that are used in tablets for converting handwritten notes to digitized text in real time [6][7]. Their success in recognizing off-line text has been limited by their loss of long-range contextual information [5].

A group led by Dr. Jürgen Schmidhuber of the Swiss Institute for Artificial Intelligence has developed a type of recurrent neural network (RNN) which is called a long short-term memory (LSTM) [8]. The LSTM architecture allows the network to incorporate long-range contextual information about the text while scanning in both directions. Unlike other types of RNNs, the LSTM can be trained on whole words rather than individual characters. This allows them to recognize and segment simultaneously, thus circumventing Sayre's paradox. Using a dictionary and natural language processing, the researchers have been able to achieve word accuracies of 74.1% on off-line writing and 79.7% with on-line writing. They performed their experiments using the IAM-DB database of 13,040 lines of text written by 283 writers and presented their results in 2009 [5].

One of the difficulties in training handwriting recognition systems is the time-consuming task of segmenting and annotating the training data. One way to reduce the human labor required for annotation is to combine supervised training using annotated data with

unsupervised training. The basic approach is to find words that can be recognized with a high confidence score and then use those words for retraining the network. In 2010, Horst Bunke and Volkmar Frinken reported on such a system using a language model to assist in retraining and finding additional words to repeat the process [9]. In 2012, the same team presented a language-independent semi-supervised training model based on keyword searches [10]. They achieved 71% word accuracy using this method to train a bidirectional LSTM network.

So far, LSTM networks have achieved the highest word accuracy rates for purely offline handwriting recognition. However, much better performance can be obtained when segmentation can be carried out online prior to offline recognition. In this case, a segment refers to a pen stroke as part of a character, rather than a whole character bounded by dividing points separating it from its neighbors. Pen strokes are analyzed in real time and segments are identified by abrupt changes in the pen's motion. A single character may be comprised of one to three, occasionally four, segments. The segments are grouped into candidate characters and then analyzed offline by a feed-forward convolutional neural network. A word accuracy rate of 92.2% has been reported for this hybrid online-offline approach [11].

Convolutional neural networks have been very successful in identifying individual handwritten characters. By organizing multiple CNNs into a committee, training the CNNs in parallel with each CNN trained to recognize characters of different widths, and averaging the committee output during testing, character recognition error rates as low as 0.27% have been reported [12]. However, while this error rate is extremely small, it is for individual character recognition. By focusing on individual characters, the problem of Sayre's paradox is pushed

aside. Therefore, these results are not meaningful in relation to the recognition of whole words written in cursive.

So, in addressing the problem of recognizing freeform cursive handwriting in a purely offline environment, the bidirectional LSTM-based neural network is still the best performer, at 74.1% word accuracy. Clearly, that leaves room for improvement.

CHAPTER 3

CLUSTERED NETWORK

This thesis examines the effectiveness of using a relatively simple feedforward neural network to recognize cursive handwriting. The network has an architecture in which the hidden nodes are arranged in clusters, with each cluster trained to specialize in the recognition of a particular bigram of characters. The network trains on, and attempts to recognize, overlapping bigrams, with the goal that doing so will allow contextual information not only to be utilized by the network, but also in post-processing to assist in assembling strings of characters to form words and sentences.

3.1 Preprocessing

The corpus is comprised of eight different sentences, each written three times by three different people. Altogether, there are 486 scanned words made up of a vocabulary of 41 words and 145 unique bigrams. The sentences are written with black ballpoint pens on plain white paper. The samples are then scanned at 300 dpi and saved as RGB .bmp files.

Some modest preprocessing is performed on the samples, as shown in Figure 3.1. The contrast between black and white is adjusted so that all samples exhibit similar light-to-dark ratios. Line skew and word skew are corrected so that all the words in a sentence are on an even horizontal line. Slanting letters are straightened as much as possible on a per-word basis. Finally, spacing between words is equalized and the images are scaled to a height of 32 pixels, with the main body of the text occupying approximately 12 pixels of height and with the ascenders and descenders each occupying 10 pixels.

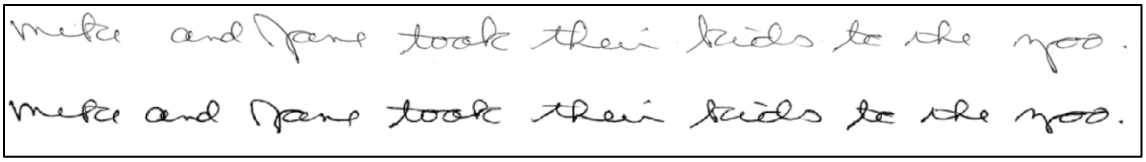


Figure 3.1 The upper line shows a handwritten sentence as originally scanned at 300 dpi, while the lower line shows the same sentence after preprocessing.

3.2 Viewport

In both the training and testing phases of the program, the preprocessed text is scanned by moving a viewport from left to right across the sample of handwriting. The viewport consists of a rectangular window of fixed height and variable width that provides snapshots of portions of the samples to be processed by the network. For each sample the program starts the scan with the left edge of the viewport aligned with the left edge of the sample. Since the preprocessed samples are a uniform 32 pixels in height, the viewport's height is fixed at 32 pixels, with the top and bottom edges aligned with the sample being scanned. At the conclusion of a scan the right edge of the viewport is aligned with the right edge of the sample.

The training samples are annotated by hand to identify characters, including spaces and punctuation, and to indicate horizontal start and end points for each character. The supervised training algorithm uses the start and end points to set the position and width of the viewport so that it is always viewing a complete bigram of characters. As the viewport moves to the right from one bigram to the next, successive bigrams overlap with one another by one character. Also, a record is kept of the minimum and maximum viewport widths ($w_{\min}$ and $w_{\max}$); this information is useful during the testing phase.

The testing samples are pre-segmented as described earlier. The points found during pre-segmentation, along with $w_{\min}$ and $w_{\max}$, determine the placement of the viewport's left

and right edges as it moves along the sample. The movement of the viewport resembles an inchworm making its way along the line of text from left to right. The viewport begins with its left edge aligned with the leftmost segmentation point (typically the leftmost column of pixels) of the sample, and its right edge aligned with the first segmentation point to the right that results in the viewport having a pixel width of at least $w_{\min}$. A snapshot is taken at this viewport configuration. Then the viewport expands by moving its right edge to the next segmentation point that is no more than $w_{\max}$ pixels to the right. Another snapshot is taken at this new viewport configuration. Once the viewport has expanded such that the next segmentation point to the right is more than $w_{\max}$ pixels away from the left edge of the viewport, the whole viewport shifts to the right so that the left edge is aligned with the next segmentation point to the left of its original position, while its width contracts so that the right edge is at the first segmentation point that is at least $w_{\min}$ pixels to the right. The process of expanding the viewport width repeats, with snapshots being taken at each position/width configuration. This process continues until it is no longer possible to position the viewport so that both edges are at segmentation points and its width is within the range defined by $w_{\min}$ and $w_{\max}$.

3.3 Observers

Each snapshot image from the viewport is processed by a group of observers that manipulate the images and measure how the images are affected. Each observer is comprised of three pairs of elements – a pair of masks, a pair of modifiers, and a pair of features. Each mask covers some portion of the image, and is often paired with a second mask that is its complement. For example, if the first mask covers the middle third of the image from top to

bottom, then its complement covers the left and right thirds of the image and leaves the middle third exposed. Not all pairings are this way; some of them include a mask that covers none of the image, leaving the entire image exposed, while its paired mask covers some portion of the image. In some cases, both masks leave the entire image exposed. Appendix A shows all of the mask-modifier-feature combinations that were used in this study, and every combination listed in the Appendix was used for each step during the training and testing phases.

A modifier is used to perform some operation on the image, such as filling in the area below the handwriting, or flipping the handwriting over and subtracting the flipped image from the original. As with masks, modifiers are often paired as complements. For example, if one modifier fills the area below the text, then its complement fills the area above the text. In some cases, one of the modifiers does nothing, leaving the image in its original state.

A feature is simply some aspect of the image that is to be quantified. The first three features are similar to those used by Dr. Schmidhuber's group [5]. Additional features were selected, and some others were eliminated, based on testing done on segmented samples using the Weka machine learning toolkit. For the experiments discussed in this thesis, the following five features are used:

1. Mean intensity
2. Horizontal center of gravity
3. Vertical center of gravity
4. Standard deviation of the mass around horizontal center of gravity (measures how loosely the mass is clustered around the center of gravity)
5. Standard deviation of mass around vertical center of gravity

Before a feature is measured the image is inverted so that the handwriting is white on a black background. Areas filled by a modifier also become white as a result of inversion. The monochromatic pixel values, which are integers ranging from 0 to 255, are converted to real values in the range of 0.0 to 1.0, respectively.

All features are designed to produce values in the range of 0.0 to 1.0. as follows:

- Mean intensity – A blank image has a mean intensity of 0.0, and a solid filled image has a mean intensity of 1.0

- Horizontal COG – A line along the left edge of the image has a horizontal COG of 0.0, a vertical line though the center of the image has a horizontal COG of 0.5, and a line along the right edge has a horizontal COG of 1.0

- Vertical COG – A line along the bottom of the image has a vertical COG of 0.0, a horizontal line through the center of the image has a vertical COG of 0.5, and a line along the top of the image has a vertical COG of 1.0

- Horizontal standard deviation – Two vertical lines, one running up the left edge and the other running up the right edge, have a horizontal standard deviation of 1.0 since the actual mass is as far from the horizontal center of gravity as possible. A single vertical line anywhere in the image has a standard deviation of 0.0 since its mass is all concentrated at the horizontal center of gravity.

- Vertical standard deviation – Two horizontal lines, one running along the bottom edge and the other running along the top edge, have a vertical center of gravity of 1.0. A single horizontal line anywhere in the image has a standard deviation of 0.0.

The final value that was generated by each observer is the delta of the values generated by the two sets of elements. The delta was compressed to fit into the range of 0.0 to 1.0 using the equation

$$\Delta v = \frac{v_2 - v_1 + 1.0}{2} \quad (1)$$

where v_1 was the value from the first set of elements and v_2 was the value from the second group of elements. Thus, the delta value is a measure of how the observer changed the image.

Appendix A gives details and examples of the observers used for this thesis. Altogether, there were 24 observers, each with a unique pairing of masks, modifiers, and features. Prior to setting up the network, different sets of observers were tested using the Weka machine learning toolkit. The observer groupings were evaluated on segmented sets of bigrams with their delta values being used to drive a support vector machine. This helped eliminate some of the weaker observers up front. Further reductions were made during network testing.

3.4 Input Nodes

The delta values are fed directly into the input nodes, with each of the 24 observers feeding one corresponding input node. The purpose of the input nodes is to distribute the observer delta values to the nodes in the hidden layer. The manner in which this occurs differs between the training and testing phases. During the training phase the identities of the bigrams being observed are known, along with their segmentation points; this information is passed along with the delta values to the hidden node defined for that bigram, with each of the 145 bigrams having exactly one hidden node. During the testing phase the delta values are distributed to all hidden nodes.

3.5 Hidden Nodes

Recent brain research involving looking at activity at the level of individual neurons has suggested that thinking about closely related topics stimulates tightly packed clusters of

neurons in the brain, while changing to an unrelated topic stimulates a more distant cluster of neurons [13]. In other words, logically related topics tend to stimulate physically related neurons.

This work inspired the architecture of the hidden layer of the neural network that is central to this thesis. Each node in the hidden layer is actually a cluster of nodes dedicated to recognizing a unique bigram of characters. Figure 3.2 shows a possible network topology following training, simplified to recognize only three bigrams with just two input nodes. The top cluster is trained to recognize the bigram "th," while the middle and lower clusters are trained to recognize the bigrams "he" and "e ," respectively.

Within each bigram cluster are subclusters of evaluator nodes, with one subcluster associated with each input node. The number of evaluator nodes is the same in each subcluster within a cluster, and that number is determined by the variability in the samples for that bigram and by a global selectivity parameter σ_T set by the user (the actual number of evaluators in any given subcluster ranged from one to nine in the experiments). Each evaluator node stores two values - the mean (μ) of the incoming delta values accepted by that evaluator during training from the input node, and the standard deviation (σ) of those same delta values – which define a Gaussian probability density function for that particular evaluator. Each evaluator also tracks a threshold (t). The threshold, which depends on σ_T and on the μ and σ of the incoming values read so far, is used during training to decide when additional evaluators need to be created. New evaluators are always created as sets so that each evaluator in one subcluster has a corresponding evaluator in all of the other subclusters within that cluster.

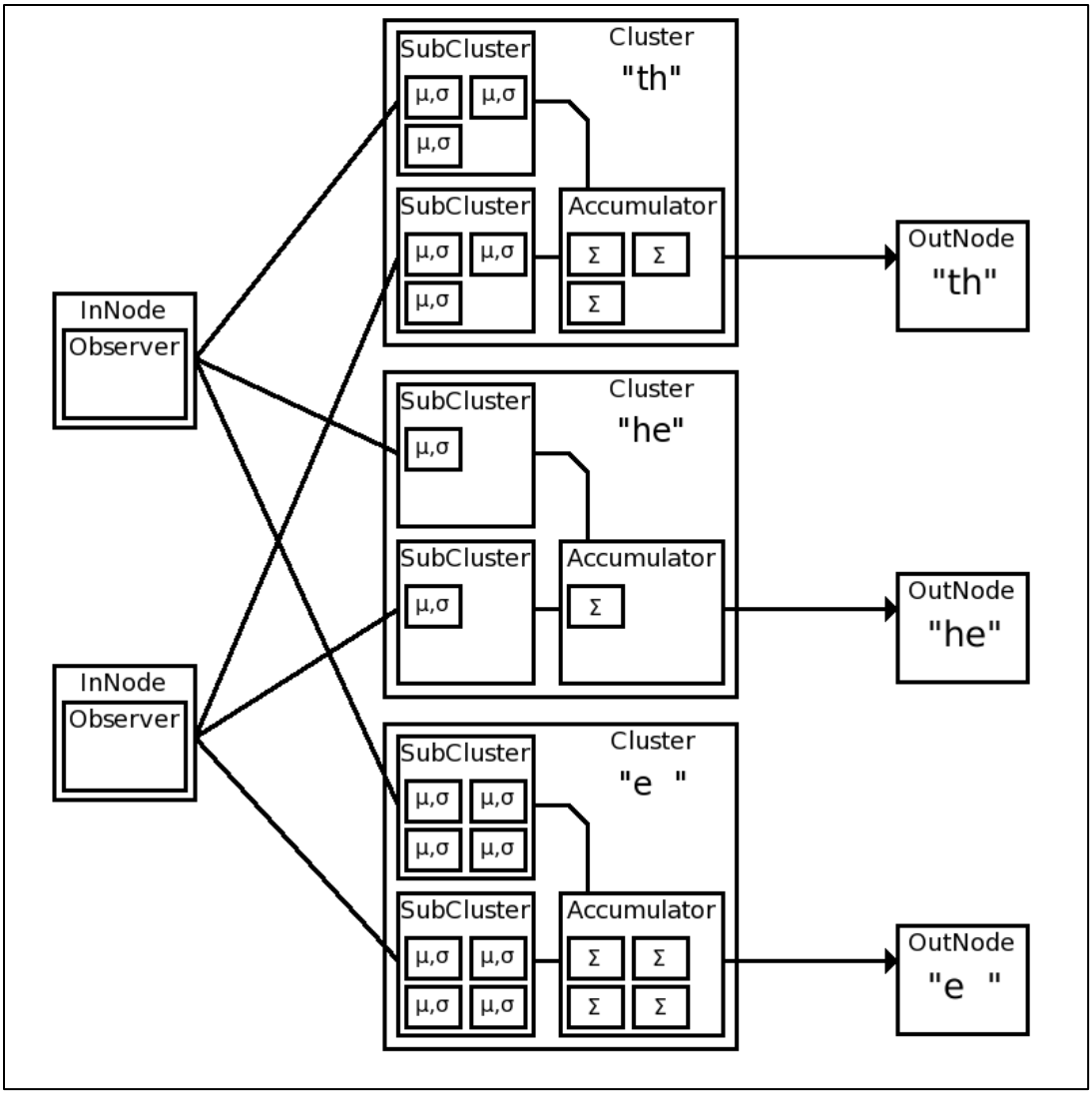


Figure 3.2 This diagram shows a simplified network after training. The network used in the experiments featured 24 input nodes and 145 output nodes, with as many as 9 evaluators in a subcluster.

A mature bigram cluster can be represented as a matrix of evaluators (**e**) where $\mathbf{e}_{j,k}$ is evaluator k within subcluster j. Each row in the matrix represents a subcluster of evaluators, while each column represents a set of evaluators created together in response to a training sample. For example, a mature cluster for bigram b having five input nodes and three evaluator sets would look like the following:

(2) $$\mathbf{C}_b = \begin{bmatrix} \mathbf{e}_{0,0} & \mathbf{e}_{0,1} & \mathbf{e}_{0,2} \\ \mathbf{e}_{1,0} & \mathbf{e}_{1,1} & \mathbf{e}_{1,2} \\ \mathbf{e}_{2,0} & \mathbf{e}_{2,1} & \mathbf{e}_{2,2} \\ \mathbf{e}_{3,0} & \mathbf{e}_{3,1} & \mathbf{e}_{3,2} \\ \mathbf{e}_{4,0} & \mathbf{e}_{4,1} & \mathbf{e}_{4,2} \end{bmatrix} .$$

Each evaluator is in turn a vector of μ, σ, and t contained within it:

(3) $$\mathbf{e}_{j,k} = \begin{pmatrix} \mu_{j,k} \\ \sigma_{j,k} \\ t_{j,k} \end{pmatrix} .$$

3.5.1 Training the Hidden Nodes

During training, the first instance of a bigram b results in the creation of a single evaluator node within each subcluster associated with that bigram. In other words, the first set of evaluators { $\mathbf{e}_{0,0}$, $\mathbf{e}_{1,0}$, … , $\mathbf{e}_{n,0}$ } is created for that cluster. If we use $\Delta v_{j,b,0}$ to denote the observer delta value coming into evaluator node $\mathbf{e}_{j,0}$ from input node i_j while observing bigram b, then initial mean $\mu_{j,0,0}$ and initial standard deviation $\sigma_{j,0,0}$ are determined as follows:

(4) $$\mu_{j,0,0} = \Delta v_{j,b,0}$$

and

(5) $$\sigma_{j,0,0} = 0.0350 ,$$

where $\sigma_{j,0,0}$ is an initial standard deviation defined as a program parameter whose value has been determined empirically. The initial threshold $t_{j,0,0}$ is calculated from the Gaussian probability density function

(6) $$t_{j,0,0} = \frac{1}{\sigma_{j,0,0} \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \sigma_T^2} \quad ,$$

where the selectivity parameter σ_T is defined as

(7) $$\sigma_T = 1.0300 \quad .$$

This value, like the initial standard deviation, has been determined empirically. More will be said about this in the experimental results section.

After the first training bigram has been measured by all observers and evaluated, each subcluster within that bigram's cluster will have exactly one evaluator node, each with $\mu_{j,0,0}$, $\sigma_{j,,0,0}$, and $t_{j,0,0}$ determined by the measured delta value $\Delta v_{j,b,0}$ provided by the observer feeding that subcluster. Our five-input cluster matrix example would then look like the following:

(8) $$\mathbf{C}_b = \begin{bmatrix} \mathbf{e}_{0,0} \\ \mathbf{e}_{1,0} \\ \mathbf{e}_{2,0} \\ \mathbf{e}_{3,0} \\ \mathbf{e}_{4,0} \end{bmatrix} \quad .$$

Other bigrams are read from the training samples and are processed by their respective clusters in the same way.

Eventually, a new instance of bigram b will appear in the training samples. In response to this new instance, each observer will generate a new delta value $\Delta v_{j,b,1}$ which will differ slightly from the value generated for the first instance of b. The magnitude of the difference will depend on the variation in the handwriting. Each evaluator in the set { $\mathbf{e}_{0,0}$, $\mathbf{e}_{1,0}$, … , $\mathbf{e}_{n,0}$ } will calculate a score using the Gaussian function

(9) $$s_{j,0,1}(\Delta v_{j,b,1}, \mu_{j,0,0}, \sigma_{j,0,0}) = \frac{1}{\sigma_{j,0,0} \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \left(\frac{\Delta v_{j,b,1} - \mu_{j,0,0}}{\sigma_{j,0,0}} \right)^2} \quad .$$

If the mean of scores $s_{j,0,1}$ is greater than the mean of thresholds $t_{j,0,0}$, then the existing set of evaluators will be updated as follows:

(10) $$\mu_{j,0,1} = \frac{\mu_{j,0,0} \cdot 1 + \Delta v_{j,b,1}}{1} \quad ,$$

(11) $$\sigma_{j,0,1} = \frac{1}{2} \cdot \left(\sigma_{j,0,0} + \sqrt{\frac{\left(\mu_{j,0,1} - \mu_{j,0,0}\right)^2 + \left(\Delta v_{j,b,1} - \mu_{j,0,0}\right)^2}{2}} \right) \quad ,$$

(12) $$t_{j,0,1} = \frac{1}{\sigma_{j,0,1} \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2} \cdot \sigma_T^2} \quad .$$

On the other hand, if the mean of scores is not greater than the mean of thresholds, then a new set of evaluators { $\mathbf{e}_{0,1}$, $\mathbf{e}_{1,1}$, … , $\mathbf{e}_{n,1}$ } will be added to the cluster for bigram b. Each evaluator in the new set will be initialized in the same manner as with the first evaluator set, while the existing set will be unchanged. In the case of our five-input cluster example, its matrix representation would then be

(13) $$\mathbf{C}_b = \begin{bmatrix} \mathbf{e}_{0,0} & \mathbf{e}_{0,1} \\ \mathbf{e}_{1,0} & \mathbf{e}_{1,1} \\ \mathbf{e}_{2,0} & \mathbf{e}_{2,1} \\ \mathbf{e}_{3,0} & \mathbf{e}_{3,1} \\ \mathbf{e}_{4,0} & \mathbf{e}_{4,1} \end{bmatrix} \quad .$$

As training continues, other instances of bigram b are likely to be observed. For any such instance, if the mean of scores $s_{j,k,n}$ is greater than the mean of thresholds $t_{j,k,\,n-1}$ for any of the evaluator sets in that cluster, then the evaluator set that produces the highest mean score will be updated. Otherwise, a new evaluator set will be created.

In the experiments, the training sets were scanned twice. At the conclusion of the training phase, each cluster in the hidden layer of the network is comprised of subclusters of evaluators that are tuned to measure features suitable for identifying a particular bigram.

3.5.2 Optimizing the Hidden Nodes

Before entering the test phase, some network optimization is performed. Not all observers are equally adept at distinguishing between different bigrams. If the delta values generated by an observer looking at different bigram types are similar, then the discernment performance of that observer is weak and its influence on the final network output should be reduced. On the other hand, if different bigram types cause the observer to generate widely varying delta values, then its discernment capabilities are strong and its influence should not be reduced. Since each observer is associated with one input node and each input node is associated with one subcluster per cluster, the output score of each subcluster j associated with an input node i_j is weighted by taking the mean of delta values for each bigram type and calculating the standard deviation of those means across all bigram types. This weight w_j is applied to the output of each subcluster going into the accumulators.

Optimizing the hidden nodes resulted in a dramatic improvement in network performance. Without optimization the word accuracy rates were typically less than 40%. Also, several observers were eliminated completely after noting their small weights following optiimization. Doing so reduced the computational load and actually resulted in a slight improvement in accuracy.

3.5.3 Testing the Hidden Nodes

During the test phase the identity of the viewport content is unknown. In fact, most of the time the viewport is not aligned with any actual bigram in the written text. The goal is that as it scans a test sample, moving from left to right and trying different widths as it goes (see section 3.2 for details), the network will interpret most of what is being viewed as low-level noise, with an actual bigram identity rising above the noise floor as the viewport comes closer to aligning with a true bigram. This is analogous to the sound coming from an analog AM receiver as the tuning dial is moved; the listener hears static and garbled sounds transitioning to a loud and clear signal as the tuner approaches an active broadcast frequency.

The observers generate delta values and submit them to their corresponding input nodes in the same way they did during training. However, because the viewport content is unknown, instead of passing the delta values only to the cluster associated with a specific bigram, they pass the values to all clusters in the hidden layer. Referring back to the diagram in Figure 3.2, each evaluator **e** has a mean μ and a standard deviation σ associated with it as determined during training. These will remain fixed throughout the test phase. The threshold t plays no role in the test phase.

More formally, let the observed delta value Δv_j (the subscript b has been omitted since the bigram identity, if it exists, is unknown) be passed into input node i_j. This value will be distributed to subcluster j in each cluster $\mathbf{C}_b$ (we employ the b subscript here since each cluster is associated with a known bigram even if the incoming delta value is not). If we let $\mathbf{e}_{j,k}$ be an evaluator of set k within subcluster j, then the score $s_{j,k}$ calculated by $\mathbf{e}_{j,k}$ using its internal $\mu_{j,k}$ and $\sigma_{j,k}$ values in the Gaussian probability density function is

(14) $$s_{j,k}(\Delta v_j, \mu_{j,k}, \sigma_{j,k}) = \frac{1}{\sigma_{j,k} \cdot \sqrt{2\pi}} \cdot e^{-\frac{1}{2}\cdot\left(\frac{\Delta v_{j,b} - \mu_{j,k}}{\sigma_{j,k}}\right)^2} \quad .$$

This score is then multiplied by the subcluster weight w_j to produce the weighted score

(15) $$s'_{j,k} = w_j \cdot \left(s_{j,k}\right) \ .$$

Within each cluster, accumulator for set k (see Figure 3.2) sums the weighted scores for the evaluators in set k and divides them by the number of subclusters to get a mean weighted score for each evaluator set as follows:

(16) $$\mu_{s'_k} = \frac{1}{n} \cdot \sum_{j=1}^{n} s'_{j,k} \quad .$$

3.6 Output Nodes

The highest mean score is passed to the output node o_b associated with that cluster (and by extension, with the bigram associated with that cluster). This represents the final score for bigram b,

(17) $$S_b = MAX\left(\mu_{s'_k} \,\middle|\, k = 1 \; to \; m\right) \ ,$$

with the viewport at that position and width, and where m is the number of evaluator sets per subcluster within the cluster $\mathbf{C}_b$.

This process is performed for each bigram. The top ten bigrams for each viewport configuration are output to a text file, with one line representing one bigram score and one viewport configuration, in the following format:

Origin MinSegment MaxSegment Termination Bigram Score

where

- *Origin* is the pixel column of the left edge of the viewport
- *MinSegment* is the minimum expected pixel column segmenting this bigram of characters, based on the training samples
- *MaxSegment* is the maximum expected pixel column segmenting this bigram of characters, based on the training samples
- *Termination* is the pixel column of the right edge of the viewport
- *Bigram* is the identity of the bigram represented by this line
- *Score* is the score S_b for this bigram

3.7 Pruning the Output

One line of text in the output file represents one candidate bigram at one viewport configuration. The top ten candidates are posted for each viewport configuration. Since a viewport configuration consists of one viewport width at one viewport position (origin), and there are several possible widths at each position and many possible positions on a scanned line of handwriting, this output file is huge. One file had almost 10,000 lines just for the word "Mike". It would have been many times larger if the test samples had not been pre-segmented (see sections 1.3 and 3.2).

To make the network output more manageable for post-processing, it is necessary to prune it. Fortunately, much of the output can be easily identified as chaff and discarded. The 10,000 lines mentioned above were cut in half by pruning. To be retained through the pruning process a line must be both lexically and geometrically compatible with any other retained lines above and below it. The rules for compatibility are as follows:

1. Line A is lexically compatible with line B below it if the second character in bigram A is the same as the first character in bigram B.

2. Line B is lexically compatible with line A above it if the first character in bigram B is the same as the second character in bigram A.

3. Line A is geometrically compatible with line B below it if the *Termination* of line A lies within the range of *MinSegment* and *MaxSegment* of line B, inclusive.

4. Line B is geometrically compatible with line A above it if the *Origin* of line B lies within the range of *MinSegment* and *MaxSegment* of line A, inclusive.

5. If the origin of line A is within two pixels of the beginning of the scan, then it does not need to be compatible with any lines above it.

6. If the termination of line B is within two pixels of the end of the scan, then it does not need to be compatible with any lines below it.

The following is a very condensed sample of a network output file that has not been pruned:

Orig	MinSeg	MaxSeg	Term	Bigram	Score
1	31	45	72	“un”	0.0272
1	39	44	58	“we”	0.0258
1	47	53	71	“we”	0.0270
2	14	20	37	“ex”	0.0234
7	14	20	22	“us”	0.0180
19	43	49	62	“wa”	0.0263
27	46	50	70	“ee”	0.0286
42	65	74	86	“ne”	0.0251
43	62	67	88	“ee”	0.0222
44	58	62	77	“ee”	0.0255
58	75	83	109	“ek”	0.0295
60	77	84	109	“eh”	0.0237
69	82	89	109	“ek”	0.0288
77	90	99	108	“li”	0.0207

Figure 3.3 This sample shows a few lines selected from a network output file.

This sample is represented graphically in Figure 3.4. In the diagram, each horizontal bar represents one line from the output file. The ends of each bar correspond to the origin and

termination (first and fourth columns in the output file) of that line. The white gap within each bar corresponds to the segmentation range (second and third columns) of that line. Each bar is also labeled with a bigram and the score for that bigram.

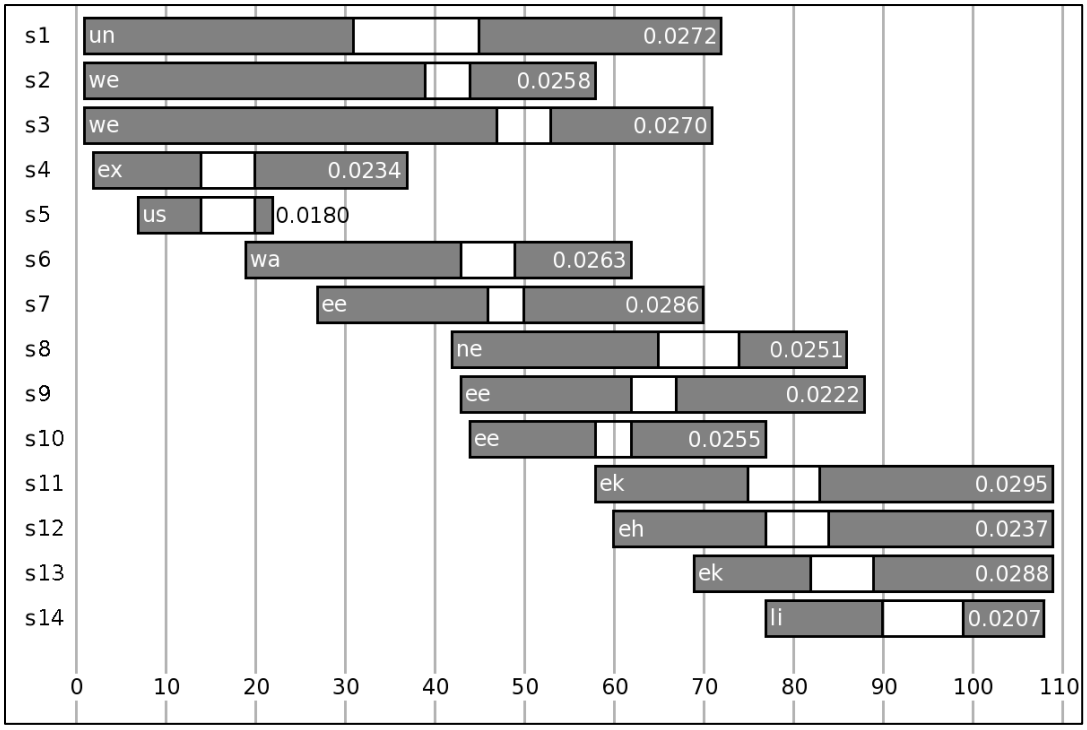


Figure 3.4 This diagram represents network output prior to pruning.

Line s1 in Figure 3.4 is compatible with line s8 because they share the common character *n*, the termination of s1 lies within the segmentation range of s8, and the origin of s8 lies within the segmentation range of s1. Similarly, s8 is compatible with s13. Thus, s1, s8, and s13 together form a string that runs from the left end of the scan to the right end. On the other hand, s7 is not geometrically compatible with any line before it, so it is eliminated. Likewise, s6 is geometrically compatible with s4 and s5 before it, and s9 and s10 after it, but it is lexically compatible with none of these. Figure 3.5 shows the results of pruning the output.

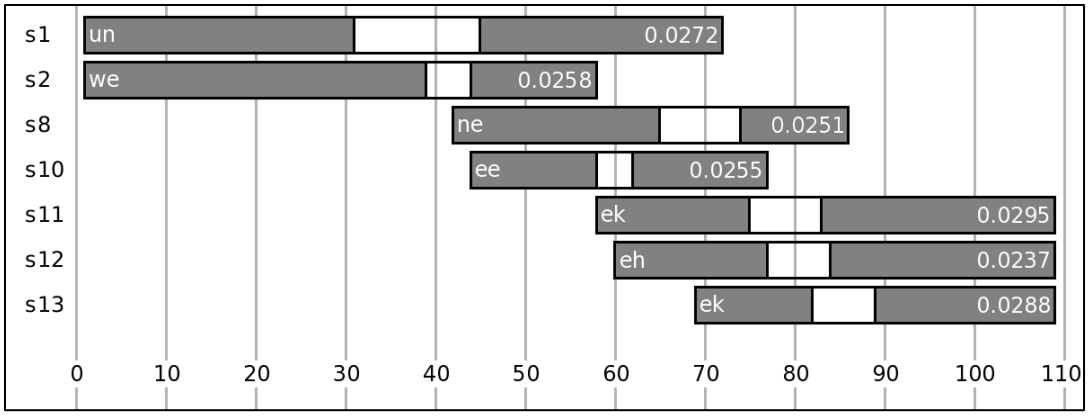


Figure 3.5 This diagram shows the same output after being pruned.

The rules for pruning ensure that every line that remains in the output file represents a bigram that can be linked with other bigrams in the file to form a continuous string of characters from the beginning of the file to the end. The only problem that remains is finding the best such string.

3.8 Searching the Output for the Best Solution

The primary focus of this thesis is the development of an appropriate neural network architecture. Nevertheless, considerable time and effort went into finding a satisfactory way to convert the pruned network output into finished strings of characters. The method described here starts by transforming the contents of a pruned network output file into a directed-acyclic graph, or DAG. The graph in Figure 3.6 represents the output from Figure 3.5.

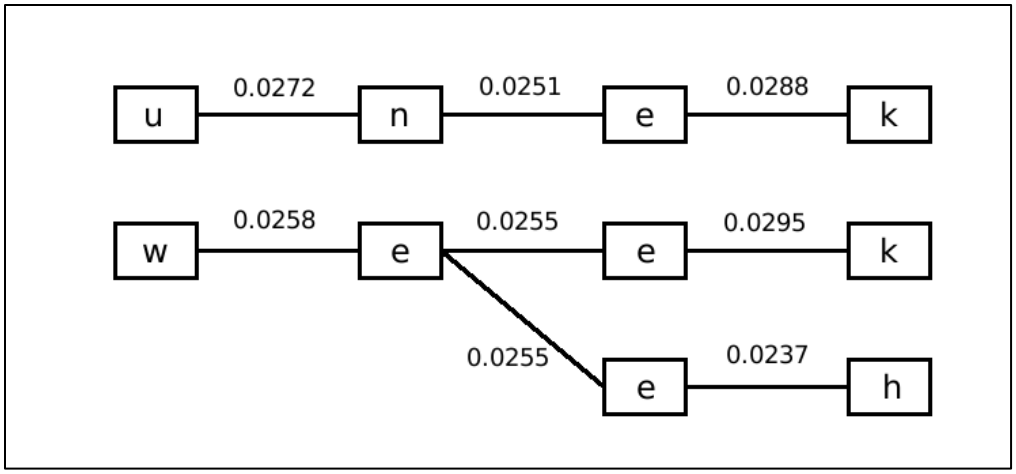


Figure 3.6 This diagram shows the pruned output represented as a directed-acyclic graph.

The transformation process starts by creating a vertex for each line in the pruned output, and storing the first character of the line's bigram in its respective vertex. In Figure 3.5, all but the three rightmost vertices are created in this step. Next, for each compatible pair of lines in the output, the corresponding vertices are connected by an edge whose weight is the score on the first of the two lines. For example, in Figure 3.5 the weight of the edge between vertices 'u' and 'n' is 0.0272, which is the score on the line for bigram 'un' in the pruned output (s1 in Figure 3.4). Finally, for those vertices that are from terminating lines in the pruned output, add a new vertex with the second character in the bigram and connect it with and edge weighted with the score for that line.

The final step is to find the highest scoring path through the graph, where the score of a path is the mean of the weights of the edges comprising the path. Using the mean of the edge weights rather than the sum prevents the search algorithm from favoring paths with many vertices over those with few. While the graph in Figure 3.5 has only three possible paths, an

actual graph generated from a pruned output file has thousands of highly interconnected vertices, making a brute force examination and comparison of every possible path out of the question. Because of this, a simple dynamic programming approach is used to dramatically reduce the runtime of the search. It starts as a depth-first search, but any time the search ascends back up the graph to a previous vertex, the best score found from that vertex to a terminating vertex is recorded along with a pointer to the next vertex in that path and the number of vertices along that path. Any future path that leads to that same vertex can combine its score, up to that point, with the best score recorded at that vertex and know immediately if proceeding further down that path will produce a new higher score. This prevents the program from revisiting the same paths repeatedly. The highest scoring path from this search is saved as a candidate solution.

To improve the odds of finding the best solution, a second search is performed. This time, in addition to using the recorded scores to decide whether or not to continue down a path, a dictionary search is used to determine if the character in the next vertex, when added to the characters of the vertices along the path to that point, is a starting sequence of characters for a word. For example, if the path so far contains the characters ‘v’, ‘o’, and ‘t’, and the next vertex is for ‘o’, it will not be considered in this search because “voto” is not a starting sequence for any word in the dictionary. The dictionary used in the experiments contains over 1100 words, including different forms of words such as “vote,” “voter,” and “votes.” Many of the words in the dictionary do not appear as such in the corpus, but contain bigrams that do. For example, the word “votes” is not in the corpus, but the word “vote” is, and the bigram ‘es’ is found in “likes” and “yesterday”, which are in the corpus.

Once the best paths, with and without the dictionary, have been determined, they are compared. If the non-dictionary solution has a score that is, at least, 1.25 times higher than the dictionary solution, then the non-dictionary solution is retained. Otherwise, the dictionary solution, if there is one, is retained as the best solution. The bias in favor of dictionary solutions promotes correct solutions over solutions with slightly higher scores but having one or two incorrect letters. In one case, the word being scanned was “Jane,” and the search algorithm found “Jane” with a score of 0.0394 and “sene” with a score of 0.0408. Even though “sene” scored slightly higher, “Jane” was selected as the best solution due to the bias in favor of dictionary solutions. In another case, the word being scanned was “yesterday,” and the search algorithm found “zostenday” with a score of 0.0441 and “likely” with a score of 0.0250. In this case, “zostenday” was selected as the correct solution because its score exceeded the score of “likely” by more than a factor of 1.25. While “zostenday” is not the correct solution, it is arguably better than “likely” in that it more closely resembles “yesterday,” both in its spelling and its general appearance to a human reader.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Handwriting Corpus

The corpus was created by collecting cursive handwriting samples from three people. Each person wrote eight sentences three times each for a total of 72 sentences, or 24 per writer. The corpus featured a vocabulary of 41 words made up of 145 unique bigrams. The small corpus size prevents a direct comparison with the work of Dr. Jürgen Schmidhuber's team. However, it is sufficient to test the concept to see if it merits further investigation.

4.2 Selectivity Parameters

Referring back to section 3.5.1, two parameters were used to adjust the selectivity of the network, σ_0 and σ_T. Multiple experiments were run, with σ_0 values of 0.0300, 0.0350, and 0.0400, and σ_T values of 1.0200, 1.0300, 1.0400, and 1.0500. The worst combination was σ_0 = 0.0400 and σ_T = 1.0400, with a word accuracy of 63.0%. The best combination was σ_0 = 0.0350 and σ_T = 1.0300, with a word accuracy of 66.5%.

4.3 10-fold Cross Validation

All experiments were run as 10-fold cross validations. For each experiment, the corpus was divided into ten sets of sentences, with eight of the sets containing seven sentences each, and two of the sets containing eight sentences each. The sentences were assigned to the sets in a simple round-robin manner. Then a series of ten training-testing cycles were performed, with one of the ten sets of sentences being used for testing and the other nine for training. The ten

cycles concluded with each sentence having been used for testing exactly once. The results of the ten cycles were gathered into one output file for analysis. The results of the most successful experiment (with σ_0 = 0.0350 and σ_T = 1.0300) are shown in the tables follow.

Table 4.1 10-fold Cross Validation Results

Test	Correct	Out of	Accuracy (%)
1	38	57	66.7
2	34	54	63.0
3	29	44	65.9
4	28	44	63.6
5	32	48	66.7
6	37	52	71.1
7	32	49	65.3
8	26	45	57.8
9	33	45	73.3
10	34	48	70.8
Cumulative	323	486	66.5

Table 4.2 Word Accuracy by Writer

Writer	Correct	Out of	Accuracy
1	125	162	77.2%
2	94	162	58.0%
3	104	162	64.2%
Cumulative	323	486	66.5%

As is clear in Table 4.2, the results varied considerably between writers. Writer 1 was trained in the Palmer method of cursive writing and displayed a high degree of consistency from one sample to the next. Writer 2, who wrote with the left hand, wrote in a style characterized by wider variations in size, slant, and cadence. Writer 3 shows considerable variation in slant, but is otherwise consistent in form. The variation in style between the writers is apparent in Figure 4.1.

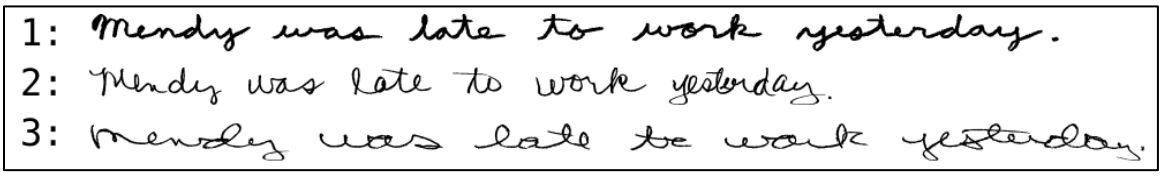


Figure 4.1 This image shows pre-processed samples of the same sentence written by three different individuals.

The solutions found for these samples, respectively, are as follows:

1. “Mendy was late to work zostenday.”
2. “Thindy sees late to work igerdaz.”
3. “Mendy none late tee work gesterday”

Note that the second and third samples score the same in terms of word accuracy (50%) even though many human readers will perceive the third one as “better” than the second.

Table 4.3 breaks down the results by vocabulary word. There is a moderate negative correlation between word identification accuracy and word length, with a Pearson’s correlation coefficient (R-value) of -0.5988, indicating a tendency to identify shorter words with greater success. However, it is clear that this correlation is not very strong since, among other

examples, “early” was identified correctly in every instance, while “do” was identified correctly in 7 out of 9 instances.

Table 4.3 Accuracy by word

Word	Correct	Out of	Accuracy (%)
early	9	9	100.0
took	9	9	100.0
zoo	9	9	100.0
jane	17	18	94.4
to	58	63	92.1
and	8	9	88.9
forget	8	9	88.9
joseph	8	9	88.9
late	8	9	88.9
music	8	9	88.9
voter	8	9	88.9
work	8	9	88.9
the	23	27	85.2
mike	15	18	83.3
do	7	9	77.8
for	7	9	77.8
kids	7	9	77.8
light	7	9	77.8
their	7	9	77.8
voting	7	9	77.8
jazz	6	9	66.7
next	6	9	66.7
not	6	9	66.7
was	11	18	61.1
begin	5	9	55.6
does	5	9	55.6

his	5	9	55.6
mendy	5	9	55.6
turnout	5	9	55.6
will	5	9	55.6
week	4	9	44.4
country	2	6	33.3
laundry	3	9	33.3
please	3	9	33.3
seldom	3	9	33.3
vote	3	9	33.3
likes	3	18	16.7
listen	3	18	16.7
elections	1	9	11.1
county	1	12	8.3
yesterday	0	9	0.0

CHAPTER 5

CONCLUSION

In this thesis, a neural network with a hidden layer based on a clustered architecture was used to identify text written in cursive. Each cluster was trained to recognize a unique bigram of characters. The clustered architecture allowed for using multiple observers to characterize the text being scanned using a variety of criteria. A structure of subclusters within each cluster provided the flexibility needed to recognize bigrams written in varying styles by different writers. This flexibility enabled the network to evaluate sections of scanned text of varying widths, thus allowing context to be considered without the need for recurrence within the hidden layer. Context consideration was further enhanced by training the network to look for overlapping bigrams of characters, rather than single characters.

Twenty-four observers were used to feed the network during both the training and testing phases. Each observer was comprised of pairs of masks, modifiers, and features as described in Appendix A. The initial group of observers was based on work done by Dr. Schmidhuber and his team in Switzerland [5]. This was expanded to a much larger set of observers, which was then culled by evaluating their effectiveness in recognizing fixed bigrams using the Weka toolkit. A final culling was done by eliminating those observers whose contributions to the overall scores was minimized by applying weights during the network optimization phase, which came between the training and testing phases. Optimization was based on an observer's ability to distinguish between bigrams by generating different scores for different bigrams. The contribution of an observer which produced similar scores for different bigrams was assigned a lower weight, while the contribution of an observer which produced

widely varying scores for different bigrams was assigned a higher weight. Removing the weakest observers, based on their optimized weights, actually resulted in a slight improvement in accuracy performance, while reducing the load on the computing resources.

With dictionary-assisted post-processing, an overall word accuracy of 66.5% was attained on a corpus of 486 words. This level of performance does not break new ground, especially on such a small corpus, but it does demonstrate that this approach has merit. Further investigation and development might push its performance significantly higher.

One obvious area for improvement is in the observers. While considerable effort was put into selecting the combinations of masks, modifiers, and features comprising the observers, they are not likely to be the optimum set of observers. One possible strategy for improving them would be to use a genetic algorithm to manipulate the masks directly, trying configurations that might not be apparent to human testers, and coupling them with randomly selected modifiers and features. This would greatly expand the search area for possible observers. However, it would require computing resources well beyond what was used for these experiments.

Another possibility for improvement is in the way the network output is interpreted. In these experiments, the score for each bigram was considered in isolation; the only relation between scores was their relative ranking. It might be worthwhile to consider how scores relate to each other prior to ranking. One way to accomplish this would be to treat the scores at the output nodes as components of a vector using the cosine distance between a vector obtained during the test phase and a mean vector associated with each candidate bigram obtained during the training phase. Those candidate bigrams whose cosine distance from the test vector

is smaller would rank higher than those whose cosine distance was greater. This approach would have the possibility of capturing relational information that is lost by considering only individual bigram scores.

Finally, the use of a dictionary to assist in post-processing, while effective, is somewhat crude. A more sophisticated natural language processing algorithm could be employed to improve the final output. Such an algorithm might take into consideration individual word frequencies, word pairing frequencies, and parts of speech, in addition to the simple existence of a word in a dictionary. However, it needs to be understood that the use of sophisticated NLP techniques introduces other issues, such as providing corrected solutions to incorrectly written sentences. For example, one might encounter the written word “here” when the correct word would have been “hear.” In this case, an NLP algorithm might be smart enough to provide “hear” as part of the solution, but is that the correct solution? In other words, do we want the system to correct grammatical errors or be true to the original written text?

In conclusion, automated offline conversion of cursive handwriting to machine searchable text has been and continues to be a challenging problem in computer science, complicated by the large variations and inconsistencies in writing styles, and by the difficulty in identifying segmentation points between adjacent characters.

APPENDIX

OBSERVERS

There are 24 unique observers, each comprised of a pair of filters. Each filter is itself a combination of a mask, a modifier, and a measured feature. All of the observers are shown here working on a sample of the bigram “ke.”

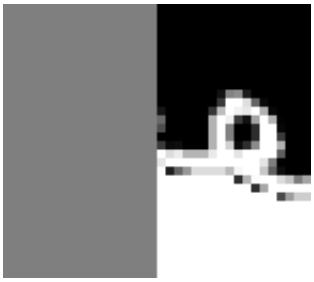
Observer 1	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Mean Intensity
Post-Mask	None
Post-Modification	Left Fill
Post-Feature	Mean Intensity
Example Delta Value	0.6768

Observer 2	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Mean Intensity
Post-Mask	None
Post-Modification	Right Fill
Post-Feature	Mean Intensity
Example Delta Value	0.7822

Observer 3	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Mean Intensity
Post-Mask	None
Post-Modification	Bottom Fill
Post-Feature	Mean Intensity
Example Delta Value	0.6950

Observer 4	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Mean Intensity
Post-Mask	None
Post-Modification	Top Fill
Post-Feature	Mean Intensity
Example Delta Value	0.6644


Observer 5	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Mean Intensity
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	Mean Intensity
Example Delta Value	0.4522




Observer 6	
Pre-Mask	Left Half
Pre-Modification	Bottom Fill
Pre-Feature	Mean Intensity
Post-Mask	Right Half
Post-Modification	Bottom Fill
Post-Feature	Mean Intensity
Example Delta Value	0.5812


Observer 7	
Pre-Mask	Left Half
Pre-Modification	Top Fill
Pre-Feature	Mean Intensity
Post-Mask	Right Half
Post-Modification	Top Fill
Post-Feature	Mean Intensity
Example Delta Value	0.4434


Observer 8	
Pre-Mask	Full
Pre-Modification	None
Pre-Feature	Horizontal COG
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	Horizontal COG
Example Delta Value	0.4732


Observer 9	
Pre-Mask	Full
Pre-Modification	None
Pre-Feature	Vertical COG
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	Vertical COG
Example Delta Value	0.5402


Observer 10	
Pre-Mask	Full
Pre-Modification	None
Pre-Feature	HCOG Diffusion
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	HCOG Diffusion
Example Delta Value	0.5368


Observer 11	
Pre-Mask	Full
Pre-Modification	None
Pre-Feature	VCOG Diffusion
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	VCOG Diffusion
Example Delta Value	0.5236


Observer 12	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Horizontal COG
Post-Mask	None
Post-Modification	Bottom Fill
Post-Feature	Horizontal COG
Example Delta Value	0.5236


Observer 13	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Horizontal COG
Post-Mask	None
Post-Modification	Top Fill
Post-Feature	Horizontal COG
Example Delta Value	0.5587


Observer 14	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Vertical COG
Post-Mask	None
Post-Modification	Left Fill
Post-Feature	Vertical COG
Example Delta Value	0.3881


Observer 15	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Vertical COG
Post-Mask	None
Post-Modification	Right Fill
Post-Feature	Vertical COG
Example Delta Value	0.4509


Observer 16	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	HCOG Diffusion
Post-Mask	None
Post-Modification	Bottom Fill
Post-Feature	HCOG Diffusion
Example Delta Value	0.7165


Observer 17	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Horizontal Diffusion
Post-Mask	None
Post-Modification	Top Fill
Post-Feature	Horizontal Diffusion
Example Delta Value	0.6886


Observer 18	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Horizontal Diffusion
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	Horizontal Diffusion
Example Delta Value	0.3488


Observer 19	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	Horizontal Diffusion
Post-Mask	None
Post-Modification	Vertical Asymmetry
Post-Feature	Horizontal Diffusion
Example Delta Value	0.3754

Observer 20	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	VCOG Diffusion
Post-Mask	None
Post-Modification	Left Fill
Post-Feature	VCOG Diffusion
Example Delta Value	0.8717

Observer 21	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	VCOG Diffusion
Post-Mask	None
Post-Modification	Right Fill
Post-Feature	VCOG Diffusion
Example Delta Value	0.9572

Observer 22	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	VCOG Diffusion
Post-Mask	None
Post-Modification	Center Fill
Post-Feature	VCOG Diffusion
Example Delta Value	0.4230

Observer 23	
Pre-Mask	None
Pre-Modification	None
Pre-Feature	VCOG Diffusion
Post-Mask	None
Post-Modification	Horizontal Assymetry
Post-Feature	VCOG Diffusion
Example Delta Value	0.4685

Observer 24	
Pre-Mask	Left Half
Pre-Modification	None
Pre-Feature	VCOG Diffusion
Post-Mask	Right Half
Post-Modification	None
Post-Feature	VCOG Diffusion
Example Delta Value	0.5628

REFERENCES

[1] *Character Set for Optical Character Recognition (OCR-A)*, ANSI Standard INCITS 17-1981 (R2002).

[2] Rose Holley, "Analysing and improving OCR accuracy in large scale historic newspaper digitisation programs," *D-Lib Magazine*, vol. 15, no. 3/4, Mar./Apr., 2009.

[3] Kenneth M. Sayre, "Machine recognition of handwritten words: a project report," *Pattern Recognition*, vol. 5, pp. 213-228, May, 1973.

[4] J. Hu *et al.*, "Writer independent on-line handwriting recognition using an HMM approach," *Pattern Recognition*, vol. 33, no. 1, pp. 133-147, Jan., 2000.

[5] Alex Graves *et al.*, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, May 2009.

[6] T. Starner *et al.*, "Online cursive handwriting recognition using speech recognition techniques," *International Conference on Acoustics, Speech and Signal Processing*, vol. 5, pp. 125-128, 1994.

[7] S. Bercu and G. Lorette, "On-line handwritten word recognition: an approach based on hidden Markov models," *Proc. 3rd Int. Workshop on Frontiers in Handwriting Recognition*, pp. 385-390, 1993.

[8] Sepp Hochreiter, and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, 9(8):1735-1780, 1997.

[9] Volkmar Frinken and Bunke Horst, "Self-training for handwritten text line recognition," *15th Iberoamerican Congress on Pattern Recognition*, pp. 104-112, 2010.

[10] Volkmar Frinken *et al.*, "Semi-supervised learning for cursive handwriting recognition using keyword spotting," *2012 International Conference on Frontiers in Handwriting Recognition*, pp. 49-54, 2012.

[11] Aiquan Yuan *et al.*, "Handwritten English word recognition based on convolutional neural networks," *2012 International Conference on Frontiers in Handwriting Recognition*, pp.207-212, 2012.

[12] Dan Claudiu Ciresan *et al.*, "Convolutional neural network committees for handwritten character classification," *2011 International Conference on Document Analysis and Recognition*, pp. 1135-1139, 2011.

[13] Nanthia Suthana and Itzhak Fried, “Percepts to recollections: insights from single neuron recordings in the human brain,” *Trends in Cognitive Sciences*, vol. 16, no. 8, pp. 427-436, Aug. 2012.