Samuel Krasnik
CS674 Natural Language Processing - Spring 2004

On-line Handwritten Word Recognition

Final Report

1. Introduction

This report presents a system which is able to recognize handwritten words from a lexicon in an on-line setting with high accuracy and is able to deal with highly deformed and highly ambiguous words. The report proceeds by introducing the problem, contrasting the problem being solved by the presented system with the off-line version of the problem, presenting related work, defining algorithms used by the system, motivating and describing new algorithms created for use with the presented system, evaluation of the system, stating future work and summarizing the presented work.

2. Problem Definition

2.1 Task Definition

The problem of on-line handwriting recognition can be defined as follows. A handwriting recognition system is presented with a handwritten word, usually on a digitizer tablet with a pen device in the form of a sequence of spatial and temporal coordinates. The system is allowed to utilize either a grammar for proper word formation or a lexicon of allowed words. The presented system utilized a lexicon instead of a grammar. The output of the system must be the most likely sequence of characters the input is likely to have been, if a grammar is being used, or if a lexicon is being used, the most likely word in the lexicon corresponding to the input.

Written languages consist of a symbol set (the alphabet) which combine to make strings (words). Current recognizers can handle both single word and multi-word recognition. Handwriting consists of a sequence of strokes, where each stroke is from pen down to pen up. Individual elements from the alphabet are usually written sequentially, but there are exceptions (such as crossing “t”s and dotting “i”s). While the core algorithms used by most recognizers are not language or alphabet specific, the presented system will focus on the recognition of individual English words.

2.2 On-line vs. Off-line HWR

The key difference between the on-line and off-line versions of the problem is that in the on-line version, the machine recognizes the user's handwriting as the user is writing. This condition has led the problem to be called *real time* handwriting recognition, imposing a constraint on the allowable time for completion of recognition. Advantages of the on-line recognizer is that the algorithms have access to time sequence information as well as the number of strokes, the direction of writing and the speed of writing (Tappert et. al).

2.3 Sub-Problems

The problem of handwriting recognition is usually divided into four subproblems: preprocessing (which can include segmentation), individual character recognition, candidate string generation, and post-processing.

Pre-processing consists of noise reduction and may involve segmentation of strokes into potential characters. Since multiple characters can, and in the case of cursive writing are, written in a single stroke, segmentation is necessary to feed the segmented strokes into the character recognizer. The earliest form of segmentation consisted of an explicit signal from the user. Another alternative is to force the user to write each character as a single stroke or in a box such as often appears on a form. However, this restriction is unrealistic and often slows the user. Therefore, most modern recognizers do segmentation implicitly-- it is the result of recognition as opposed to a preprocessing step.

Another problem that arises is noise in the input. While many character recognizers are trained to deal with noisy input, much of the unwanted noise can be eliminated manually without affecting the features used by the character recognizer. First, smoothing is usually performed on the input. Next, filtering eliminates duplicate points. Dehooking can remove irrelevant hooks that occur at the beginnings of strokes as the pen is placed on the surface of the tablet. Dot reduction reduces dots, such as those present on “i”s and “j”s to individual points. Stroke connection can eliminate accidental lifting of the pen while writing a single character. Finally, a collection of algorithms can be optionally applied to normalize the input.

There exist many methods to recognize isolated characters, both without context and with context. Some character recognizers rely on prior knowledge of the shapes of the various symbols in the alphabet such as ascenders and descenders. Others analyze a set of predefined features, most of which are static shape features and a few of which may be dynamic features. The most popular character recognizers involve matching of time sequences of coordinates, directions or dynamically processed features such as high curvature or low velocity points. One such algorithm is the elastic matching algorithm which seeks to find the optimal alignment of two time sequences of coordinates. Another now virtually abandoned algorithm involves analyzing characters based on rules of character synthesis. In the case of cursive or mixed cursive/discrete script recognition, loose stroke segmentation is often used to generate possibilities for segmenting strokes and the segmentations are then compared in post-processing. Such segmentations were based on either high curvature points or low-velocity points in the strokes.

Candidate string selection can be accomplished either explicitly or implicitly while selecting the optimal candidate strings. The most popular methods use dynamic programming on the lexicon to find the word in the lexicon that best matches the input (Tappert et. al).

3. Related Work

Manke and Bodenhausen's *A Connectionist Recognizer for On-Line cursive Handwriting Recognition* presented a Neural Network method for individual character recognition. Their algorithm utilizes a sequence of time delay neural networks which are time shift independent and aligns the output of the sequence with the Dynamic Time Warping algorithm. The Dynamic Time Warping algorithm that appears in this paper as well as many others is described in detail later.

The Time Delay Neural Network (TDNN) architecture consists of input, hidden and output layers. Each node in the hidden layer is connected to a fixed sized time-window of the input layer, and each node in the output layer is connected to a fixed size time-window of the hidden layer. Thus, multiple TDNN's can be strung together for the length of the input and produce, for each element in the time sequence, a set of estimated likelihoods for all letters. In practice, 78 output neurons are used, 3 for each letter, denoting the beginning, the middle and end of each letter. This combination of the TDNN with DTW is called the Multi-State Time Delay Neural Network (MS-TDNN). Each neuron is a summing unit with a sigmoid squashing function applied to the output. Like many other multi-layer neural networks, it is trained via back-propagation with a momentum term.

The alignment of the outputs is done with the aid of a dictionary. For each word in the dictionary, the DTW algorithm is used to align the word optimally with the outputs of the MS-TDNN. The word with the smallest cumulative error in its optimal alignment is outputted as the optimal word. This reliance on a dictionary as the sole language model is the weakness of most current algorithms as they cannot recognize strings such as names that are not in the dictionary or abbreviations which humans can.

The authors evaluate both a writer dependent and writer independent version of the MS-TDNN algorithm. The size of the dictionary ranges from 400 to 20000 words. Naturally, as the dictionary size increases, the recognition rate decreases. Distinguishing words like “jog” and “joy” or “dump” and “clump” can confuse even human recognizers. The recognition rate varied between 97.7% and 83.0%. Fortunately, the future work section of the paper indicates that the authors are testing the MS-TDNN framework with an N-best dynamic programming search driven by a language model instead of a fixed dictionary. The architecture is also being used for speech recognition.

Keogh and Pazzani present a modification of the standard DTW algorithm that aligns a sequence of derivatives instead of the raw data in *Derivative Dynamic Time Warping*. The problem with the standard DTW algorithm is that it may try to explain warping of the Y-axis by warping the X (time) axis. This can lead to very non-intuitive alignments where a single point on the time series is mapped onto a long section of the other sequence. The authors call this effect a “singularity.” Most methods deal with such a problem by constraining the possible deviation from the perfect alignment. The algorithm presented by the authors operates on the approximate local derivatives of the data instead of the coordinates themselves.

The problems arise when there are small differences in the Y axis, and singularities are produced by the DTW algorithm. The DDTW algorithm first smooths the data so local derivatives can be measured more accurately. The distance measure between two data-points is the difference in their derivatives. The time complexity for the two algorithms is the same as the distance between two points is computed in constant time and the rest of the algorithm is virtually identical. All the optimizations that apply to DTW also apply to DDTW.

Experimental results show that the mean warping, defined as the average warping for all data-points in the optimal alignment, is significantly lower in the DDTW algorithm than in DTW. Across three datasets, the mean warping for DTW ranged from 0.17 to 0.24 and from 0.03 to 0.04 for DDTW. The results of the authors show that the mean warping stays the same as the size of random Gaussian bumps in the data increase, while the DTW algorithm implies much more warping in the alignment. Future work seeks to use piecewise segments and Fourier transforms instead of local derivatives, which would reduce the algorithm's complexity, since Fourier transforms with a constant number of coefficients would be used.

In *An Efficient Algorithm for Matching a Lexicon with a Segmentation Graph*, Chen et. al. propose an algorithm for efficiently finding the optimal interpretation of the aforementioned segmentation hypothesis graph. The authors make the observation that creation of a segmentation graph and using a dynamic programming approach to match each word in the lexicon grows linearly with the size of the lexicon, which means that the algorithm does not utilize common features between words in the lexicon. Furthermore, this does not make sense from the point of view of how humans do such recognition. Although an English professor will have a significantly larger vocabulary than a high school student, he reads at the same speed as the high school student.

The proposed algorithm expresses the lexicon as a trie and attempts to optimally interpret the hypothesis graph with respect to the trie. The algorithm assumes that a character recognizer has already assigned a cost to each edge in the graph. The horizontal axis in the DP table represents the transition between characters while the vertical axis represents the transition between graphemes. There are three options for each element in the DP table. Either one grapheme is matched to a character, a grapheme is omitted, or multiple graphemes are matched to a single character. Each transition is weighted by the cost of the edge in the hypothesis graph corresponding to that transition. Each entry in the table represents the cost or confidence of matching the first i characters with the first j graphemes.

In his dissertation, Jong Oh presents a complete on-line handwriting recognition engine. One of the key elements in the dissertation is the comparison between Viterbi search and Hypothesis Propagation Network search. The latter is a modification of the dynamic programming algorithm presented by Chen 2 years earlier, and also uses a trie data structure to store the lexicon. According to Oh, the Viterbi search resulted in a 73% recognition rate, while the hypothesis graph search achieved a much higher 85% recognition rate. This increase is most likely due to the lack of a 1st order Markov assumption in the hypothesis graph search.

4. Algorithms

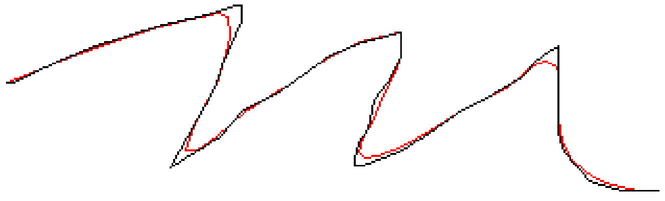

Illustration 1 Original Stroke in Black, Resampled Stroke in Red

4.1 Preprocessing

The preprocessing stage of the system consisted of smoothing the input and creating a segmentation hypothesis graph.

The smoothing algorithm is a variation of spline-resampling. The input is presented as a sequence of (x,y) coordinate pairs, divided into strokes. In order to smooth the data, each stroke is first approximated with a multi-point Bezier spline. The first and last control points of each spline are set to coincide with the endpoints of each respective stroke. Since the Bezier spline guarantees that the end points will correspond to the first and last control points, the endpoints of a spline coincide with the endpoints of the stroke it is approximating. The interior control points of the spline are set to coincide with points along the stroke separated by a predefined arc-length. After the Bezier spline is defined, the spline is then resampled at a finer resolution and a set of resampled points is generated. See Illustration 1 for sample output of the smoothing algorithm.

The segmentation algorithm creates a graph, known as a Segmentation Hypothesis Graph. This graph represents the possible ways to segment the graph. The probable segmentation points are found by identifying high-curvature points (cusps) in the input. The algorithm for finding cusps is based on the IPAN99 algorithm (Chetverikov). A point in a sequence is defined to be a *corner* if a triangle of a specified size can be inscribed in the polygon defined by nearby points. For details see Chetverikov's description of the algorithm.

Once cusps are detected, a graph is created. The nodes in the graph represent cusps. There are two unique nodes, **S** and **T**, which denote the beginning of the word and the end of the word. Edges on the graph are directed edges **(u, v, c, ε(p))**, where **u** and **v** are nodes, **c** is a character in the input alphabet, and **ε(p)** is a function of the likelihood that the stroke starting at cusp **u** and ending with cusp **v** is in fact character **c** which defines the cost of the edge. A path from **S** to **T** See Illustration 2 for a sample Segmentation Hypothesis Graph. There can be multiple edges between two nodes **u** and **v**, but only one for each **u**, **v**, and **c**. The sum of all **ε(p)** between two cusps does not need to add to 1. **ε(p)** is known as the Character Recognizer's Error Function.

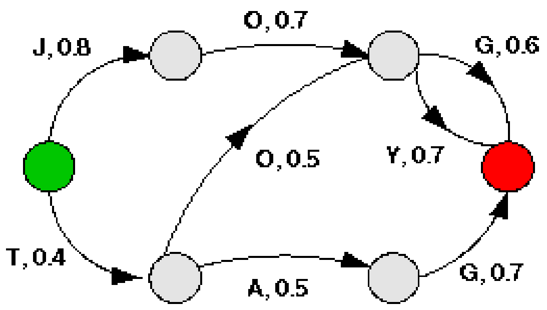

Illustration 2 S - Green; T - Red

In the Segmentation Hypothesis Graph, a path from **S** to **T** defines a character sequence Ψ, which consists of the sequence of edges traversed from **S** to reach **T**. The cost of the sequence, ε(Ψ), is defined as either the sum or the product of the functions of likelihood of the edges in the sequence. The presented system uses the sum. Therefore,

$$\varepsilon(\Psi) = \sum \varepsilon(\mathrm{p}) \mid (u \;\; , v , c , \varepsilon(\mathrm{p})) \in \Psi$$

The goal of the system, is therefore, given a Segmentation Hypothesis Graph Γ, to find the optimal path Ψ', such that

$$\Psi' = argmin_{\Psi \in \Gamma} \;\; \varepsilon(\Psi)$$

4.2 Character Recognition

4.2.1 Neural Network Character Recognition

During the development of the presented system, the character recognizer component was found to be the most critical to the success and performance of the system as a whole. Therefore, the algorithms used for character recognition will be described in more detail than the rest of the algorithms in this report.

Three different algorithms were tried for character recognition. The first was a neural network approach. The network architecture consisted of three layers with each node in the network functioning as a summing unit with a biased sigmoid squashing function applied to the output. The topology of the network is as follows: the input layer to the network consisted of 120 units for spatial coordinates and 40 nodes for temporal information. In addition to these local features, the input layer also included 9 nodes for global features. The features utilized as input to the neural network are a 10x12 spatial representation of the written word, 20 nodes to represent the horizontal spatial coordinate as a function of time and 20 nodes to represent the vertical spatial coordinate as a function of time. The global inputs consisted of:

- total time
- arc length
- average slant
- total length in the horizontal direction
- total length in the vertical direction
- number of horizontal coordinate sign changes
- number of vertical coordinate sign changes
- number of strokes
- total vertical size to total horizontal size ratio

The hidden layer consisted of 50 nodes, and the output layer consisted of 26 nodes, one for each letter in the English alphabet. The size of the output layer can be changed easily if the size of the alphabet used changes.

The neural network was trained with the back-propagation algorithm with bias and momentum terms. The parameters to the training included the training rate, the momentum coefficient, the minimum tolerance and the magnitude of the noise applied. The network was trained against a database of character templates. The same character template database was used with all three character recognition algorithms. The constants involved are:

α	– the learning rate, set to 0.3
μ	– the momentum rate, set to 0.25
τ	– the minimum tolerance, set to 0.15
ϕ	– the noise applied, set to 0.05

4.2.2 DTW, DDTW and MDDTW

In algorithms from the family known as Dynamic Time Warping, input to the algorithm consists of two time series, Q and R of length n and m, respectively:

$$Q=q_{1,}q_{2,}\ldots,q_{n}, \quad R=r_{1,}r_{2,}\ldots,r_{m}$$

The algorithm must output a warping path, W:

$$W=w_{1,}w_{2,}\ldots,w_{k} \qquad max(m,n)\leq k\leq m+n-1$$

where each element w is a pair $w_k=(q_i,r_j)$. The warping path is restricted by the following constraints:

- $w_1=(q_{1,}r_1)$ and $w_k=(q_n,r_m)$
- Adjacent elements in the warping path must contain adjacent elements in Q and R
- Adjacent elements in the warping path must contain monotonically increasing elements from Q and R

The alignment of Q and R is accomplished with a m x n matrix. The value of an element (i,j) in the matrix is set to:

$$val[i,j]=\delta(q_i,r_j)$$

Here, δ is an arbitrary distance function. In the simplest cast, namely the standard Dynamic Time Warping algorithm, the distance function is simply the Euclidean distance between the two points. Clearly, there are exponentially many warping paths from (1,1) to (m,n). The DTW algorithm uses dynamic programming to find the optimal one. The value of optimal warping is defined as:

$$\varepsilon(W)=\sum w_k$$

The recurrence relation necessary for the dynamic programming solution of all Dynamic Time Warping family algorithms is as follows. The optimal cumulative distance for each cell in the m x n matrix

$$\Delta(i,j) = \delta(q_i, r_i) + min\left[\Delta(i-1,j-1), \Delta(i-1,j), \Delta(i,j-1)\right]$$

In the version of Dynamic Time Warping implemented in the presented system, additional constraints were added to the algorithm to favor less warped warpings. The warped-ness of a given warping W can be represented as the distance from the warping closest to the direct path from (1,1) to (m,n). The DTW algorithm can produce warpings which deviate heavily from this direct path, and these should be discouraged. This is most easily accomplished by slope weighting, which seeks to penalize non diagonal steps. This is done by modifying the recurrence relation as follows:

$$\Delta(i,j) = \delta(q_i, r_i) + min\left[\Delta(i-1,j-1), \alpha\Delta(i-1,j), \beta\Delta(i,j-1)\right]$$

where α, β are positive constants greater than 1. In the presented system, where the range of spatial coordinates ranges from -1000 to 1000, both of these penalty constants were set to 100.

Despite its flexible nature, the DTW algorithm can produce very non-intuitive warpings. This can occur when the algorithms attempts to explain warping in the X axis as warping in the Y axis. As a result, the algorithm maps a single point in one sequence to a large number of points in the other. Keogh and Pazzani call these points *singularities*. The problem is that the presented system often interprets singularities as letter boundaries, which creates erroneous output. The authors propose a modification to the DTW algorithm called Derivative Dynamic Time Warping. They redefine the distance function in terms of the local derivatives of the two points being aligned. The details are described in (Keogh and Pazzani). The advantage of DDTW is that it produces much more intuitive and less warped warping paths.

Unfortunately, the DDTW algorithm Keogh and Pazzani propose cannot be used without modification. Their algorithm is only able to process single-dimensional values in the time series. This allows the DDTW algorithm to make the assumption that the time series is a function. However, in a multi-dimensional settings, calculating local derivatives has little meaning. Partial derivatives can be used, but there is no clear way to combine multiple partial derivatives to represent the value of a single distance function.

Instead, the presented system simply uses pairwise difference vectors. The distance function is therefore defined as:

$$\delta(i,j) = d_{Euclidean}(q_i - q_{i-1}, r_j, r_{j-1})$$

This allows the distance function to be expressed as a particular directional derivative in the multi-dimensional time series. This Multi-dimensional Derivative Dynamic Time Warping Algorithm (MDDTW) preserves the advantages of DDTW, but extends it to an arbitrary dimensional setting.

4.2.3 Character String Selection

Since a lexicon was used in the presented system instead of a word-formation grammar, the goal of the system is to find the most likely word in the lexicon given the input. Recall that in the preprocessing stage, a Segmentation Hypothesis Graph was generated. Therefore, the goal of the system is to find the word in the lexicon whose corresponding path in the graph has the lowest cost. However, a given word in the lexicon can have multiple paths corresponding to it. For example, if two nearby cusps represent the boundary between the letters "o" and "g" in the word "jog," then there will be at least two paths in the Segmentation Hypothesis Graph representing the word "jog." Therefore, the goal of the system can finally be defined as the finding the word with the minimal minimum-cost path through the graph. Given a Lexicon Λ, the optimal word λ', is defined as:

$$\lambda' = argmin_{\lambda \in \Lambda} \; argmin_{i, \Psi_i(\lambda) \in \Gamma} \; \varepsilon(\Psi_i(\lambda))$$

To find the optimal word in the lexicon satisfying this property, the lexicon is represented as a trie. A trie is a tree with a node for each proper prefix of a word in the lexicon. Dynamic programming is done on the trie where each node has a corresponding column of the dynamic programming table associated with it. Details of this algorithm can be found in (Chen et. al.)

5. Empirical Results

5.1 Methodology

The lexicon used by the presented system was gathered from the SCOWL word lists. All words up to level 35 were used. All words with apostrophes and non ASCII characters were removed from the lexicon. The remaining lexicon consisted of 38781 words.

Handwritten word data was first collected from 5 participants. Each of these initial participants was asked to write a selection of words from the lexicon. These words contained all letters in the English alphabet, and as such could function as a character template database. The words written by these initial participants were manually segmented and functioned as the character template database. In addition, these 5 participants were asked to write each letter of the English alphabet once. Therefore, the character template database contained between 5 and 10 copies of each letter.

Subsequently, handwritten word data was collected from 20 participants. Each of these secondary participants was asked to write between 10 and 15 words from a prepared list of words from the lexicon. This word list was generated randomly by selecting each word in the lexicon with probability 1/n, where n was chosen so that the word list would have approximately 200 words.

The presented system was tested with three different character recognizers: the Neural Network recognizer, the DTW recognizer and the MDDTW recognizer. The Neural Network recognizer was trained as per the aforementioned algorithm description with the values stated. The DTW and MDDTW recognizers, which do not require training, were tested using the collected character template database. Matching is done against smoothed versions of elements in the character template database.

5.2 Numerical Results

What follows is a table summarizing the results obtained by utilizing the presented system to recognize the collected handwritten word using the collected character template database. The first column of entries in the table measures the accuracy of the system, by specifying the percentage of words written that were recognized correctly. Subsequent columns denote the accuracy within the top 5 and 10 candidates from the lexicon, respectively.

Recognizer	*Accuracy / top 1*	*Accuracy / top 5*	*Accuracy / top 10*
Neural Network	30.91%	36.36%	38.18%
DTW	62.27%	80.91%	88.63%
MDDTW	78.18%	90.45%	91.36%

As can be seen from the table, the best accuracy on the most likely word was achieved by using the MDDTW recognizer. However, the MDDTW recognizer has a much smaller advantage over the DTW recognizer in the accuracy in top 10. The MSTDNN approach presented in Manke and Bodenhausen's paper achieved an 83% accuracy on a 20000 word lexicon. This result is comparable to that of the presented system.

5.3 Discussion

From the results presented in the previous section, it can be concluded that the presented system succeeded in providing a viable solution to the on-line HWR problem.

5.3.1 Failures

However, there is a disappointing apparent plateau that is reached by both the DTW and MDDTW algorithms. This plateau is likely due to the fact that the presented system has an inherent difficulty recognizing words in which letter boundaries do no coincide with high curvature points in the input. This problem is entirely due to the segmentation and not the recognizer. However, a reliable solution was not found or implemented. Possible solutions are discussed in the future work section.

The failure of the Neural Network recognizer can be attributed to the error function that it approximates. The importance of the character recognizer's error function is paramount to the success of the system as a whole. When recognizing individual characters, the importance of the error function is not amplified by the reliance on the recognizer to recognize multiple sections of the input. When the neural network is trained, due to the nature of the problem, only positive instances can be presented to the network for training. These instances consist of a value of 1. for the actual character presented, and a 0. for all other characters. No other values besides 0. and 1. are ever presented to the network. Arguably, partially correct instances could be presented to the network, but such instances are highly subjective and difficult to gather. Because of these training instances, the neural network is able to gage the relative probability of characters in the alphabet, but is not able to learn a proper error function. On the other hand, both the DTW and MDDTW recognizers inherently compute error functions.

The failure of the DTW recognizer to reach accuracy rates as high as the MDDTW recognizer is most likely due to the problem presented in (Keogh and Pazzini). If the DTW finds singularities in the input, the singularities are often interpreted as letter boundaries, and these decisions are often incorrect.

Unfortunately, the increase in accuracy gained by utilizing the DTW family of algorithms for the character recognizer also saw a decrease in speed of running time. On a Compaq Tablet PC, each word was recognized in ~2-3 seconds. This is not fast enough for a truly real-time system. The future work section describes possible solutions to the problem.

5.3.2 Successes

Due to the dynamic programming approach used when selecting candidate strings from the lexicon, the system is flexible with respect to both ambiguity and deformation.

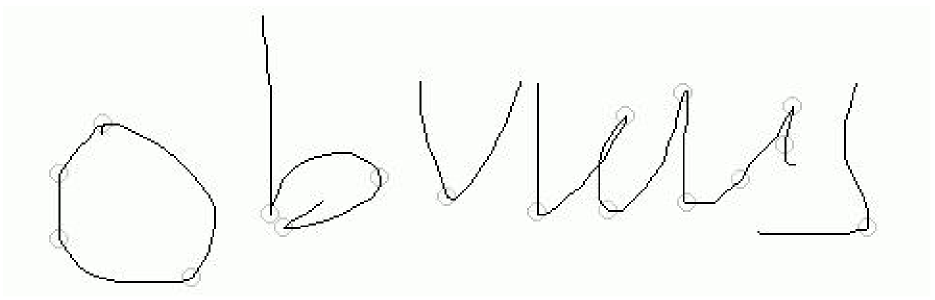

Illustration 3 Deformed example - obvious

Illustration 3 is an example of a highly deformed word that was recognized correctly by the presented system. Since the “o,” “b,” and “v” characters were not nearly as deformed as the rest of the word, the system was able to infer the most likely characters for the rest of the word. Other words beginning with “obv” other than “obvious” include “obverse” and “obviate.” Among these three options, the system chose the clear winner, as the other two words end in the “e” and the cost of selecting the last letter as an “e” was very high compared to an “s.”

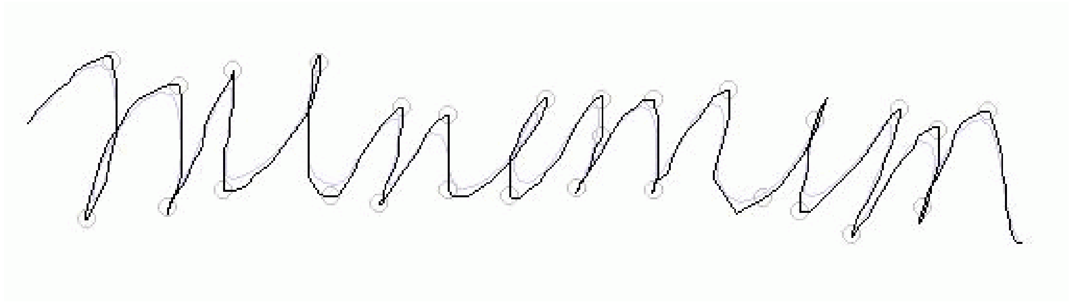

Illustration 4 Ambiguous example - minimum

Illustration 4 is an example of a highly ambiguous word that was recognized correctly by the presented system. The only letters with a low cost that appear anywhere in this word are “m,” “i,” “u,” “w,” “n,” “r,” “v,” “a,” “e,” “c,” “o,” and “l.” There are only 27 words in the lexicon with only these letters. Candidates for the most likely word included such words as “millennium,” “mullion,” and “uranium.” Among these options, “minimum” was the clear winner.

6. Future Work

6.1 Segmentation

The problem discussed in the discussion section with respect to smooth letter boundaries has a number of possible solutions. The most obvious solution is to increase the number of probable segmentation points, since the current segmentation algorithm already over-segments the input. In the limit, this would hypothesize a segmentation point for every point of the input. This has two adverse effects. Firstly, the resulting system would be much slower, since the DTW character recognizes run in O($\mathbf{c}^2$) where **c** is the number of probable segmentation points. Secondly, this method defeats the purpose of segmenting the input-- recall that the segmentation of the input was done in order to narrow the number of possible segmentation points.

Another possible solution is to hypothesize segmentation points that divide long sections of the input with no high curvature points. While this does in fact cover a higher number of actual letter boundaries, the optimal letter boundary often does not exactly divide long segments on the input.

Finally, a likely solution to the problem is to hypothesize a large number of probably segmentation points, but assign variable probabilities to the points themselves. This addresses the problem of extreme over-segmentation and long sections with no high curvature points, but does not address the problem of running time.

6.2 Neuro-Dynamic Programming

To solve the problem of the speed of running time of the presented system, there are two possible solutions. Firstly, the Neural Network recognizer, which is much faster than a DTW approach, can be improved to approach the accuracy of the MDDTW recognizer. However, this approach is likely to yield few results.

Secondly, direct speed enhancing methods can be used to optimize the MDDTW recognizer. More specifically, the dynamic programming approach can be modified to use heuristics to prune large portions of the dynamic programming table. These heuristics can take the form of neural networks trained using TD learning. Such methods are described in (Bertsekas and Tsitsiklis).

6.3 Language Model

Due to the use of a trie to represent the lexicon, large lexicons do not pose a problem with respect to the running time of the presented system. For example, recognition with a 96,000 word lexicon was attempted and the difference in running time was negligible. However, the use of a lexicon does not allow for the recognition of unseen words (e.g. slang words which follow a proper word formation grammar, but are not in the lexicon). The solution to such a problem is to use a more flexible language model, most likely in the form of a word formation grammar. This is the subject of future research and has not been explored.

7. Conclusion

A system to recognizer on-line handwritten words has been presented. It has been shown that a MDDTW character recognizer out-performs both DTW and Neural Network recognizers. The resulting system can achieve up to 78.18% accuracy and a 91.36% top 10 accuracy. The presented system is able to handle both highly deformed and highly ambiguous words. In the future, I hope to extend the system to solve the running time problem and utilize a more flexible language model.

8. References

Bellegarda, Jerome R. et al. *Supervised Hidden Markov Modeling for On-line Handwriting Recognition.* 1994. Yorktown Heights, New York.

Bertsekas, Dimitri and Tsitsiklis, John. *Neuro-Dynamic Programming.* 1996. Athena Scientific.

Breuel, Thomas M. *A System for the Off-Line Recognition of Handwritten Text.* 1994. Martigny, Switzerland.

Breuel, Thomas M. *Segmentation of Handprinted Letter Strings using a Dynamic Programming Algorithm.* 2001. Palo Alto, CA.

Chen, David Y. et al. *An Efficient Algorithm for Matching a Lexicon with a Segmentation Graph*. 1999. San Jose, CA

Chetverikov, Dmitry and Szabó, Zsolt. *Detection of High Curvature Points in Planar Curves*. 1999. URL: http://visual.ipan.sztaki.hu/corner/index.html

Crochemore, Maxime et al. *A Sub-quadratic Sequence Alignment Algorithm for Unrestricted Cost Matrices.* 2002. Haifa University.

Keogh, Eamonn J. and Pazzani, Michael J. *Derivative Dynamic Time Warping*. 2001(?). Irvine, California.

Manke, Stefan and Bodenhausen, Ulrich. *A Connectionist Recognizer for On-line Cursive Handwriting Recognition*. 1994. Karlsruhe, Germany.

Oh, Jong. *An On-Line Handwriting Recognizer with Fisher Matching, Hypotheses Propagation Network and Context Constraint Models*. 2001. New York University.

Tappert, Charles C. et al. *The State of the Art in On-line Handwriting Recognition.* 1990.