Continuous Handwritten Script Recognition 12

Volkmar Frinken and Horst Bunke

Contents

V. Frinken (✉)
Computer Vision Center, Autonomous University of Barcelona, Bellaterra, Spain
e-mail: vfrinken@cvc.uab.es

H. Bunke
Research Group on Computer Vision and Artificial Intelligence (FKI), Institute of Computer Science and Applied Mathematics, University of Bern, Bern, Switzerland
e-mail: bunke@iam.unibe.ch

D. Doermann, K. Tombre (eds.), *Handbook of Document Image Processing and Recognition*, DOI 10.1007/978-0-85729-859-1_12,
© Springer-Verlag London 2014

Abstract

The transcription of written text images is one of the most challenging tasks in document analysis since it has to cope with the variability and ambiguity encountered in handwritten data. Only in a very restricted setting, as encountered in postal addresses or bank checks, transcription works well enough for commercial applications. In the case of unconstrained modern handwritten text, recent advances have pushed the field towards becoming interesting for practical applications. For historic data, however, recognition accuracies are still far too low for automatic systems. Instead, recent efforts aim at interactive solutions in which the computer merely assists an expert creating a transcription. In this chapter, an overview of the field is given and the steps along the processing chain from the text line image to the final output are explained, starting with image normalization and feature representation. Two recognition approaches, based on hidden Markov models and neural networks, are introduced in more detail. Finally, databases and software toolkits are presented, and hints to further material are provided.

Keywords

BLSTM neural networks • Feature extraction • Handwriting recognition • Hidden Markov models • Sequential approaches

Introduction

Continuous handwritten script recognition (HWR) is the task of transforming handwritten text into a machine-readable format, for example, a string of ASCII characters. The way the text is entered into the computer can either be *online* or *off-line*. In the case of *online* recognition, information about the pen is sampled at well-defined points in time while the text is written. Therefore, the input is a sequence of values, such as the x- and y-coordinates of the pen tip. By contrast, *off-line* recognition is the task of reading text from an image, such as a scanned document. Early approaches tried to treat handwritten text one character or word at a time [6, 52]. However, a main problem became evident with this approach. While machine-printed text can easily be segmented into single words or characters to be recognized individually, this does not hold true for cursive handwritten text. In 1973, Robert Sayre summarized that knowledge in what is known as the Sayre's paradox: *To recognize a letter, one must know where it starts and where it ends, to isolate a letter, one must recognize it first* [50]. Nowadays it has become a widely accepted position in *off-line* recognition that it is necessary to treat the entire text line as one sequence. This way, one does not need to separate the letter extraction from letter recognition task. In realizing the sequential nature of continuous handwritten script, its recognition has more similarities to speech recognition than machine-printed optical character recognition.

This chapter focuses on the off-line recognition of Latin scripts, but it has been shown that the same, or similar, approaches are also valid for numerous other scripts, such as Arabic or Devanagari [36, 53]. ▸Chapter 13 (Middle Eastern Character Recognition) of this book takes a detailed look at recognition techniques for Middle Eastern scripts. The rest of this chapter is structured as follows. In section "Overview," an overview of the field of continuous recognition is given. Preprocessing steps specific to *off-line* handwriting recognition, such as the normalization of the writing slant, are reviewed in section – "Preprocessing" while common features used to convert a text line into a sequence of features vectors are discussed in section "Overview of Common Features." Recognition techniques are presented in section "Recognition Methods" with a focus on Hidden Markov models and neural networks. Common databases, software toolkits, and remarks about evaluation methodology are given in section "Datasets, Software, and Evaluation Methodology." Finally, the chapter closes with a summary of the key issues and promising approaches in section "Conclusion."

Overview

History and Importance of the Problem

Handwriting evolved as means of communication between humans and became a common way of exchanging information. The automatic recognition of handwritten text offers a natural way for human-computer interaction. Instead of requiring a person to adopt to technology, or to require a person to extract valuable information from a given handwritten text and type it manually into a computer, automatic handwriting recognition tries to directly decode the written information.

Depending on the context in which such a system is used, two fields with different prerequisites and requirements can be distinguished. In one field, handwriting recognition serves as a preprocessing step for further interpretation of the content. Later in this book, in ▸Part D (Processing of Non-textual Information) and in ▸Part E (Applications), several key applications based on handwriting recognition are presented. Examples of the case where handwriting recognition is used as a preprocessing step are address reading on pieces of mail (see ▸Chap. 21 (Document Analysis in Postal Applications and Check Processing)), bank check processing, or analyzing handwritten letters in order to classify them. Typically, the vocabulary is limited, and robust post-processing methods might be able to deal with a set of recognition hypotheses instead of just one final recognition result. Sometimes, single-word recognition (see ▸Chap. 11 (Handprinted Character and Word Recognition)) approaches are sufficient for that. The other case is the pure transcription of handwritten text, for example, when digitizing historic documents (see ▸Chap. 23 (Analysis of Documents Born Digital)) or personal notes. In this case, a restricted vocabulary cannot be expected, and only one recognition output is sought after.

Well-working commercial applications exist for bank check processing and address reading. As far as transcription is concerned, recognition engines for tablet computers and mobile devices exist that yield good results. For historic data, however, the variability of existing writing styles limits the applications in that domain to interactive systems that aid in generating a transcription.

Evolution of the Problem

Interest in automatic recognition of handwritten text started in the middle of the last century when banks began to recognize the potential of automatic check processing [49]. Soon, automatic postal address processing followed [2]. These two tasks, as well as form reading, have the advantage of strict constraints. The underlying vocabulary is limited, and, due to some redundancy implicitly included in the task, error detection is possible. For example, on a bank check, the legal amount written in words has to match the one written in digits. Similarly, in a postal address, the name of the city has to match the zip code, and the street and house number have to exist.

With the advent of more powerful recognition systems, more and more constraints were lifted and the recognition of unconstrained continuous handwritten script became a focus of attention. State-of-the-art recognition systems are nowadays able to perform, e.g., at an accuracy of 75 % on writer-independent, unconstrained English texts [14]. However, this figure is still far away from what is needed in many practical applications. Therefore, after several decades of intensive research, serious efforts are still undertaken to improve the performance of current recognition technology.

Traditionally, humans were involved in the transcription process only towards the end of the recognition process, in order to correct errors committed by the system. Recently, more elaborate approaches have been proposed where a much tighter interaction between the recognition system and a human corrector is achieved. In this way, the human effort for error correction can be significantly reduced [58].

Applications

The recognition of handwritten text can make human-machine interaction more natural. Automatically recognizing documents written in a way to be legible for humans and not specifically for machines reduces the overhead of digitalizing data. Hence, applications of continuous handwritten script recognition are present, wherever written information has to find a way into a computer system. In the previous paragraphs, bank check reading [22], address reading [7], and form processing [34] have been mentioned. In addition, digitalization and classification of text in handwritten letters and forms are becoming more and more automatized. Companies such as ITESOFT®, VisionObjects®, and A2iA® offer such solutions to businesses for efficient communications with customers.

In the context of preserving the world's cultural heritage, the digitalization of historic documents emerges as an important task. Huge amounts of handwritten data are stored in libraries, either completely unavailable to the public or only in the form of scanned images [44]. Obviously, the usefulness of collecting those images increases dramatically with the possibility of searching, browsing, and reading the transcribed content.

Main Difficulties

Similarly to other applications dealing with data not specifically designed for automatic processing, handwriting recognition poses several challenges. Extreme variability can be encountered in the writing styles even within a group of writers with a similar cultural and educational background. Samples of English text lines written by Swiss students can be seen in Fig. 12.1a. In addition to that, handwriting is taught differently across different countries and generations. In contrast to speech, which humans use frequently in daily communication, writing text is not done as frequently and often only for personal use. As a consequence, handwriting styles can become sloppy to a degree where even the writer has difficulties reading his or her own handwriting after some time.

But not only in those extreme cases, reading handwritten text requires understanding the context. Latin cursive script is too ambiguous to be recognized one isolated word at a time. The example in Fig. 12.1b shows that the letters "cl" are often written very similarly to the letter "d." Hence, the presented words could be "day" or "clay." Obviously, as the meaning of the two possible transcriptions is so different, a human instantly recognizes the words as "day" if they are preceded by "Have a nice," for example. However, when the surrounding text is about the history of water jugs, the images are recognized as "clay."

Furthermore, the segmentation of text lines into individual words is not a straightforward task. It is easier than segmenting a word into characters, since interword gaps are usually larger than intra-word gaps. Nevertheless, many exceptions exist. In Fig. 12.1c, the gap in the first word is larger than the gap between the two words at the end of the text line.

In a general text, every possible sequence of characters may occur, especially when proper names are used or words of different languages. Nevertheless, allowing any possible character sequence as an output is not desirable since it may impede the robustness of the system. As a consequence, one can restrict the vocabulary of words that can be recognized, at the cost of not being able to recognize out-of-vocabulary words at all. The benefit of this approach is an increased average recognition rate, since most words in a single text originate from a limited vocabulary.

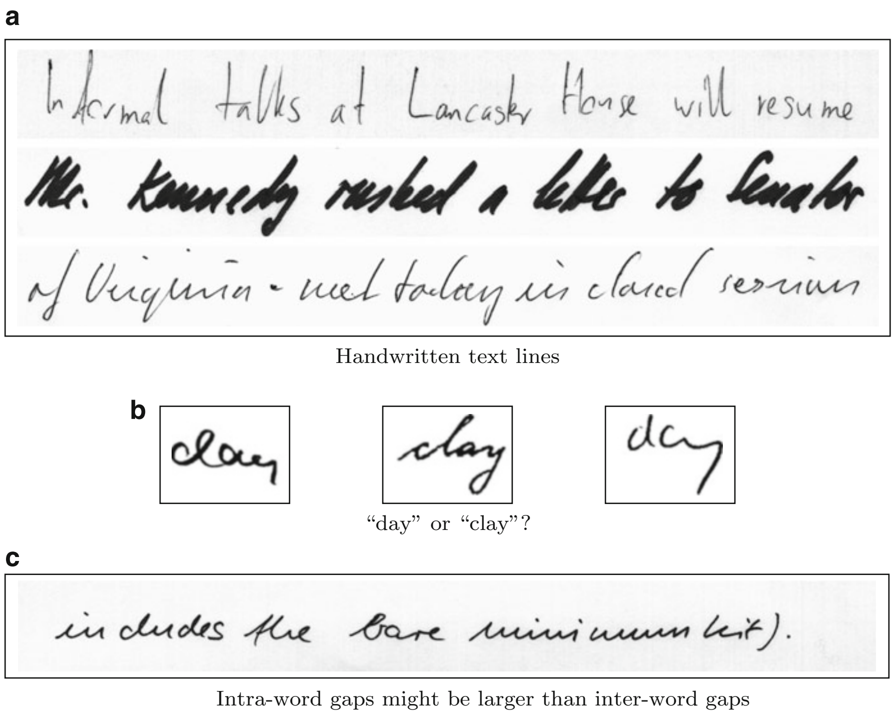


Fig. 12.1 (**a**) Examples of different writing styles, (**b**) ambiguous words, and (**c**) segmentation challenges

Summary of the State of the Art

The state of the art in continuous handwriting recognition has led to a number of well-working applications for small vocabulary or single-writer tasks. But it was also realized that general, large vocabulary unconstrained handwriting recognition continues to be a hard task for which only research prototypes exist. Typically, high-performing recognition systems try to make use of as much external information as possible, such as specific dictionaries or language models in the form of word probabilities. Also, research has started to focus on particular tasks like interactive transcription systems or cases where training data is limited.

As far as recognition technology is concerned, nearly all modern approaches are segmentation-free. In such a setup, a text line is transformed into a sequence of feature vectors. Afterwards, methods specifically designed to process sequences are used to recognize handwritten text. Many of those methods have their origins in speech recognition.

Preprocessing

Overview

The transformation of text line images into sequences of features vectors requires several preprocessing steps. At first, the area of the scanned page containing the handwritten text needs to be localized and cleaned of artifacts like horizontal ruling on the page. Afterwards, individual text lines are extracted and normalized. Up to text line skew detection and normalization, the same methods as the ones reviewed in ▸Chap. 3 (The Evolution of Document Image Analysis) of this book can be applied. Nevertheless, succeeding normalization steps are needed to cope with the immense differences in writing styles. A sample of typical normalization steps are given in the following subsections.

Normalization

A property specific to handwritten text lines in the angle by which the writing is inclined is the so-called writing slant. An example is given in Fig. 12.2. The slant angle can be detected in several ways (see below). Once it has been estimated, a sheer operation is used to normalize the slant angle without affecting the slope correction. Afterwards, the height of the text is normalized. The major challenge in this step is to treat ascenders, descenders, and the core region of the text differently.

Uniform Slant Correction

In [37], the contour of the script is computed by marking all pixels whose horizontal neighbors have a different brightness with the difference exceeding a certain threshold. Note that only horizontal neighbors are considered to put an emphasis on vertical strokes. Next, the pixels are approximated by straight lines. Finally, a weighted histogram of the inclinations of all lines is computed using the squared lines' lengths as weights. The angle where the maximum of the histogram occurs is then considered to be the writing slant.

Nonuniform Slant Correction

In [52] an edge detection operator is used to find the edges of the writing. Then the average angle of nearly vertical strokes is computed. In [55], a non-uniform approach is proposed that estimates the local slant angles by solving a global optimization problem on the sequence of local slant angles using dynamic programming. In [14] a neural network is used to correct the local slant.

Text Height Normalization

The height normalization is done by horizontally segmenting text into three writing zones: the core zone, the ascender zone, and the descender zone. The core zone is

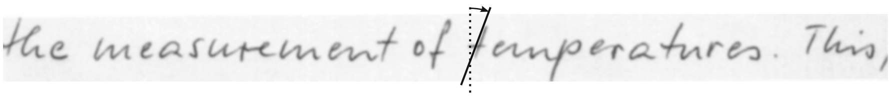


Fig. 12.2 Illustration of writing slant

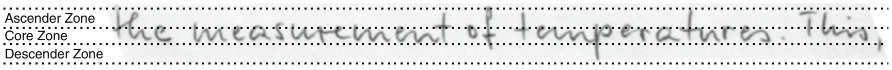


Fig. 12.3 The three different writing zones

the region in which all lowercase letters are written that do not have ascending or descending strokes. It is known as "x-height," and examples of characters occupying only this zone are *a*, *c*, *m*, or *x* (see Fig. 12.3). The area above the core zone, which is filled by capital letters or ascenders, is the ascender zone. The zone below the core zone in which descending strokes are written is called descender zone.

By analyzing the horizontal histogram of the black pixels, the three different zones can be found since the number of black pixels in the ascender and descender zone is substantially lower than in the core zone. More work on height normalization with histogram analysis is described in [6]. A learning-based method is proposed in [51]. Similarly, in [14], the borders between the different zones are estimated using a neural network. Once the three different zones are detected, they can be normalized to uniform height in a nonlinear way.

Text Width Normalization

In a next step, the writing width can also be normalized. Ideally, the text line image should be stretched or compressed horizontally so that each character has a predefined width. However, this is not possible since the actual text needs to be known for this. Instead, the frequency of strokes crossing the line that runs horizontally through the middle of the image is counted. This quantity may then be used to adjust the image's width by setting the average distance between these crossings to a predefined value.

Overview of Common Features

Features for continuous handwritten text can be separated into holistic features, extracted from an entire word for segmentation-based approaches, or sequential features for segmentation-free approaches. While ▸Chap. 11 (Handprinted Character and Word Recognition) of this book covers features for single-word recognition, this chapter focuses on the latter ones. Feature extraction processes are discussed that transform a text line into a sequence of vectors. These processes typically use a sliding window. In such an approach, a small window with a width of *n* pixels is swept across the text line image in steps of *m* pixels, usually in the direction of writing, i.e., left to right in Latin scripts. At each position, a feature vector is extracted from the window and the feature vectors are then concatenated to a sequence. Thus, the two-dimensional information of the text image is transformed into a one-dimensional sequence.

In short, the goal of feature extraction is to represent arbitrarily complex shapes by a fixed set of numerical attributes as well as possible. A good feature set should contain discriminative information but should also be easy to compute. Here, we present popular pixel-based features that directly use the grayscale values; statistical features that focus on properties of the black pixel distribution; gradient-and filter-based features; and shape-based features that try to capture significant information of the shape around the sliding window.

Pixel-Based Features

The most basic features than can be extracted from a text line image are the raw pixel intensities of the window, without any further processing. This can basically be done in two ways. The first way is to read the gray-level value of one pixel after another, e.g., top to bottom, one column after another, as depicted in Fig. 12.4a. This results in a one-dimensional sequence of the image that is easy to process. However, the two-dimensional information of the image is lost or must be transmitted separately. In [24], this approach is used in combination with a recurrent neural network, specifically designed for such sequences. At each time step t, not only the state of the hidden nodes at time $t-1$ but also the state at time $t-H$ can be accessed, where H is the height of the image. This way, when a pixel is processed, the context to the top and to the left of that pixel is known. The other approach to use the gray-level values of an image is via a sliding window of width 1, i.e., one column of the image, which is directly interpreted as a column vector (see Fig. 12.4b). This approach is followed in [57].

Using directly the pixels, especially as a column vector, does not tend to be extremely robust when it comes to different writing styles. In fact, a vertical displacement of the same word by just one pixel changes the feature vector substantially. To counter this effect, the sliding window can be divided into several partitions, and only the average value of each partition is considered, as depicted in Fig. 12.4b. Those partitions can be the three different writing zones [32] or just arbitrary blocks [4].

To cope even better with variations in text size and vertical positions, a quite successful set of features has been proposed in [60]. The sliding window is subdivided into a regular 4×4 grid, and its height is adjusted to only consider the area containing black pixels, as shown in Fig. 12.4c.

Statistical Features

From a mathematical point of view, one can focus one the distribution of the black pixels within the sliding window and use statistical measures of that distribution as feature values. A common approach to do this is to compute the distribution's moments

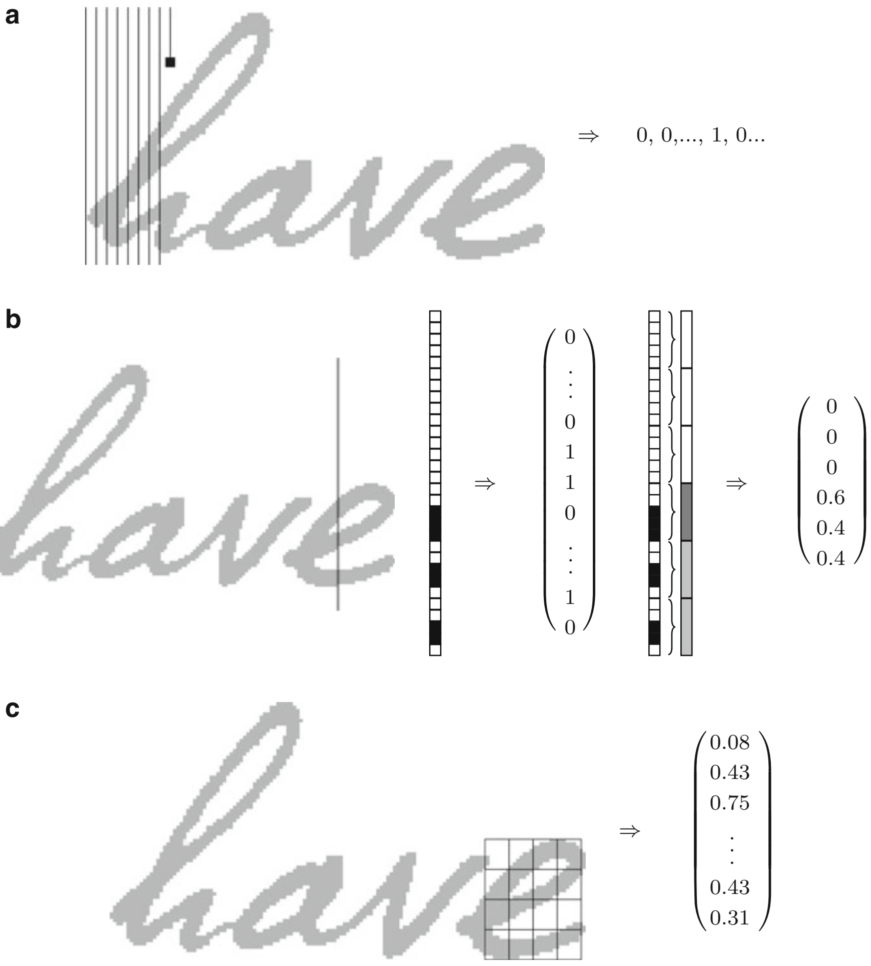
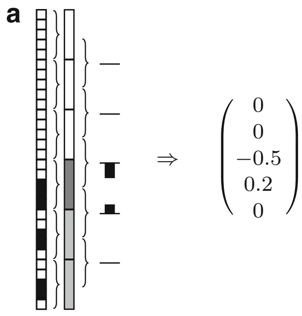


Fig. 12.4 Examples of pixel-based features. (**a**) A vertical scan line creates a one-dimensional feature sequence (**b**) Gray-level values from a small sliding window (**c**) Averages from a height-adjusted regular grid

$$m_W^{p_x,p_y} = \sum_{(x,y)\in W} D(x,y) \cdot x^{p_x} y^{p_y} \; ,$$

where $m_W^{p_x,p_y}$ is the (p_x, p_y)th moment, $p_x, p_y = 0, 1, \ldots,$ of the pixels' distribution in window W, $(x, y) \in W$ are the individual pixels, and the darkness value $D(x, y)$ returns 1 if the pixel is black and 0 otherwise [32, 38].



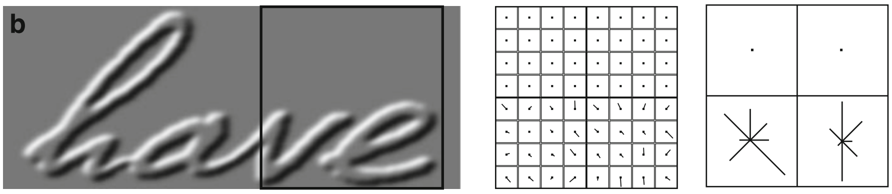


Fig. 12.5 The derivative of the zones act as features. (**a**) Vertical derivatives between zones. (**b**) Histogram of quantized gradients of 4 zones, subdivided into 16 cells each

Gradient and Filter Features

For the following class of features, assume that the image is not binarized but given as a continuous intensity function $I(x, y) \in [0, 1]$. In fact, most approaches that use gradient features blur the image using a Gaussian function prior to the feature extraction process. Using the intensity function $I(x, y)$, the horizontal and vertical gradient can be computed as its partial derivatives along the axes. Usually, however, the gradient is not taken between adjacent pixels but rather between larger zones as the difference of the average intensities in these zones. An example is given in Fig. 12.5a.

Consequently, the gradient within a rectangular cell can also be computed. The horizontal gradient is the difference between the average intensity on the left half and the right half of the cell. Similarly, the vertical gradient is given by the difference between the top and the bottom part of that cell. Hence, when dividing the sliding window into rectangular cells, the set of vertical and horizontal gradients from the different cell constitute a set of gradient features [4, 57].

Frequently, not the single gradients themselves are used, but a histogram of gradients. In this approach, the horizontal and vertical gradients (G_x, G_y) from each cell are treated as a vector and represented by its length and direction. The direction is then quantized and weighted according to the length of the vector. Finally, the gradient vectors from different parts of the sliding window are accumulated in different histograms of gradients. An example of the process is depicted in Fig. 12.5b. On the left hand side, the magnitude of the vertical gradients of the sample image is represented by different gray values (1 is white, -1 is black), superimposed with the sliding window. To the right, the gradient vectors for each cell are indicated and finally combined to four different histograms. Notable works following a comparable approach are [47, 56].

Mathematically speaking, the computation of a gradient between adjacent areas can be described as a convolution of the image with a simple, Haar-like filter. Instead of such a rather simple filter, more sophisticated ones are also applicable. Gabor filters, e.g., are sensitive towards a sinusoidal stimulus of a specific direction and frequency. Hence, several Gabor filters with different directions and frequencies can be applied to areas of the sliding window to compute features [9]. There is evidence that the human eye processes information in a similar way [59].

Shape-Based Features

Shape-based features express attributes about the shape of the writing within the sliding window as numerical values. Frequently used, yet simple attributes are the frequency of vertical black-white transitions as well as the position and inclination of the contour [20, 38]. Contour features of this type are displayed in Fig. 12.6a.

A different way to represent a shape is by using run-length information, which is the number of similar, consecutive pixels along a given line. In that regard, the position of the contour can also be seen as a run-length, since it reflects the number of white pixels from the top, resp. the bottom, of the image until the first black pixel is reached. Likewise, the number of black pixels along a line can also be used. Figure 12.6b shows such an approach. For each pixel, its horizontal run-length is given. The sum of the run-lengths in different zones of the sliding window can then be used as features. Obviously, the direction of the run-length does not necessarily need to be horizontal, but can be any direction [19].

Still, the problem exists of how to represent complex, two-dimensional shape information using a vector. Upon a closer look, however, the absolute shape is not as important as its similarity to known shapes, such as characters. In the context of continuous text recognition, it was proposed to match a set of prototype shapes with the content of different sliding windows, each having the same size as the corresponding prototype shape (see Fig. 12.6c). Afterwards, the distances to the prototype shapes is used as feature values. In [16], the prototype shapes and the writing are represented as a skeleton graph. The distance between the sliding window subgraph and the prototype graph is computed via a graph edit distance.

Feature Transformation

Often recognition systems do not make use of just one class of features but combine several types. Also, feature transformation methods have shown to be beneficial, such as using the Loeve-Karhunen transformation [42], a Kernel-principal

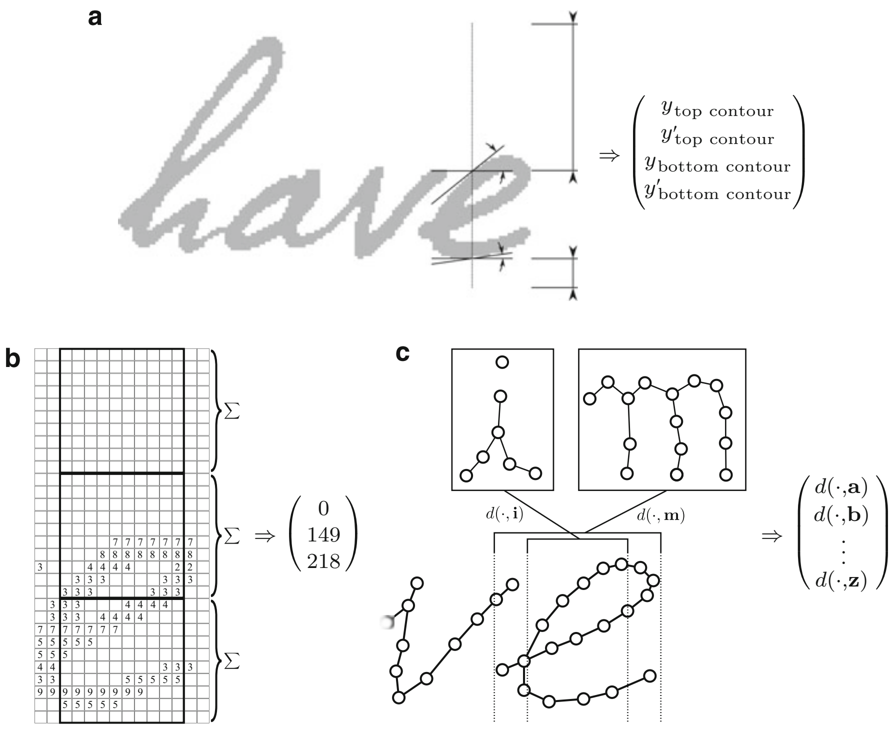

a
$\Rightarrow \begin{pmatrix} y_{\text{top contour}} \\ y'_{\text{top contour}} \\ y_{\text{bottom contour}} \\ y'_{\text{bottom contour}} \end{pmatrix}$
b
Σ
Σ ⇒ $\begin{pmatrix} 0 \\ 149 \\ 218 \end{pmatrix}$
Σ
c
$d(\cdot, \mathbf{i})$
$d(\cdot, \mathbf{m})$
$\Rightarrow \begin{pmatrix} d(\cdot,\mathbf{a}) \\ d(\cdot,\mathbf{b}) \\ \vdots \\ d(\cdot,\mathbf{z}) \end{pmatrix}$

Fig. 12.6 Some shape-based features. (**a**) The position y and inclination y' of the contour. (**b**) Horizontal run-length features. (**c**) Graph edit distance to prototypes

component analysis (PCA) [15], neural network-based non-linear PCA [61], or independent component analysis [15, 61].

Summary of the Features

In List 12.1, a list of the feature types are given, together with a list of their advantages and disadvantages. Table 12.1 contains an overview of key references. Note that some features have only been proposed for word recognition or keyword spotting. However, it is obvious that they can be used for continuous text line recognition as well.

For a successful recognition system, selected features should not be too sensitive to noise and have discriminative power, i.e., different characters should produce different feature vector sequences. Selecting too few features might not be able to do that while too many features might lead to unstable or overtrained systems. Selecting a well-performing set of features can be difficult and may depend upon the script, the writers, the specific task, and even the amount of the available training data.

Pixel-based features:
+ Easy implementation
+ Script and style independent
+ Fast
− Careful height normalization required, sensitive against vertical variations
− Puts more burden on the recognition system

Gradient-based features:
+ The best performing features in the literature are gradient-based.
− Not independent of stroke thickness.
− The areas in which the gradients are computed are fixed, hence not extremely robust against vertical variations of the text.

Shape-based features:
+ Complex shape information is extracted.
+ Each feature carries meaningful information.
− Potentially slow to compute.
− Shape representation important.
− Not script independent.

Feature transformations:
+ Discriminative information can be enhanced.
+ Reduction of irrelevant information.
− A further step that adds complexity and a possible bias.
− Some feature transformations require learning, which requires more training data.

List 12.1 A list of advantages and disadvantages for different types of features for continuous handwriting recognition

Table 12.1 An overview of features for converting an image of handwritten text into a sequence of vectors

Feature description	References
Pixel value of a column as binary vector	[57]
1D sequence of pixels, top to bottom, left to right	[24]
Number of black/white transitions	[20, 38]
Partial derivatives along x and y axis	[4, 14, 57]
Histogram of derivatives in subregions	[56]
Histogram of gradients in subregions	[47]
Position and slope of the contour	[38]
Graph edit distances to prototype letters	[16]
Correlation of intensities between subregions	[4]
Post-processing of medium and low-level features	[15, 42, 59, 61]

Recognition Methods

Overview

In this section, methods are discussed to transform a sequence of feature vectors, extracted from a handwritten text line, into a character or word sequence.

An obvious approach is to segment the text line into individual words and individually classify the segments using one of the methods described in ▸Chap. 11 (Handprinted Character and Word Recognition). However, similarly to classifying a word into individual letters, ambiguities might arise when segmenting a text line into words. Hence, the classification results obtained on the single segments have to undergo suitable post-processing procedures, using, e.g., dynamic programming. Further details on segmentation can be found in ▸Chap. 11 (Handprinted Character and Word Recognition) and [31].

A handwritten text line can also be considered as a horizontal signal. From that point of view, similarities to speech recognition are apparent. Consequently, most, if not all, sequential approaches to continuous handwriting recognition have their roots in speech recognition. The most promising ones include statistical methods such as hidden Markov models. Due to the outstanding role these models play in this field, a detailed review is given in section "Hidden Markov Models." A well-performing neural network architecture that was recently developed and since then has been consistently outperforming traditional approaches is the so-called bidirectional long short-term memory neural network. Its emergence significantly changed the field and further improvements are to be expected. This network is presented in section "BLSTM Neural Network with CTC Token Passing Algorithm."

Other sequential recognition techniques [11] might theoretically also be applied to this task. Although Conditional Random Fields and their derivatives have occasionally been proposed for single-word recognition [12], they play practically no major role in continuous handwriting recognition.

Hidden Markov Models

General Definition

A hidden Markov model (HMM) is a model of a stochastic process which generates a sequence of output elements or emissions over a period of time. The system may have several internal and unobservable states which are assumed to resemble a first-order Markov chain, i.e., a *discrete-time* random process, which is *stationary*, *causal*, and *simple*. Discrete time means that at each time step, the system is in exactly one of several distinct states. Stationary means that the absolute time does not have an influence on the state the system is in. Causal means that only states the system has been in the past, but no future events, influence the next state. Finally, simple means that only the current state influences the next state. In short, the probability $p(S(t) = s_i)$ of the system to be in state $S = s_i$ at time t does only depend upon the state at time $t-1$ and can therefore be written in the form of a state transition probability matrix $A = (a_{ij}) \in [0;1]^{n \times n}$ with

$$a_{ij} = P\left(S(t) = s_j | S(t-1) = s_i\right) .$$

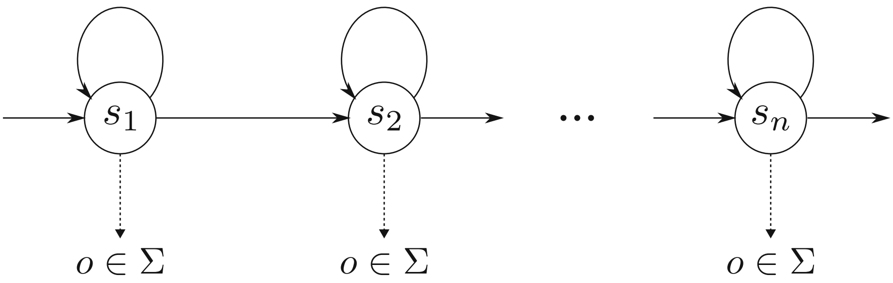


Fig. 12.7 Linear HMM state transition topology for handwriting recognition

Formally, an HMM λ is a tuple $\lambda = (S, A, \pi, B)$, where $S = \{s_1, s_2, \ldots, s_n\}$ is a set of states, $A = (a_{ij})$ the state transition probability matrix, $\pi \in [0; 1]^n$ the start probability vector, and $B = \{b_1, b_2, \ldots, b_n\}$ a set of state-dependent emission probabilities. The system starts in one of the states, according to the start probabilities π. At each time step it changes its internal state, according to the state transition probability matrix A and emits an element according to the (state-dependent) emission probability function.

Although no restrictions are given for possible state transitions according to this definition, certain topologies are frequently used. The ergodic model is the most general case, allowing a transition from every state to every other. If A is an upper triangular matrix, the states form an ordering, since from one state, only transitions into following states are possible; hence, it is called left-right model. Common for handwriting recognition are even more restricted models, especially the Bakis model in which only the next two states can be changed into and the linear model that allows transitions into the next state only. A linear hidden Markov model is shown in Fig. 12.7.

Training and Decoding Algorithms

The popularity of HMMs and their applicability to a diverse class of problems can be accounted to the fact that efficient training algorithms exist to automatically estimate the parameters of an HMM and to infer the most likely sequence of hidden states given an observed sequence.

The most common training method is the *Baum-Welch* algorithm, introduced in [3]. It considers the probability $P(O|\lambda)$ that a given sequence O is produced by the hidden Markov model λ and computes a new model λ' which is more (or equally) likely to have produced the given sequence, so that $P(O|\lambda') \geq P(O|\lambda)$. It is an iterative optimization algorithm which converges towards a local maximum, but it cannot guarantee to find the globally optimal parameters. A detailed description of the Baum-Welch algorithm is given in [45].

If the emission probabilities are modeled by Gaussian distributions, every sequence of hidden states could have produced the results. Therefore, the most likely sequence

$$s_{\text{best}} = \underset{s \in S*}{\arg\max}\, P(s|O, \lambda)$$

out of a set of sequences S^* is sought after. S^* does not contain every possible character sequence but only those that form words contained in the language model. Using the Bayes rule [13] and dropping the constant factor $P(O|\lambda)$, which does not influence the result, leads to

$$s_{\text{best}} = \underset{s \in S*}{\arg\max}\, P(s|O, \lambda) = \underset{s \in S*}{\arg\max}\, \frac{P(O, s|\lambda)}{P(O|\lambda)} = \underset{s \in S*}{\arg\max}\, P(O, s|\lambda)\ . \quad (12.1)$$

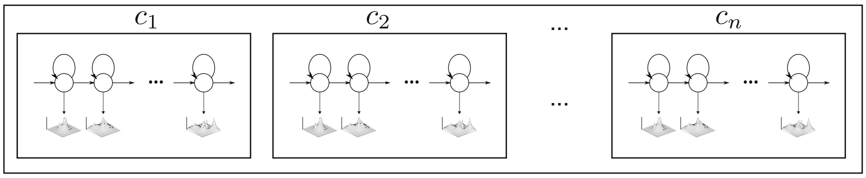
In the *Viterbi* algorithm [63], this equation is used to compute the most likely sequence. In the first step, the starting probabilities are computed. In each of the following iteration over all observed sequence elements, a state's probability can be inferred by exploiting the fact that next to the observed emission, only the system's predecessor state is important. The algorithm stores for all steps and each state the most likely preceding state and after the sequence elements have been processed, starting from the end, the most likely states can be efficiently inferred in a dynamic programming approach.

Hidden Markov Models for Continuous Handwriting Recognition

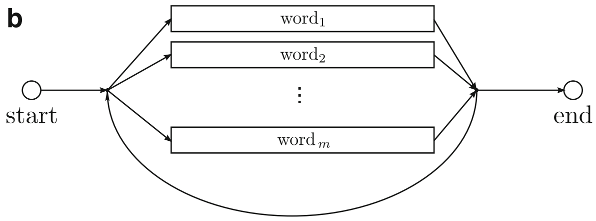
With their ability to compute a likely sequence of hidden states given a sequence of elements, hidden Markov models are well suited for handwriting recognition. For continuous handwriting recognition, a text line is represented as a sequence of elements. It is assumed that the writing process that created the letters and finally the sequence can be modeled by a hidden Markov model. In this setup, the emissions of the system are the feature vectors of the sequence. The internal states correspond to words, characters, or even smaller units. Character HMMs have advantages and disadvantages over word HMMs. On the one hand, small, common words are often written in one stroke and do not resemble the simple concatenation of the individual letters very well. On the other hand, in each text, the number of training elements is much higher when only characters are to be trained as opposed to whole words. In most cases, the advantages gained by the increase in training data when using character models outweigh the disadvantage of having to model, in one HMM, very diverse ways a character can be written.

In the training phase, text line HMMs are assembled using character HMMs as well as HMMs representing punctuation marks and spaces. To decode more than just a single word, a modified version of the Viterbi algorithm, called *Token Passing* Model, is applied [66].

Usually a separate hidden Markov model is constructed for each text unit. Hence, character HMMs or whole-word HMMs are used frequently. Afterwards, these are in turn aggregated into a larger network. In each HMM, the states follow a clear ordering, mostly in a linear or a Bakis topology. The number of states in such HMMs depends upon the size and complexity of the object and has to be validated or set according to prior knowledge. In Fig. 12.8 an architecture of a character-based system can be seen. First, character HMMs are connected to form word HMMs

a


A word HMM assembled by concatenating character HMMs.

b


A complete HMM network to recognize text.

Fig. 12.8 (**a**) A character HMM consisting of several states can be seen. These models are connected to form words. (**b**) All words are connected in a network for the recognition

according to a lexicon (Fig. 12.8a). Under such an approach, the recognition is restricted to those words that occur in the lexicon. Next, given the word models, the end of each word is connected to the beginning of each word (Fig. 12.8b). As a result, a model for word sequences is obtained.

At each time step, an output is generated according to the current state's output probability distribution. This implies that it is necessary to compute for a given element x of the sequence the emission probability $p(x|q)$ of x being generated in state q in order to derive the most likely sequence of internal states. Different approaches exist to model the emission probabilities. They will be discussed next.

Discrete HMM

If x is from a discrete set of possible elements $\{x_1, x_2, \dots, x_n\}$, a simple list can be used to store the individual probabilities. Such a model is called *discrete HMM* and can be used for handwriting recognition by applying a vector space quantization technique on the input elements. The simple representation, fast computation, and the need to train only a few parameters are the most prominent advantages of this approach.

Continuous HMM

When using real-valued, continuous vectors, arbitrary probability density functions

$$b_i : \mathbb{R}^n \to \mathbb{R} \ \text{ with } \int_{\mathbb{R}^n} b_i(x)\,\mathrm{dx} = 1$$

need to be modeled. This is commonly done in one of three ways, using *continuous HMM*, *semicontinuous HMM*, or *hybrid* models.

In the continuous case, b_i is approximated by a weighted sum of Gaussian normal distributions

$$b_i(x) = \sum_{k=1}^{g_i} c_{k,i} \cdot \mathcal{N}(x; \mu_{k,i}, K_{k,i})$$

with $\sum_{k=i}^{g_i} c_{k,i} = 1$ for all i and

$$\mathcal{N}(x; \mu, K) = \frac{1}{\sqrt{2\pi\,|K|}} \cdot \exp\left(-\frac{1}{2}(x-\mu)' K^{-1}(x-\mu)\right) \ ,$$

where $i = 1, \ldots, n$ indicates the state and $k = 1, \ldots, g_i$ the specific Gaussian. Since any probability density function can still be approximated if the covariance matrices are diagonal matrices, a complete description of continuous HMMs is therefore given by a state transition probability matrix A, a vector indicating the starting probabilities π and a set of weights, means, and the diagonal elements for the covariance matrices for each state $(c_{k,i}, \mu_{k,i}, \sigma_{k,i,1}, \ldots, \sigma_{k,i,n})$. The number of Gaussian distributions used in the sum to approximate the density function can be chosen by the user.

Semicontinuous HMM

Semicontinuous HMMs make use of the same principle. However, instead of approximating each state's emission probability density function with an own set of Gaussians, one common pool $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_l\}$ of l Gaussian distribution is used. This way, only the weights $c_{k,i}$ are specific to a state. Obviously, this approach offers other advantages in addition to a reduced set of parameters that need to be trained. The Gaussians are state independent and can therefore be learned in an unsupervised way, possibly making use of unlabeled data [48].

Hybrid Approaches

A completely different approach is taken by the so-called hybrid methods. In an ANN/HMM approach [14], a neural network is trained in such a way that it returns for a given element x the probability for being emitted in each state (See Fig. 12.9). Hence, instead of modeling $p(x|q_i)$ using a weighted sum of Gaussians, $p(q_i|x)$ is modeled. Using Bayes' rule, the target function can be rewritten

$$p(x|q_i) = \frac{p(q_i|x)p(x)}{p(q_i)} \ .$$

The probability $p(q_i)$ of the HMM being in a specific state can be directly computed from the Model's state transition probabilities. The a priori probability $p(x)$ is a constant for all states and does therefore not contribute to finding the most likely state transitions. Consequently, it is set to 1. Hence, the likelihood ratio $\frac{p(q_i|x)}{p(q_i)}$ is used instead of the emission probability.

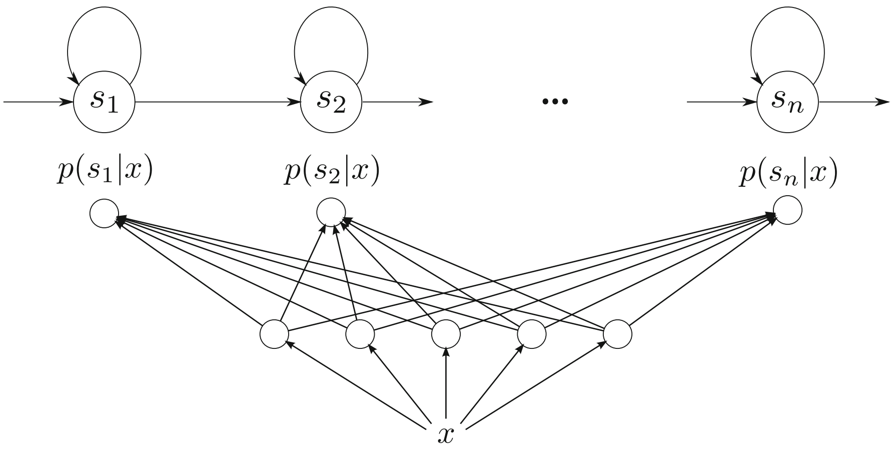


Fig. 12.9 In an ANN/HMM hybrid approach, a neural network models the emission probability of each state

By using a sophisticated learning-based approach, any function can theoretically be modeled, just like a weighted sum of Gaussian. However, unlike the classical Baum-Welch approach, training of the neural network is discriminant. Adjusting the weights according to a training sample affects the probability estimates for all states. The authors in [14] also report a remarkable speedup compared to evaluating Gaussian distributions.

The training of hybrid models pose difficulties as the target output for each state must be known for the back-propagation algorithm. One approach is to label every element of every sequences in the training set. Given a transcription, a hidden Markov model can return the most likely segmentation of the sequence into the individual states in a preprocessing phase. Then, during training, back-propagation and Baum-Welch iterations alternate [8]. Also, more sophisticated approaches have been proposed that optimize the neural network and hidden Markov model simultaneously [8].

Language Model Integration

The accuracy of a recognition system can be increased by enhancing the estimation of the prior probabilities $P(s|\lambda)$ of state sequences s in the term

$$s_{\text{best}} = \arg\max_{s \in S*} P(O, s|\lambda) = \arg\max_{s \in S*} P(O|s, \lambda) P(s|\lambda) \ .$$

With external language information, word transition probabilities $p(w_1 \to w_2)$ for each pair of words, so-called *bigram* probabilities, are computed. However, since neither the result of the decoding algorithm nor the bigram probabilities are true probabilities [40], the language information is included via two parameters that also control the influence of the language model on the recognition. These two

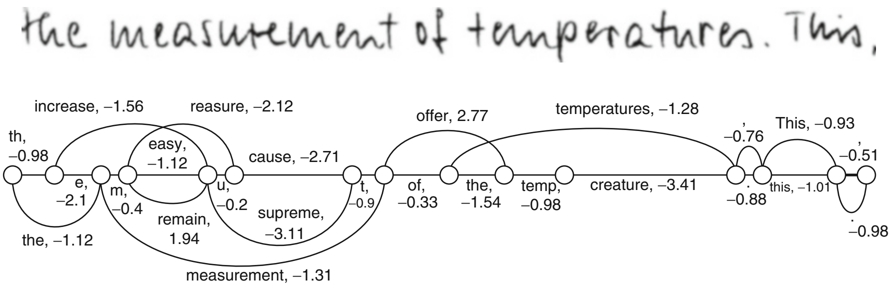


Fig. 12.10 A sample recognition lattice

parameters are the *grammar scale factor* (*gsf*) and the *word insertion penalty* (*wip*). The *wip* is a fixed value added to the likelihood of every new word. The *gsf* is a factor by which the bigram probability is scaled before it is also added.

The resulting likelihood of a word sequence W, given an observation O, is therefore

$$\log \hat{P}(O, W|\lambda) = \log P(O|W, \lambda) + gsf \cdot \log P(W) - l \cdot wip$$

where l is the number of words in the sequence.

Using a Token Passing algorithm, it is not only possible to produce a n-best list for single-word recognition but also to produce a recognition lattice. A recognition lattice is a graph, containing the positions of possible word boundaries as node labels and possible words as well as their recognition likelihood as edge labels, as demonstrated in Fig. 12.10. A recognition lattice, produced by the Token Passing algorithm, resembles the subspace of the most likely recognition results.

The so-called bigram probability $p(w_2|w_1)$ of word w_2 to follow w_1 can be estimated using an external text corpus. This corpus does not need to be handwritten text, but instead should be a representative cross section of the underlying language. For a given word w_1, one can simply count how often w_2 succeeds it. Nevertheless, using this technique, frequent word pairs are likely to be overestimated, while not all possible word combinations might appear in the corpus, which would result in a probability of zero. Hence, post-processing methods are applied that take probability mass from frequent word pairs and distribute it over the rest. This technique is also known as *smoothing*. A detailed description of language modeling can be found in [21].

BLSTM Neural Network with CTC Token Passing Algorithm

The second type of recognizer presented in this chapter is based on neural networks. Their general applicability and good performance have made neural networks popular for diverse classification tasks in document analysis. Especially for isolated character recognition, but also for online and off-line recognition of single words, neural networks have been intensively utilized. For more details on this, see ▶Chaps. 11 (Handprinted Character and Word Recognition) and ▶26 (Online Handwriting Recognition) of this book.

To decode feature sequences representing text of arbitrary length, the application of neural networks is not completely straightforward since the output is not a single element but a sequence. Actually, one needs to proceed in two steps.

In the first step, a character probability vector for each position in the sequence is estimated. It indicates the probability for each character to be written at that position in the text line. The estimation is done using the actual neural network. In the second step, the probability vector sequence is transformed into a possibly shorter sequence of likely characters or words. This is achieved by using an algorithm called *Connectionist Temporal Classification* (CTC) Token Passing algorithm [25].

Neural Networks

The idea behind handwriting recognition using artificial neural networks is to read the sequence of feature vectors, extracted from the image of a text line, into the network. At each position of the input sequence, the network produces an output that is to be interpreted in a subsequent step. From this point of view, neural networks can be regarded as a nonlinear feature transformation function. In the ideal case, the output is a sequence of symbols that can easily be converted into a sequence of words. In a more complex scenario, sophisticated methods, possibly even hidden Markov models, have to be applied.

Using a feed-forward network, only a fixed-sized window of the sequence can be observed at each position. Contextual information, which is important to recognize handwritten text, can only be considered to a limited degree. A recurrent neural network has the advantage that it can be used like a memory to store information over several time steps. However, the value either exponentially decays or increases as it circles around the network, depending upon the weights of the recurrent connections. This drawback is called *vanishing gradient problem* and poses a huge problem for current learning techniques. Hence, such recurrent neural networks are only capable of successfully storing and using information for a few time steps [5].

A recently proposed solution to this problem is *long short-term memory* (LSTM) cells. These cells are specifically designed to address the vanishing gradient problem [28]. An illustration of an LSTM cell can been seen in Fig. 12.11. A core node, with a recurrent weight of 1.0, is in the middle of the cell and serves as a memory. Three nodes are used to control the information flow from the network into the cell and from the cell back into the network. The net input node receives values from the network and forwards its activation into the cell. However, the value is only passed into the core if a second node, the input gate, is activated, or open. Similarly, an output gate regulates if the core's value is fed into the network. A fourth node, the forget node, resets the core's value upon activation. Using these memory cells, information can be stored without loss for arbitrarily many time steps. Interestingly, a similar node architecture has been found in the cortex of the human brain [41].

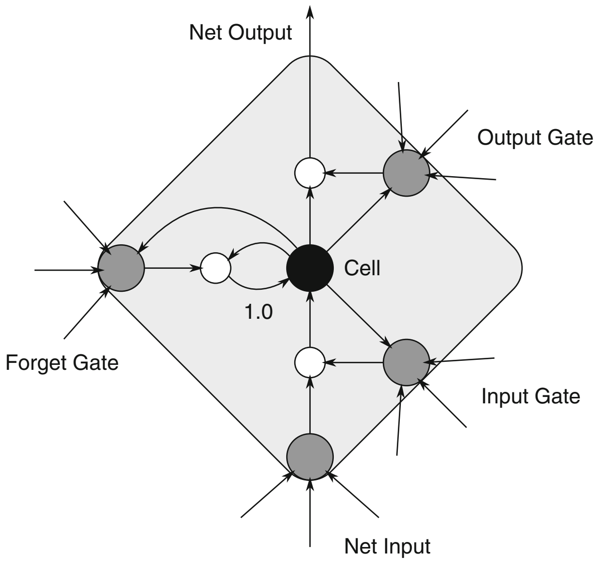


Fig. 12.11 Gates control the information flow into and out of each LSTM cell [25]

BLSTM Neural Networks

In sequential data, information is often spread across several time steps. Recurrent neural networks are able to use information from previously processed sequence elements. To use the information not only from past but also from future sequence positions, *bidirectional* LSTM (BLSTM) neural networks can be used. Two distinct parts of the network, each endowed with input and LSTM layers, process the sequence separately; one from left to right and the other from right to left. For off-line information like scanned handwritten text lines, it is not unusual to access context to the left and to the right of each position to recognize a character. For online data, such as speech, an input sequence has to be recorded and segmented into larger segments. This does not necessarily pose a problem, since obvious segmentation points, such as silence between two sentences, can be used. The signal is saved to memory, segmented, and each segment is then processed separately. A BLSTM NN with one hidden LSTM layer processing a line of handwritten text is depicted in Fig. 12.12.

The output layer contains not only one node for each of the possible classes but also one extra node, the ε node, to indicate that no decision about the output class can be made at that position.

The activation function in the output layer realizes the softmax normalization

$$y_n(t) = \frac{\mathrm{e}^{a_n(t)}}{\sum_k \mathrm{e}^{a_k(t)}} \ ,$$

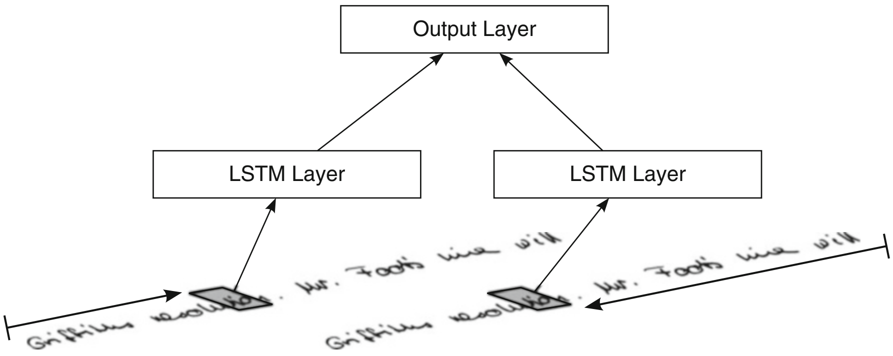


Fig. 12.12 An illustration of the mode of operation of the BLSTM neural network. For each position, the output layer sums up the values of the two hidden LSTM layers

where $a_n(t)$ is the input into node n at time t. Due to the normalization, all output activations sum up to one and $y_n(t)$ can be treated as a posterior class probability $p(\text{class}(O(t)) = n)$ that the class of the sequence O at time t is n.

In Fig. 12.13 such a probability sequence, where each node corresponds to a character, can be seen. The activation is close to 0 most of the time for normal letters and peaks only at distinct positions. In contrast, the activation level of the ε node is nearly constantly 1.

The underlying rationale is now to find a path through the matrix of output activations that represents a meaningful character sequence while simultaneously maximizing its probability score. The probability $P(s|O)$ of a path s through the output sequence $p(c_{k_1} c_{k_2} \ldots_{k_n} |O)$ is given by multiplying the individual probabilities

$$P(s|O) = \prod_{t=1}^{T} y_{k_t}(t) \ .$$

To recognize class sequences that might be shorter than the input sequence, a given path can be shortened using operator $\mathcal{B}$. The operator first deletes consecutive occurrences of the same class and then all ε entries:

$$\mathcal{B}(c_1, c_2, c_2, c_2, \varepsilon, c_2) = c_1 c_2 c_2$$
$$\mathcal{B}(\varepsilon, c_1, c_1, c_1, c_2, \varepsilon, \varepsilon, c_2) = c_1 c_2 c_2$$
$$\mathcal{B}(\varepsilon, c_1, c_1, c_1, \varepsilon, c_2, c_2) = c_1 c_2$$

Recognition via the CTC Token Passing Algorithm

The most simple form text recognition using BLSTM NN is to concatenate the most likely character for each time step and apply the shortening operator $\mathcal{B}$. In this case, no dictionary or language model is used and a high-recognition accuracy cannot be expected necessarily. Using dynamic programming, however, a dictionary

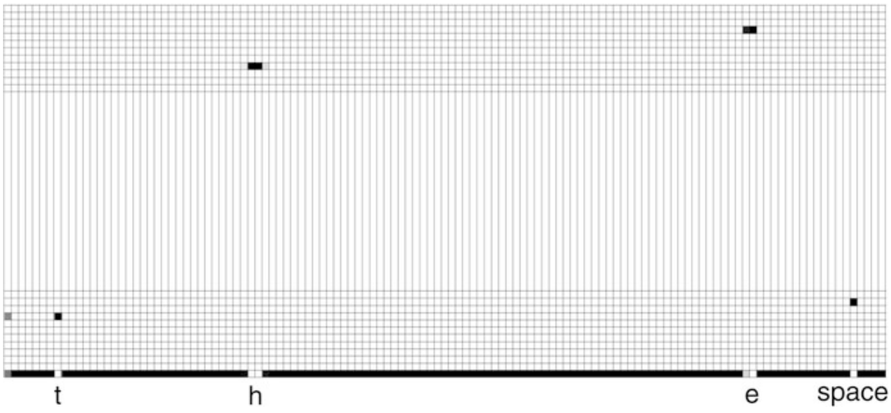


Fig. 12.13 An actual output produced from an image of the word "the"

of possible words as well as a language model can be included to improve the recognition accuracy. When only a single word is to be recognized, the optimal path that result in a word from the dictionary using operator $\mathcal{B}$ can be found efficiently. For continuous text recognition, a more complex version, the *Connectionist Temporal Classification* (CTC) *Token Passing*, is used that takes bigram probabilities into account to connect different words. A finite-state machine is constructed out of each word representing the inverse function of the shortening operator $\mathcal{B}^{-1}$ by allowing transitions to the ε-node between consecutive characters and direct connections between characters only if they are different. Then, similar to HMM-based recognition where each word is represented as a Markov model, tokens are passed along, taking into account word transition probabilities given by the language model and character observation probabilities given by the network's output nodes.

Training

To train a neural network, the weights of all internal connections are initialized randomly and then gradually changed to minimize a global objective function $\mathcal{O}$ via gradient descent and *back-propagation through time* (BPTT) [64]. For that, the gradient of the objective function w.r.t. the network input for output node n for each time step t is needed.

Given a training set of samples (O_i, W_i) consisting of pairs of observation sequences O_i and a word sequence W_i, the goal is to maximize the probability of correctly labeling the entire training set $\prod_i P(W_i|O_i)$, or equivalently, minimizing the log probabilities

$$\begin{aligned}\mathcal{O} &= -\sum_i \ln P(W_i|O_i)\\ &= -\ln \sum_{s\in\mathcal{B}^{-1}(W_i)} P(s|O_i)\ ,\end{aligned}$$

since $P(W|O)$ is the sum of all paths s through the matrix of output activations that represent the word sequence. Keeping track of all possible paths is not traceable, but we can split the sum into parts using the Forward-Backward algorithm. Given a point in time t and a character c, the probability of an admissible path to be at time t in character c can be expressed as

$$P(W|O, s_t = c) = \sum_{s \in \mathcal{B}^{-1}(W) \wedge s_t = c} P(s|O) \ .$$

Now, with any arbitrary t, the probability for a single training example $P(W|O)$ can be computed by summing up all characters in the alphabet A:

$$P(W|O) = \sum_{c \in A} P(W|O, s_t = c) \ .$$

As shown in [23], the sought-after gradient can be expressed in terms of these partial sums:

$$\frac{\partial \mathcal{O}}{\partial a_j(t)} = y_j(t) - \frac{P(W|O, s_t = c)}{P(W|O)} \ .$$

This gradient can then be propagated back into the network to adjust the individual weights. In case of several training elements, the weights can be adjusted after each training element or after computing the average gradients over the entire training set. A separate validation set is used to check the performance of the neural network after each back-propagation iteration to prevent overfitting.

Summary of the Recognizer

Table 12.2 points out the key references of the presented systems, and a comparison showing the main advantages and disadvantages of the presented recognizers is given in List 12.2. Hidden Markov models are well understood and have a long tradition in being used as recognizers for sequential data. As a consequence, a variety of specific HMM type have been developed for different scenarios. Their capacity to model complex sequences but also the need for training data increases from discrete HMM, via semicontinuous HMM, to continuous HMM. In contrast, the performance of ANN/HMM systems depends on the neural network that models the state probabilities. The hybrid ANN/HMM systems requires a frame-level ground truth, that is, each single-feature vector has to be assigned to a character. However, an approximation is enough for a successful training so that a continuous HMM can be used in a preprocessing step to label each frame.

Table 12.2 An overview of the most frequently used and established recognition methods

	System	Training	Recognition	Key references
HMM	Discrete	Baum-Welch	Viterbi	[1, 33]
	Continuous	Baum-Welch	Viterbi	[38, 62]
	Semicontinuous	Baum-Welch	Viterbi	[65]
	Hybrid (ANN)	Baum-Welch and back-propagation	Viterbi	[8, 14]
BLSTM NN		Back-propagation	CTC Token Passing	[25]

The BLSTM neural network and the hybrid ANN/HMM both belong to the best performing approaches. Furthermore, both systems have only been introduced recently and therefore still have a large potential of improvement.

Datasets, Software, and Evaluation Methodology

Overview

In this section, some of the key databases to train and evaluate off-line handwriting recognition methods are presented. Moreover, an overview of software for the HMM and BLSTM NN recognition technologies is presented in section "Recognition Methods," as well as software tools to create statistical language models. Finally, details for evaluating the recognition rate of a system for continuous handwriting recognition are explained.

Databases

Due to the variety of possible applications, common databases cover different areas of the field. They range from databases containing historic documents written by a single writer to a collection of modern, French letters. The following list is an overview of commonly used sources of annotated, continuous handwritten data. A regularly updated repository of databases can be found at the website of the IAPR Technical Committee 11 – Reading Systems.[1] A summary of the presented databases is given in Table 12.3.

Modern Databases

One of the largest and most frequently used databases for modern continuous handwritten data is the *IAM Handwriting Database*[2] [39], consisting of English texts from the Lancaster-Oslo/Bergen Corpus of British English [30]. It therefore forms a representative cross section of the English language. The *RIMES*[3] database is a collection of artificially created letters to individuals and companies in French, *Hidden Markov models:*

+ Very well understood.
+ As a generative model, new characters can be added easily to an existing system.
− Markov property restricts the inclusion of long-term dependencies.
− The returned likelihood values do not reflect a recognition confidence.

Discrete-HMM with feature vector quantization:

+ Easy and fast approach.
+ Only few parameters to train, works with limited training data.
− Complex probability distribution functions cannot be modeled accurately.

Semicontinuous HMM:

+ No vector quantization needed.
+ The estimation of the individual Gaussian distributions can be done with unlabeled data.
+ A state is given by a small set of coefficients.
+ Fast to train, robust for limited training data.
− Some states' probability distribution functions might only be modeled insufficiently.

Continuous HMM:

+ Can make use of abundant training data.
+ The complexity of each state can be modeled independently.
− Many parameters to train.

Hybrid HMM/ANN:

+ Complex probability distributions functions are learned in a discriminative way.
+ Evaluating the neural network instead of Gaussian distributions decreases the runtime.
+ Among the best performing systems.
− Training is slow and requires a frame-accurately labeled ground truth.

BLSTM neural network:

+ The network returns true character occurrence posterior probabilities.
+ The random initialization allows a straightforward generation of recognizer ensembles.
+ Automatic learning of long-term dependencies.
+ Among the best performing approaches to date.
− Random initialization of the neural networks requires that several networks need to be trained and tested to guarantee a good performance.
− Additional characters cannot be added to an existing system.
− An activation peak of an output node does not occur at a predictable position (beginning, middle, end) of the character. Character and word segmentation using BLSTM NNs is difficult.
− Works only well with large amounts of training data.
− Not as well understood and frequently used as HMMs. Nearly no existing software packages.

List 12.2 A list of advantages and disadvantages for different types of recognizers for continuous handwriting recognition

which makes it therefore useful for business-related applications. Online databases also form an interesting source of data. The given pen trajectories can easily be mapped onto an image to create off-line handwritten texts. Among online databases, *UNIPEN*[4] [26] is the most widely used one. Frequently held "calls for data" ensure a constant growth of the database.

Historical Databases

Due to the nature of historical documents, the recognition task on these datasets is usually much harder than on modern datasets. The spelling is not standardized, the paper might be degraded with barely legible ink, and the pages can be highly

Table 12.3 Overview of databases containing continuous handwritten text

			Size		
Name	Language	Time	KWords	Writer	Comments
IAM DB	English	Modern	115	657	
RIMES	French	Modern	250	1,300	Letters
UNIPEN	Several	Modern	74	320	Online data
GW	English	Eighteenth century	Various sets are used		
GERMANA	Several	Nineteenth century	217	1	
RODRIGO	Spanish	Sixteenth century	231	1	

ornamented. On the other hand, the writing style itself is usually more regular within one document. A specific set of *Letters, Orders and Instructions* of George Washington, the GW[5] dataset, evolved to a de facto standard for word retrieval tasks [46].

The *GERMANA* and *RODRIGO* datasets[6] represent historic Spanish books from the nineteenth and the sixteenth century, respectively. The GERMANA dataset also contains passages written in other European languages.

Software

Several software tools for hidden Markov models are available on the Internet, such as the widespread *Hidden Markov Model Toolkit (HTK)*[7] [66] and the *Hidden Markov Model Toolbox for Matlab*.[8] HTK was created to use HMMs for speech recognition and offers sophisticated tools for manipulating existing models. Furthermore, general toolkits for machine learning exist, e.g., TORCH[9] [10], that include options for using HMMs, while for other toolkits, such as WEKA[10] [27], third-party extensions exist.

Comparable toolkits for BLSTM neural networks are not as well developed. To the knowledge of the author, RNNLIB[11] [25] is the only available source of software for this new type of neural network architecture.

To create and modify language models, plenty of programs exist. While, e.g., the MIT[12], the CMU[13], and Microsoft®[14] all have their own toolkits, the most frequently used one is *SRILM*[15] [54].

Evaluation Methodology

The recognition accuracy of a system that classifies single objects, as in the single-word recognition case, is given by the fraction of correctly recognized elements in relation to all classified elements. Let n indicate the number of tested elements, out of which c have been correctly recognized. Then, the accuracy is given by

$$\text{Accuracy} = \frac{c}{n} \ .$$

For sequential data, when the number of classes is not known, several error cases can occur. First, an element might be correctly recognized as an element, but as an element of the wrong class. This is called a *substitution*. Secondly, an element might falsely be recognized or inserted. Consequently, this is called an *insertion*. Thirdly, an element might not be recognized at all. This is called a *deletion*. Finally, the accuracy is defined by

$$\text{Acc} = \frac{n - S - D - I}{n}$$

where S is the number of substitutions, D is the number of deletions, and I is the number of insertions [62]. In case of text line recognition, the elements are words. For the unconstrained character sequence recognition task, the elements are characters.

The computation of the accuracy can be reduced to computing the string edit distance. Using dynamic programming, the recognized string is matched with the string in the ground truth. This procedure directly returns the number of substitutions, deletions, and insertions. Note that the accuracy can be negative. The edit costs used in the HTK Software [66] for computing the string edit distance are

$$\begin{aligned} c_{\text{substitution}}(u \to u) &= 0 \\ c_{\text{substitution}}(u \to v) &= 10 \\ c_{\text{insertion}}(\varepsilon \to u) &= 7 \\ c_{\text{deletion}}(u \to \varepsilon) &= 7 \end{aligned}$$

where $u \neq v$ are two different elements and ε indicates the *empty word*.

Conclusion

In this chapter, an overview of the current state-of-the-art research on continuous handwriting recognition is given, with a focus on off-line, English texts. Segmenting a text line into individual characters or just words can be a challenging task. Contextual information and sometimes even a semantic understanding of the content is needed, which can only be obtained after recognition. Therefore, modern recognition systems are all segmentation-free, sequential approaches which puts handwriting recognition closer to speech recognition than to the recognition of isolated machine-printed characters.

The normalization of the writing slant and the height of the three text areas reduces writing style variation. After normalizing the text line, it is converted into a sequence of feature vectors using a sliding window. Traditionally, recognition is done using hidden Markov models. A recently proposed type of neural network, however, has introduced a novel recognition technology to the field. Interestingly, these BLSTM neural networks and recent ANN/HMM hybrid systems perform equally well. Further advances have been achieved by replacing more and more steps in the entire processing chain with learning-based systems instead of simple mathematical functions, such as text normalization or feature extraction.

Yet, it has become clear that the construction of a universal recognition system with 100 % accuracy is still impossible to reach in the near future. Open problems arise from the huge variety of different writing styles and the need to grasp the meaning of the text that is to be transcribed. Statistical language models do certainly not provide enough context knowledge and leave no room for out-of-vocabulary words.

Notes

[1]http://www.iapr-tc11.org

[2]http://www.iam.unibe.ch/fki/databases

[3]http://www.rimes-database.fr

[4]http://unipen.nici.kun.nl

[5]George Washington Papers at the Library of Congress, 1741–1799: Series 2, Letterbook 1, pages 270–279 & 300–309, http://memory.loc.gov/ammem/gwhtml/gwseries2.html

[6]http://prhlt.iti.upv.es/page/projects/multimodal/idoc

[7]http://htk.eng.cam.ac.uk

[8]http://www.cs.ubc.ca/∼murphyk/Software/HMM/hmm.html

[9]http://torch5.sourceforge.net

[10]http://www.cs.waikato.ac.nz/ml/weka

[11]http://rnnl.sourceforge.net/

[12]http://code.google.com/p/mitlm

[13]http://www.speech.cs.cmu.edu/SLM_info.html

[14]http://research.microsoft.com/apps/pubs/default.aspx?id=70505

[15]http://www.speech.sri.com/projects/srilm

References

1. Arica N, Yarmal-Vural FT (2002) Optical character recognition for cursive handwriting. IEEE Trans Pattern Anal Mach Intell 24(6):801–814
2. Ahmed P, Suen CY (1987) Computer recognition of totally unconstrained handwritten zip codes. Int J Pattern Recognit Artif Intell 1(1):1–15
3. Baum LE, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Ann Math Stat 41(1):164–171
4. Bazzi I, Schwartz R, Makhoul J (1999) An omnifont open-vocabulary OCR system for English and Arabic. IEEE Trans Pattern Anal Mach Intell 21(6):495–505

5. Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient decent is difficult. IEEE Trans Neural Netw 5(2):157–166
6. Božinović RM, Srihari S (1989) Off-line cursive script word recognition. IEEE Trans Pattern Anal Mach Intell 11(1):68–83
7. Brakensiek A, Rigoll G (2004) Handwritten address recognition using hidden Markov models. In: Dengel A et al (eds) Reading and learning. Volume 2956 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 103–122
8. Caillault É, Viard-Gaudin C (2007) Mixed discriminant training of hybrid ANN/HMM. Int J Pattern Recognit Artif Intell 21(1):117–134
9. Chen J, Cao H, Prasad R, Bhardwaj A, Natarajan P (2010) Gabor features for offline Arabic handwriting recognition. In: 9th IARP international workshop on document analysis systems, Boston, pp 53–58
10. Collobert R, Bengio S, Mariéthoz J (2002) TORCH: a modular machine learning software library. Technical report IDIAP-RR 02-46, IDIAP Research Institute
11. Dietterich T (2009) Machine learning for sequential data: a review. In: Caelli T, Amin A, Duin R, de Ridder D, Kamel M (eds) Structural, syntactic, and statistical pattern recognition. Volume 2396 of lecture notes in computer science. Springer, Berlin/Heidelberg, pp 227–246
12. Do TMT, Artieres T (2006) Conditional random fields for online handwriting recognition. In: International workshop of frontiers in handwriting recognition, La Baule, pp 197–204
13. Duda RO, Hart PE, Stork DG (2001) Pattern classification, 2nd edn. Wiley-Interscience, New York
14. España-Boquera S, Castro-Bleda MJ, Gorbe-Moya J, Zamora-Martinez F (2010) Improving offline handwritten text recognition with hybrid HMM/ANN models. IEEE Trans Pattern Anal Mach Intell 33(4):767–779
15. Fischer A, Bunke H (2009) Kernel PCA for HMM-based cursive handwriting recognition. In: 13th international conference on computer analysis of images and pattern, Münster, pp 181–188
16. Fischer A, Riesen K, Bunke H (2010) Graph similarity features for HMM-based handwriting recognition in historical documents. In: 12th international conference on frontiers in handwriting recognition, Kolkata. pp 253–258
17. Fischer A, Keller A, Frinken V, Bunke H (2011, submitted) Lexicon-free handwritten word spotting using character HMMs pattern recognition letters 33(7):934–942
18. Frinken V, Fischer A, Manmatha R, Bunke H (2012) A novel word spotting method based on recurrent neural networks. IEEE Trans Pattern Anal Mach Intell 34(2), pp 211, 224
19. Gader PD, Mohamed MA, Chiang J-H (1995) Comparison of crisp and fuzzy character neural network in handwritten word recognition. IEEE Trans Fuzzy Syst 3(3):357–364
20. Gader PD, Mohamed MA, Chiang J-H (1997) Handwritten word recognition with character and inter-character neural networks. IEEE Trans Syst Man Cybern 27(1):158–164
21. Goodman JT (2001) A bit of progress in language modeling – extended version. Technical report MSR-TR-2001-72, Microsoft Research, One Microsoft Way Redmond, WA 98052, 8
22. Gorski N, Anisimov V, Augustin E, Baret O, Maximov S (2001) Industrial bank check processing: the A2iA CheckReader(tm). Int J Doc Anal Recognit 3(4):196–206
23. Graves A (2012) Supervised sequence labelling with recurrent neural networks. Springer, Heidelberg/New York/Dordrecht/London
24. Graves A, Schmidhuber J (2009) Offline handwriting recognition with multidimensional recurrent neural networks. In: Koller D et al (eds) Advances in neural information processing systems 21. MIT Press, Cambridge, pp 545–552
25. Graves A, Liwicki M, Fernández S, Bertolami R, Bunke H, Schmidhuber J (2009) A novel connectionist system for unconstrained handwriting recognition. IEEE Trans Pattern Anal Mach Intell 31(5):855–868
26. Guyon I, Schomaker L, Plamondon R, Liberman M, Janet S (1994) UNIPEN project of on-line data exchange and recognizer benchmarks. In: 12th international conference on pattern recognition, Jerusalem, pp 29–33

27. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD Explor 11(1):10–18
28. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
29. Jiang H (2005) Confidence measures for speech recognition: a survey. Speech Commun 45:455–470
30. Johanson S, Leech GN, Goodluck H (1978) Manual of information to accompany the Lancaster-Oslo/Bergen corpus of British English, for use with digital computers. Technical report, Department of English, University of Oslo, Norway
31. Kavallieratou E, Fakotakis N, Kokkinakis GK (2002) An unconstrained handwriting recognition system. Int J Doc Anal Recognit 4(4):226–242
32. Knerr S, Augustin E, Baret O, Price D (1998) Hidden Markov model based word recognition and its application to legal amount reading on French checks. Comput Vis Image Underst 70(3):404–419
33. Koerich AL, Sabourin R, Suen CY (2003) Lexicon-driven HMM decoding for large vocabulary handwriting recognition with multiple character models. Int J Doc Anal Recognit 6:126–144
34. Lee S-W (ed) (1999) Advances in handwriting recognition. World Scientific, Singapore/River Edge/London
35. Liwicki M, Graves A, Bunke H (2012) Neural networks for handwriting recognition. In: Ogiela MR, Jain LC (eds) Computational intelligence paradigms in advanced pattern classification, vol 386/2012. Springer, Berlin/Heidelberg, pp 5–24
36. Lorigo LM, Govindaraju V (2006) Offline Arabic handwriting recognition: a survey. IEEE Trans Pattern Anal Mach Intell 28(5):712–725
37. Marti U-V (2000) Off-line recognition of handwritten texts. PhD thesis, University of Bern, Bern
38. Marti U-V, Bunke H (2001) Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. Int J Pattern Recognit Artif Intell 15:65–90
39. Marti U-V, Bunke H (2002) The IAM-database: an English sentence database for offline handwriting recognition. Int J Doc Anal Recognit 5:39–46
40. Ogawa A, Takeda K, Itakura F (1998) Balancing acoustic and linguistic probabilities. In: International conference on acoustic, speech, and signal processing, Seattle, pp 181–184
41. O'Reilly RC, Frank MJ (2003) Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia. Ics-03-03, ICS, Mar 2003
42. Pechwitz M, Maergner V (2003) HMM based approach for handwritten Arabic word recognition using the IFN/ENIT-database. In: 7th international conference on document analysis and recognition, Edinburgh, pp 890–894
43. Plötz T, Fink G (2011) Markov models handwriting recognition. Springer, London/Dordrecht/Heidelberg/New York
44. Pramod Sankar K, Ambati V, Pratha L, Jawahar CV (2006) Digitizing a million books: challenges for document analysis. In: 7th IAPR workshop on document analysis systems, Nelson, pp 425–436
45. Rabiner L (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77(2):257–286
46. Rath TM, Manmatha R (2007) Word spotting for historical documents. Int J Doc Anal Recognit 9:139–152
47. Rodríguez JA, Perronnin F (2008) Local gradient histogram features for word spotting in unconstrained handwritten documents. In: 11th international conference frontiers in handwriting recognition, Montréal, pp 7–12
48. Rodríguez-Serrano JA, Perronnin F, Sánchez G, Lladós J (2010) Unsupervised writer adaptation of whole-word HMMs with application to word-spotting. Pattern Recognit Lett 31(8):742–749
49. Rutovitz D (1966) Pattern recognition. J R Stat Soc A (General) 129(4):504–530
50. Sayre KM (1973) Machine recognition of handwritten words: a project report. Pattern Recognit 3(3):213–228

51. Seiler R, Schenkel M, Eggimann F (1996) Off-line cursive handwriting recognition compared with on-line recognition. In: 13th international conference on pattern recognition, Vienna, vol 4, pp 505–509
52. Senior AW, Robinson AJ (1998) An off-line cursive handwriting recognition system. IEEE Trans Pattern Anal Mach Intell 20(3):309–321
53. Srihari SN, Srinivasan H, Huang C, Shetty S (2006) Spotting words in Latin, Devanagari and Arabic scripts. Indian J Artif Intell 16(3):2–9
54. Stolke A (2002) SRILM – an extensible language modeling toolkit. In: International conference on spoken language processing, Denver, pp 901–904
55. Taira E, Uchida S, Sakoe H (2004) Nonuniform slant correction for handwritten word recognition. IEICE Trans Inf Syst E87-D(5):1247–1253
56. Terasawa K, Tanaka Y (2009) Slit style HOG features for document image word spotting. In: 10th international conference on document analysis and recognition, Barcelona, vol 1, pp 116–120
57. Toselli AH, Juan A, González J, Salvador I, Vidal E, Casacuberta F, Keysers D, Ney H (2004) Integrated handwritten recognition and interpretation using finite-state models. Int J Pattern Recognit Artif Intell 18(4):519–539
58. Toselli AH, Vidal E, Casacuberta F (2011) Multimodal interactive pattern recognition and applications. Springer, London/New York
59. van der Zant T, Schomaker L, Haak K (2008) Handwritten-word spotting using biologically inspired features. IEEE Trans Pattern Anal Mach Intell 30(11):1945–1957
60. Vinciarelli A (2002) A survey on off-line cursive word recognition. Pattern Recognit 35(7):1433–1446
61. Vinciarelli A (2003) Offline cursive handwriting: from word to text recognition. Technical report IDIAP-PP 03-24, Institut Dalle Molle Intelligance Artificielle Perceptive (IDIAP), Martigny
62. Vinciarelli A, Bengio S, Bunke H (2004) Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. IEEE Trans Pattern Anal Mach Intell 26(6): 709–720
63. Viterbi A (1967) Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans Inf Theory 13:260–269
64. Werbos PJ (1988) Generalization of backpropagation to a recurrent gas model. Neural Netw 1:339–356
65. Wienecke M, Fink GA, Gerhard S (2005) Toward automatic video-based whiteboard reading. Int J Doc Anal Recognit 7:188–200
66. Young S, Evermann G, Gales M, Hain T, Kershaw D, Liu X, Moore G, Odell J, Ollason D, Povey D, Valtchev V, Woodland P (2006) The HTK book. Technical report, Cambridge University Engineering Department, Dec 2006

Further Reading

Fischer A, Keller A, Frinken V, Bunke H (2011, submitted) Lexicon-free handwritten word spotting using character HMMs

Frinken V, Fischer A, Manmatha R, Bunke H (2012) A novel word spotting method based on recurrent neural networks. IEEE Trans Pattern Anal Mach Intell. Accepted for publication

Goodman JT (2001) A bit of progress in language modeling – extended version. Technical report MSR-TR-2001-72, Microsoft Research, One Microsoft Way Redmond, WA 98052, 8

Graves A (2012) Supervised sequence labelling with recurrent neural networks. Springer, Heidelberg/New York/Dordrecht/London

Jiang H (2005) Confidence measures for speech recognition: a survey. Speech Commun 45: 455–470

Liwicki M, Graves A, Bunke H (2012) Neural networks for handwriting recognition. In: Ogiela MR, Jain LC (eds) Computational intelligence paradigms in advanced pattern classification, vol 386/2012. Springer, Berlin/Heidelberg, pp 5–24

Plötz T, Fink G (2011) Markov models handwriting recognition. Springer, London/Dordrecht/Heidelberg/New York

Rabiner L (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77(2):257–286

Toselli AH, Vidal E, Casacuberta F (2011) Multimodal interactive pattern recognition and applications. Springer, London/New York