A Novel Connectionist System for Unconstrained Handwriting Recognition

Alex Graves, Marcus Liwicki, Santiago Fernández,
Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber

Abstract—Recognizing lines of unconstrained handwritten text is a challenging task. The difficulty of segmenting cursive or overlapping characters, combined with the need to exploit surrounding context, has led to low recognition rates for even the best current recognizers. Most recent progress in the field has been made either through improved preprocessing or through advances in language modeling. Relatively little work has been done on the basic recognition algorithms. Indeed, most systems rely on the same hidden Markov models that have been used for decades in speech and handwriting recognition, despite their well-known shortcomings. This paper proposes an alternative approach based on a novel type of recurrent neural network, specifically designed for sequence labeling tasks where the data is hard to segment and contains long-range bidirectional interdependencies. In experiments on two large unconstrained handwriting databases, our approach achieves word recognition accuracies of 79.7 percent on online data and 74.1 percent on offline data, significantly outperforming a state-of-the-art HMM-based system. In addition, we demonstrate the network's robustness to lexicon size, measure the individual influence of its hidden layers, and analyze its use of context. Last, we provide an in-depth discussion of the differences between the network and HMMs, suggesting reasons for the network's superior performance.

Index Terms—Handwriting recognition, online handwriting, offline handwriting, connectionist temporal classification, bidirectional long short-term memory, recurrent neural networks, hidden Markov model.

1 Introduction

Handwriting recognition is traditionally divided into online and offline recognition. In online recognition, a time series of coordinates, representing the movement of the pen tip, is captured, while in the offline case, only an image of the text is available. Because of the greater ease of extracting relevant features, online recognition generally yields better results [1]. Another important distinction is between recognizing isolated characters or words and recognizing whole lines of text. Unsurprisingly, the latter is substantially harder, and the excellent results that have been obtained for digit and character recognition [2], [3] have never been matched for complete lines. Last, handwriting recognition can be split into cases where the writing style is constrained in some way—for example, only hand-printed characters are allowed—and the more challenging scenario where it is unconstrained. Despite more than 30 years of handwriting recognition research [2], [3], [4], [5], developing a reliable general-purpose system for unconstrained text line recognition remains an open problem.

A well-known testbed for isolated handwritten character recognition is the UNIPEN database [6]. Systems that have been found to perform well on UNIPEN include a writer-independent approach based on hidden Markov models (HMMs) [7]; a hybrid technique called cluster generative statistical dynamic time warping [8], which combines dynamic time warping with HMMs and embeds clustering and statistical sequence modeling in a single feature space; and a support vector machine with a novel Gaussian dynamic time warping kernel [9]. Typical error rates on UNIPEN range from 3 percent for digit recognition to about 10 percent for lowercase character recognition.

Similar techniques can be used to classify isolated words, and this has given good results for small vocabularies (for example, a writer-dependent word error rate of about 4.5 percent for 32 words [10]). However, an obvious drawback of whole word classification is that it does not scale to large vocabularies.

For large vocabulary recognition tasks such as those considered in this paper, the only feasible approach is to recognize individual characters and map them onto complete words using a dictionary. Naively, this could be done by presegmenting words into characters and classifying each segment. However, segmentation is difficult for cursive or unconstrained text unless the words have already been recognized. This creates a circular dependency

- *A. Graves and J. Schmidhuber are with the Institut für Informatik, Lehreinheit VI Robotics and Embedded Systems, Technische Universität München, Boltzmannstraße 3, D-85478 Garching be München, Germany. E-mail: alex.graves@gmail.com.*
- *M. Liwicki is with the Research Group Knowledge Management, DFKI-German Research Center for Artificial Intelligence, Room 3.02, Trippstadter Str. 122, D-67663 Kaiserslautern, Germany. E-mail: liwicki@dfki.uni-kl.de.*
- *S. Fernández is with IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland. E-mail: santiago@idsia.ch.*
- *R. Bertolami and H. Bunke are with the Institute of Computer Science and Applied Mathematics (IAM), Research Group on Computer Vision and Artificial Intelligence (FKI), Neubrückstrasse 10, CH-3012 Bern, Switzerland. E-mail: {bertolami, bunke}@iam.unibe.ch.*

Manuscript received 4 Oct. 2007; revised 27 Feb. 2008; accepted 5 May 2008; published online 21 May 2008.
Recommended for acceptance by J. Hu.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2007-10-0672.
Digital Object Identifier no. 10.1109/TPAMI.2008.137.

between segmentation and recognition that is sometimes referred to as Sayre's paradox [11].

One solution to Sayre's paradox is to simply ignore it and carry out segmentation before recognition. For example, [3] describes techniques for character segmentation, based on unsupervised learning and data-driven methods. Other strategies first segment the text into basic strokes, rather than characters. The stroke boundaries may be defined in various ways such as the minima of the velocity, the minima of the y-coordinates, or the points of maximum curvature. For example, one online approach first segments the data at the minima of the y-coordinates and then applies self-organizing maps [12]. Another, offline, approach uses the minima of the vertical histogram for an initial estimation of the character boundaries and then applies various heuristics to improve the segmentation [13].

A more promising approach to Sayre's paradox is to segment and recognize at the same time. HMMs are able to do this, which is one reason for their popularity in unconstrained handwriting recognition [14], [15], [16], [17], [18], [19]. The idea of applying HMMs to handwriting recognition was originally motivated by their success in speech recognition [20], where a similar conflict exists between recognition and segmentation. Over the years, numerous refinements of the basic HMM approach have been proposed, such as the writer-independent system considered in [7], which combines point-oriented and stroke-oriented input features.

However, HMMs have several well-known drawbacks. One of these is that they assume that the probability of each observation depends only on the current state, which makes contextual effects difficult to model. Another is that HMMs are generative, while discriminative models generally give better performance in labeling and classification tasks.

Recurrent neural networks (RNNs) do not suffer from these limitations and would therefore seem a promising alternative to HMMs. However, the application of RNNs alone to handwriting recognition has so far been limited to isolated character recognition (e.g., [21]). The main reason for this is that traditional neural network objective functions require a separate training signal for every point in the input sequence, which in turn requires presegmented data.

A more successful use of neural networks for handwriting recognition has been to combine them with HMMs in the so-called hybrid approach [22], [23]. A variety of network architectures have been tried for hybrid handwriting recognition, including multilayer perceptrons [24], [25], time delay neural networks [18], [26], [27], and RNNs [28], [29], [30]. However, although hybrid models alleviate the difficulty of introducing context to HMMs, they still suffer from many of the drawbacks of HMMs and they do not realize the full potential of RNNs for sequence modeling.

This paper proposes an alternative approach, in which a single RNN is trained directly for sequence labeling. The network uses the connectionist temporal classification (CTC) output layer [31], [32], first applied to speech recognition. CTC uses the network to map directly from the complete input sequence to the sequence of output labels, obviating the need for presegmented data. We extend the original formulation of CTC by combining it with a dictionary and language model to obtain word recognition scores that can be compared directly with other systems. Although CTC can be used with any type of RNN, best results are given by networks able to incorporate as much context as possible. For this reason, we chose the bidirectional Long Short-Term Memory (BLSTM) [33] architecture, which provides access to long-range context along both input directions.

Fig. 1. Illustration of the recording.

In experiments on large online and offline handwriting databases, our approach significantly outperforms a state-of-the-art HMM-based system on unconstrained text line recognition. Furthermore, the network retains its advantage over a wide range of dictionary sizes.

The paper is organized as follows: Sections 2 and 3 describe the preprocessing and feature extraction methods for the online and offline data, respectively. Section 4 introduces the novel RNN-based recognizer. Section 5 describes the databases and presents the experimental analysis and results. Section 6 discusses the differences between the new system and HMMs and suggests reasons for the network's superior performance. Our conclusions are presented in Section 7.

2 Online Data Preparation

The online data used in our experiments were recorded from a whiteboard using the eBeam interface[1] [34]. As illustrated in Fig. 1, the interface consists of a normal pen in a special casing, which sends infrared signals to a triangular receiver mounted in one of the corners of the whiteboard. The acquisition interface outputs a sequence of (x, y)-coordinates representing the location of the tip of the pen together with a time stamp for each location. The coordinates are only recorded during the periods when the pen tip is in continuous contact with the whiteboard. We refer to these periods as *strokes*. After some standard steps to correct for missing and noisy points [35], the data was stored in XML format, along with the frame rate, which varied from 30 to 70 frames per second.

1. eBeam System by Luidia, Inc.—www.e-Beam.com.

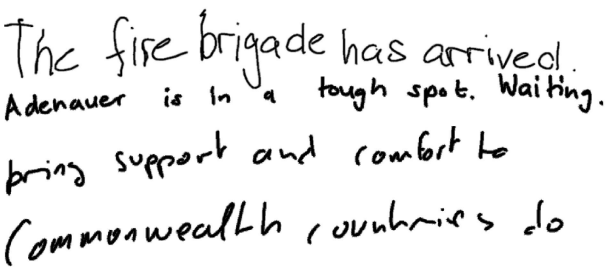


Fig. 2. Examples of handwritten text acquired from a whiteboard.

2.1 Normalization

The preprocessing begins with data normalization. This is an important step in handwriting recognition because writing styles differ greatly with respect to the skew, slant, height, and width of the characters.

Since the subjects stand rather than sit and their arms do not rest on a table, handwriting rendered on a whiteboard is different from that produced with a pen on a writing tablet. In particular, it has been observed that the baseline on a whiteboard cannot usually be approximated by a simple straight line. Furthermore, the size and width of the characters become smaller the more the pen moves to the right. Examples of both effects can be seen in Fig. 2. Consequently, online handwriting gathered from a whiteboard requires some special preprocessing steps.

Since the text lines on a whiteboard usually have no uniform skew, they are split into smaller parts and the rest of the preprocessing is done for each part separately. To accomplish the splitting, all gaps within a line are determined first. The text is split at a gap if it is wider than the median gap width and if the size of both parts resulting from the split is larger than some predefined threshold. An example of the splitting process is shown in Fig. 3 with the resulting parts indicated by lines below the text.

Next, the parts are corrected with respect to their skew and slant. A linear regression is performed through all the points, and the orientation of the text line is corrected according to the regression parameters (see Fig. 3). For slant normalization, we compute the histogram over all angles enclosed by the lines connecting two successive points of the trajectory and the horizontal line [26]. The histogram ranges from -90 to 90° with a step size of 2°. We weight the histogram values with a Gaussian whose mean is at the vertical angle and whose variance is chosen empirically. This is beneficial because some words are not properly corrected

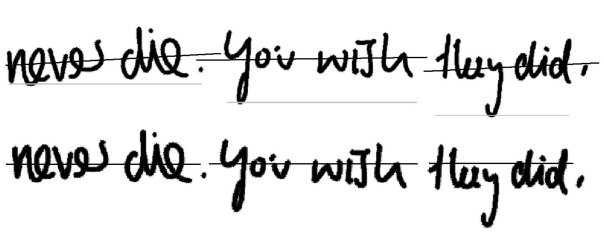


Fig. 3. Processing the text line. (Top) Text line split into individual parts with estimated skew in the middle of the text line. (Bottom) Text line after skew normalization. Note that baseline and corpus line detection (described below) give an improved final estimate of the skew.

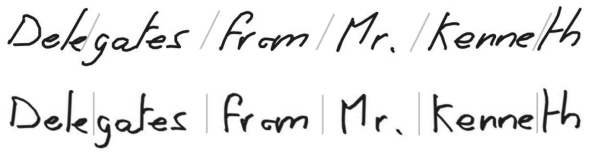


Fig. 4. Slant correction. Gray lines indicate the estimated slant angle.

if a single long straight line is drawn in the horizontal direction, which results in a large histogram value. We also smooth each histogram entry γ_i using its nearest neighbors, $\bar{\gamma}_i = (\gamma_{i-1} + 2\gamma_i + \gamma_{i+1})/4$, because, in some cases, the correct slant is at the border of two angle intervals and a single peak at another interval may be slightly higher. This single peak will become smaller after smoothing. Fig. 4 shows a text line before and after slant correction.

Delayed strokes such as the crossing of a "t" or the dot of an "i" are a well-known problem in online handwriting recognition, because the order in which they are written varies between different writers. For this reason, delayed strokes (identified as strokes above already written parts, followed by a pen movement to the right [26]) are removed. Note that some i dots may be missed by this procedure. However, this usually occurs only if there was no pen movement to the left, meaning that the writing order is not disrupted. A special *hat feature* is used to indicate to the recognizer that a delayed stroke was removed.

To correct for variations in writing speed, the input sequences are transformed so that the points are equally spaced. The optimal value for this distance is found empirically.

The next step is the computation of the baseline and the corpus line, which are then used to normalize the size of the text. The baseline corresponds to the original line on which the text was written, i.e., it passes through the bottom of the characters. The corpus line goes through the top of the lowercase letters. To obtain these lines, two linear regressions through the minima and maxima are computed. After removing outliers, the regression is repeated twice, resulting in the estimated baseline (minima) and corpus line (maxima). Fig. 5 illustrates the estimated baseline and the corpus line of part of the example shown in Fig. 3. The baseline is subtracted from all y-coordinates and the heights of the three resulting areas are normalized.

The final preprocessing step is to normalize the width of the characters. This is done by scaling the text horizontally with a fraction of the number of strokes crossing the horizontal line between the baseline and the corpus line. This preprocessing step is needed because the x-coordinates of the points are taken as a feature.

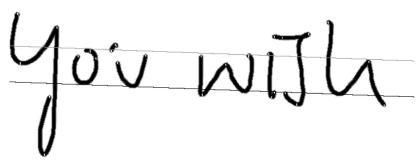


Fig. 5. Baseline and corpus line of an example part of a text line.

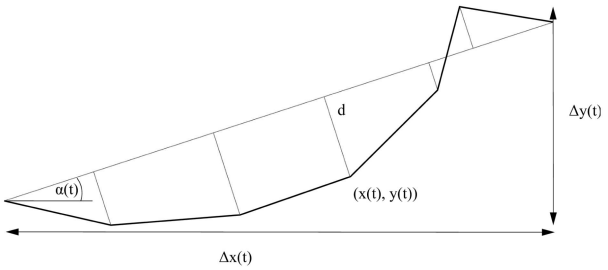


Fig. 6. Vicinity features of the point $(x(t), y(t))$. The three previous and three next points are considered in the example shown in this figure.

2.2 Feature Extraction

The input to the recognizer consists of 25 features for each (x, y)-coordinate recorded by the acquisition system. These features can be divided into two classes. The first class consists of features extracted for each point by considering its neighbors in the time series. The second class is based on the spatial information given by the offline matrix representation.

For point (x, y), the features in the first class are the following:

- a feature indicating whether the pen tip is touching the board or not,
- the hat feature indicating whether a delayed stroke was removed at y,
- the velocity computed before resampling,
- the x-coordinates after high-pass filtering, i.e., after subtracting a moving average from the true horizontal position,
- the y-coordinate after normalization,
- the cosine and sine of the angle between the line segment starting at the point and the x-axis (writing direction),
- the cosine and sine of the angle between the lines to the previous and the next point (curvature),
- The *vicinity aspect*, which is equal to the aspect of the trajectory (see Fig. 6):

$$\frac{\Delta y(t) - \Delta x(t)}{\Delta y(t) + \Delta x(t)},$$

- the cosine and sine of the angle α of the straight line from the first to the last vicinity point (see Fig. 6),
- the length of the trajectory in the vicinity divided by $\max(\Delta x(t), \Delta y(t))$, and
- the average squared distance d^2 of each point in the vicinity to the straight line from the first to the last vicinity point.

The features in the second class, illustrated in Fig. 7, are computed using a two-dimensional matrix $B = b_{i,j}$ representing the offline version of the data. For each position $b_{i,j}$, the number of points on the trajectory of the strokes is stored, providing a low-resolution image of the handwritten data. The following features are used:

- The number of points above the corpus line (ascenders) and below the baseline (descenders) in the vicinity of the current point.

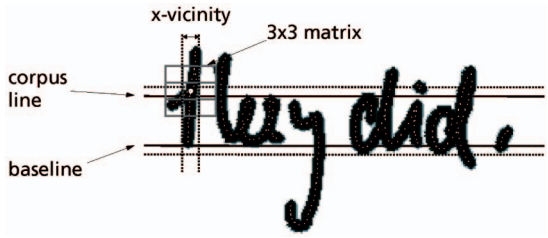


Fig. 7. Offline matrix features. The large white dot marks the considered point. The other online points are marked with smaller dots. The strokes have been widened for ease of visualization.

- The number of black points in each region of the context map (the two-dimensional vicinity of the current point is transformed to a 3×3 map with width and height set to the height of the corpus).

3 Offline Data Preparation

The offline data used in our experiments consists of gray-scale images scanned from handwritten forms, with a scanning resolution of 300 dpi and a gray-scale bit depth of eight. The following procedure was carried out to extract the text lines from the images. First, the image was rotated to account for the overall skew of the document, and the handwritten part was extracted from the form. Then, a histogram of the horizontal black/white transitions was calculated, and the text was split at the local minima to give a series of horizontal lines. Any stroke crossing the boundaries between two lines was assigned to the line containing their center of gravity. With this method, almost all text lines were extracted correctly.[2]

Once the line images were extracted, the next stage was to normalize the text with respect to writing skew and slant and character size.

3.1 Normalization

Unlike the online data, the normalizations for the offline data are applied to entire text lines at once. First of all, the image is rotated to account for the line skew. Then, the mean slant of the text is estimated, and a shearing transformation is applied to the image to bring the handwriting to an upright position. Next, the baseline and the corpus line are normalized. Normalization of the baseline means that the body of the text line (the part which is located between the upper and lower baselines), the ascender part (located above the upper baseline), and the descender part (below the lower baseline) are each scaled to a predefined height. Finally, the image is scaled horizontally so that the mean character width is approximately equal to a predefined size. Fig. 8 illustrates the offline preprocessing.

3.2 Feature Extraction

To extract the feature vectors from the normalized images, a sliding window approach is used. The width of the window is one pixel, and nine geometrical features are computed at each window position. Each text line image is therefore

2. Only about 1 percent of the text lines contain errors. These have been corrected manually in previous work [36].

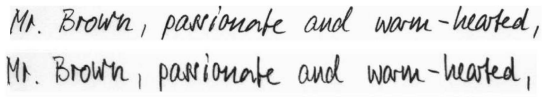


Fig. 8. Preprocessing of an image of handwritten text, showing (top) the original image and (bottom) the normalized image.

converted to a sequence of nine-dimensional vectors. The nine features are the following:

- the mean gray value of the pixels,
- the center of gravity of the pixels,
- the second-order vertical moment of the center of gravity,
- the positions of the uppermost and lowermost black pixels,
- the rate of change of these positions (with respect to the neighboring windows),
- the number of black-white transitions between the uppermost and lowermost pixels, and
- the proportion of black pixels between the uppermost and lowermost pixels.

For a more detailed description of the offline features, see [17].

4 Neural Network Recognizer

4.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a connectionist model containing a self-connected hidden layer. One benefit of the recurrent connection is that a "memory" of previous inputs remains in the network's internal state, allowing it to make use of past context. Context plays an important role in handwriting recognition, as illustrated in Fig. 9. Another important advantage of recurrency is that the rate of change of the internal state can be finely modulated by the recurrent weights, which builds in robustness to localized distortions of the input data.

4.2 Long Short-Term Memory (LSTM)

Unfortunately, the range of contextual information that standard RNNs can access is in practice quite limited. The problem is that the influence of a given input on the hidden layer and, therefore, on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections. This shortcoming (referred to in the literature as the *vanishing gradient problem* [37], [38]) makes it hard for an RNN to bridge gaps of more than about 10 time steps between relevant input and target events [37]. The

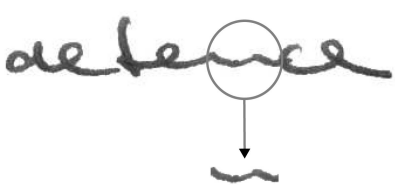


Fig. 9. Importance of context in handwriting recognition. The word "defence" is clearly legible, but the letter "n" in isolation is ambiguous.

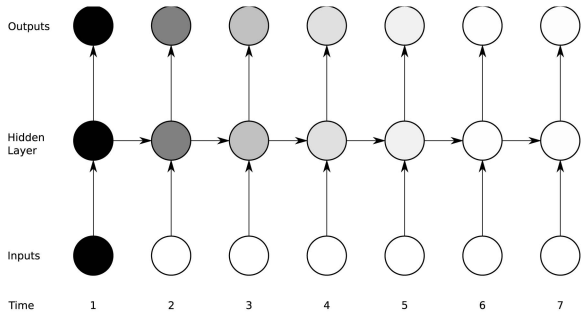


Fig. 10. Illustration of the vanishing gradient problem. The diagram represents a recurrent network unrolled in time. The units are shaded according to how sensitive they are to the input at time 1 (where black is high and white is low). As can be seen, the influence of the first input decays exponentially over time.

vanishing gradient problem is illustrated schematically in Fig. 10.

Long Short-Term Memory (LSTM) [39], [40] is an RNN architecture specifically designed to address the vanishing gradient problem. An LSTM hidden layer consists of recurrently connected subnets, called memory blocks. Each block contains a set of internal units or cells whose activation is controlled by three multiplicative gates: the input gate, forget gate, and output gate. Fig. 11 provides a detailed illustration of an LSTM memory block with a single cell.

The effect of the gates is to allow the cells to store and access information over long periods of time. For example, as long as the input gate remains closed (i.e., has an activation close to zero), the activation of the cell will not be overwritten by the new inputs arriving in the network. Similarly, the cell

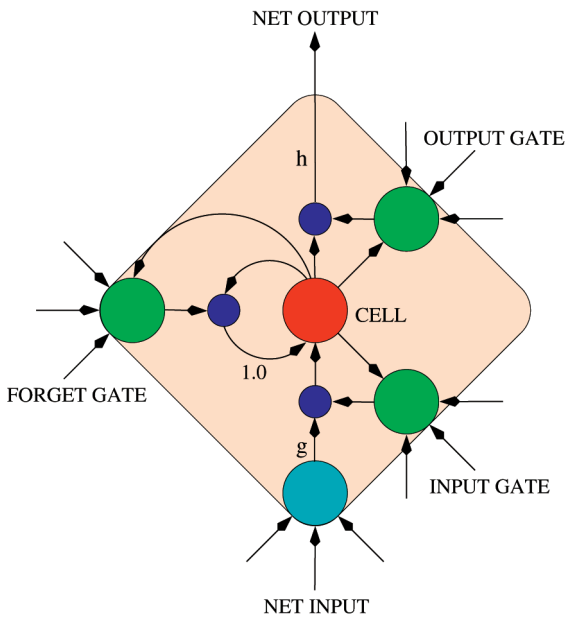


Fig. 11. LSTM memory block with one cell. The cell has a recurrent connection with fixed weight 1.0. The three gates collect input from the rest of the network and control the cell via multiplicative units (small circles). The input and output gates scale the input and output of the cell, while the forget gate scales the recurrent connection of the cell. The cell squashing functions (g and h) are applied at the indicated places. The internal connections from the cell to the gates are known as *peephole weights*.

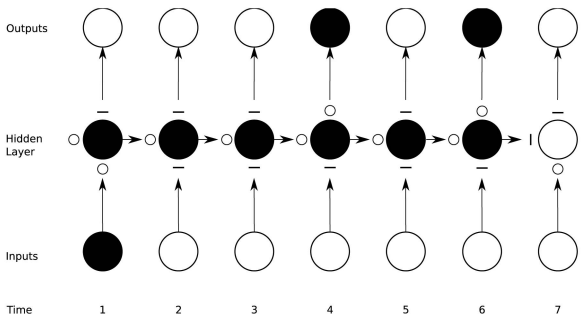


Fig. 12. Preservation of gradient information by LSTM. The diagram represents a network unrolled in time with a single hidden LSTM memory block. The input, forget, and output gate activations are, respectively, displayed below, to the left, and above the memory block. As in Fig. 10, the shading of the units corresponds to their sensitivity to the input at time 1. For simplicity, the gates are either entirely open ("O") or entirely closed ("—").

activation is only available to the rest of the network when the output gate is open and the cell's recurrent connection is switched on and off by the forget gate.

Fig. 12 illustrates how an LSTM block maintains gradient information over time. Note that the dependency is "carried" by the memory cell as long as the forget gate is open and the input gate is closed and that the output dependency can be switched on and off by the output gate, without affecting the hidden cell.

4.3 Bidirectional Recurrent Neural Networks

For many tasks, it is useful to have access to future, as well as past, context. In handwriting recognition, for example, the identification of a given letter is helped by knowing the letters both to the right and left of it.

Bidirectional RNNs (BRNNs) [41], [42] are able to access context in both directions along the input sequence. BRNNs contain two separate hidden layers, one of which processes the input sequence forward, while the other processes it backward. Both hidden layers are connected to the same output layer, providing it with access to the past and future context of every point in the sequence. BRNNs have outperformed standard RNNs in several sequence learning tasks, notably protein structure prediction [43] and speech processing [41], [44].

Combining BRNNs and LSTM gives BLSTM. BLSTM has previously outperformed other network architectures, including standard LSTM, BRNNs, and HMM-RNN hybrids, on phoneme recognition [33], [45].

4.4 Connectionist Temporal Classification (CTC)

Traditional RNN objective functions require a presegmented input sequence with a separate target for every segment. This has limited the applicability of RNNs in domains such as cursive handwriting recognition, where segmentation is difficult to determine. Moreover, because the outputs of such an RNN are a series of independent local classifications, some form of postprocessing is required to transform them into the desired label sequence.

CTC [31] is an RNN output layer specifically designed for sequence labeling tasks. It does not require the data to be presegmented, and it directly outputs a probability distribution over label sequences. CTC has been shown to outperform both HMMs and RNN-HMM hybrids on a phoneme recognition task [31]. CTC can be used for any RNN architecture, though as we will discuss in a moment, it is better suited to some than others.

A CTC output layer contains as many units as there are labels in the task plus an additional "blank" or "no label" unit. The output activations are normalized using the softmax function [46] so that they sum to one and are each in the range (0, 1):

$$y_k^t = \frac{e^{a_k^t}}{\sum_{k'} e^{a_{k'}^t}}, \tag{1}$$

where a_k^t is the unsquashed activation of output unit k at time t and y_k^t is the final output of the same unit.

The normalized outputs are used to estimate the conditional probabilities of observing label (or blank) k at time t in the input sequence $\mathbf{x}$, i.e., $y_k^t = p(k, t|\mathbf{x})$ (from now on, we will use a bold font to denote sequences). Note that each output is conditioned on the entire input sequence. For this reason, CTC is best used in conjunction with an architecture capable of incorporating long-range context in both input directions. One such architecture is BLSTM, as described in the previous section.

The conditional probability $p(\pi|\mathbf{x})$ of observing a particular *path* π through the lattice of label observations is found by multiplying together the label and blank probabilities at every time step:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} p(\pi_t, t|\mathbf{x}) = \prod_{t=1}^{T} y_{\pi_t}^t, \tag{2}$$

where π_t is the label observed at time t along path π.

Paths are mapped onto label sequences by an operator $\mathcal{B}$ that removes first the repeated labels and then the blanks. For example, both $\mathcal{B}(a, -, a, b, -)$ and $\mathcal{B}(-, a, a, -, -, a, b, b)$ yield the labeling (a,a,b). Since the paths are mutually exclusive, the conditional probability of some labeling $\mathbf{l}$ is the sum of the probabilities of all the paths mapped onto it by $\mathcal{B}$:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}). \tag{3}$$

The above step is what allows the network to be trained with unsegmented data. The intuition is that because we do not know where the labels within a particular transcription will occur, we sum over all the places where they *could* occur.

In general, a large number of paths will correspond to the same label sequence, so a naive calculation of (3) is unfeasible. However, it can be efficiently evaluated using a graph-based algorithm, similar to the forward-backward algorithm for HMMs [20].

4.5 CTC Forward-Backward Algorithm

To allow for blanks in the output paths, we consider modified label sequences $\mathbf{l}'$, with blanks added to the beginning and the end of $\mathbf{l}$ and inserted between every pair of consecutive labels. The length of $\mathbf{l}'$ is therefore $2|\mathbf{l}| + 1$. In calculating the probabilities of prefixes of $\mathbf{l}'$, we allow all transitions between blank and nonblank labels and also those between any pair of distinct nonblank labels.

For a labeling $\mathbf{l}$, define the *forward variable* α_s^t as the summed probability of all paths whose length t prefixes are mapped by $\mathcal{B}$ onto the length $s/2$ prefix of $\mathbf{l}$, i.e.,

$$\alpha_s^t = \sum_{\substack{\boldsymbol{\pi}: \\ \mathcal{B}(\pi_{1:t})=\mathbf{l}_{1:s/2}}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'}, \tag{4}$$

where, for some sequence $\mathbf{s}$, $\mathbf{s}_{a:b}$ is the subsequence $(\mathbf{s}_a, \mathbf{s}_{a+1}, \ldots, \mathbf{s}_{b-1}, \mathbf{s}_b)$ and $s/2$ is rounded down to an integer value. As we will see, α_s^t can be calculated recursively.

Allowing all paths to start with either a blank (b) or the first symbol in $\mathbf{l}$ (l_1), we get the following rules for initialization:

$$\begin{aligned} \alpha_1^1 &= y_b^1, \\ \alpha_2^1 &= y_{l_1}^1, \\ \alpha_s^1 &= 0, \ \forall s > 2, \end{aligned}$$

and recursion:

$$\alpha_s^t = y_{l'_s}^t \begin{cases} \sum_{i=s-1}^{s} \alpha_i^{t-1} & \text{if } l'_s = b \text{ or } l'_{s-2} = l'_s, \\ \sum_{i=s-2}^{s} \alpha_i^{t-1} & \text{otherwise.} \end{cases}$$

Note that $\alpha_s^t = 0 \ \forall s < |\mathbf{l}'| - 2(T-t) - 1$, because these variables correspond to states for which there are not enough time steps left to complete the sequence.

The *backward variables* β_s^t are defined as the summed probability of all paths whose suffixes starting at t map onto the suffix of $\mathbf{l}$ starting at label $s/2$:

$$\beta_s^t = \sum_{\substack{\boldsymbol{\pi}: \\ \mathcal{B}(\pi_{t:T})=\mathbf{l}_{s/2:|\mathbf{l}|}}} \prod_{t'=t+1}^{T} y_{\pi_{t'}}^{t'}. \tag{5}$$

The rules for initialization and recursion of the backward variables are given as follows:

$$\begin{aligned} \beta_{|\mathbf{l}'|}^T &= 1, \\ \beta_{|\mathbf{l}'|-1}^T &= 1, \\ \beta_s^T &= 0, \ \forall s < |\mathbf{l}'| - 1, \\ \beta_s^t &= \begin{cases} \sum_{i=s}^{s+1} \beta_i^{t+1} y_{l'_i}^{t+1} & \text{if } l'_s = b \text{ or } l'_{s+2} = l'_s, \\ \sum_{i=s}^{s+2} \beta_i^{t+1} y_{l'_i}^{t+1} & \text{otherwise.} \end{cases} \end{aligned}$$

Note that $\beta_s^t = 0 \ \forall s > 2t$, because these variables correspond to unreachable states.

Finally, the label sequence probability is given by the sum of the products of the forward and backward variables at any time:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \alpha_s^t \beta_s^t. \tag{6}$$

4.6 CTC Objective Function

The CTC objective function is defined as the negative log probability of the network correctly labeling the entire training set. Let S be a training set, consisting of pairs of input and target sequences $(\mathbf{x}, \mathbf{z})$. Then, the objective function O can be expressed as

$$O = -\sum_{(\mathbf{x},\mathbf{z}) \in S} \ln p(\mathbf{z}|\mathbf{x}). \tag{7}$$

The network can be trained with gradient descent by first differentiating O with respect to the outputs and then using backpropagation through time [47] to find the derivatives with respect to the network weights.

Noting that the same label (or blank) may be repeated several times for a single labeling $\mathbf{l}$, we define the set of positions where label k occurs as $lab(\mathbf{l}, k) = \{s : l'_s = k\}$, which may be empty. We then set $\mathbf{l} = \mathbf{z}$ and differentiate (6) with respect to the network outputs to obtain

$$\frac{\partial p(\mathbf{z}|\mathbf{x})}{\partial y_k^t} = \frac{1}{y_k^t} \sum_{s \in lab(\mathbf{z},k)} \alpha_s^t \beta_s^t. \tag{8}$$

Substituting this into (7) gives

$$\frac{\partial O}{\partial y_k^t} = -\frac{1}{p(\mathbf{z}|\mathbf{x}) y_k^t} \sum_{s \in lab(\mathbf{z},k)} \alpha_s^t \beta_s^t. \tag{9}$$

To backpropagate the gradient through the output layer, we need the objective function derivatives with respect to the outputs a_k^t before the activation function is applied. For the softmax function,

$$\frac{\partial y_{k'}^t}{\partial a_k^t} = y_{k'}^t \delta_{kk'} - y_{k'}^t y_k^t, \tag{10}$$

and, therefore,

$$\frac{\partial O}{\partial u_k^t} = y_k^t - \frac{1}{p(\mathbf{z}|\mathbf{x})} \sum_{s \in lab(\mathbf{z},k)} \alpha_s^t \beta_s^t. \tag{11}$$

4.7 CTC Decoding

Once the network is trained, we would ideally transcribe some unknown input sequence $\mathbf{x}$ by choosing the labeling $\mathbf{l}^*$ with the highest conditional probability:

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{x}). \tag{12}$$

Using the terminology of HMMs, we refer to the task of finding this labeling as *decoding*. Unfortunately, we do not know of a tractable decoding algorithm that is guaranteed to give optimal results. However, a simple and effective approximation is given by assuming that the most probable path corresponds to the most probable labeling:

$$\mathbf{l}^* \approx \mathcal{B}\left(\arg\max_{\pi} p(\pi|\mathbf{x})\right). \tag{13}$$

This is trivial to compute since the most probable path is just the concatenation of the most active outputs at every time step.

For some tasks, we want to constrain the output labelings according to a grammar. For example, in continuous speech and handwriting recognition, the final transcriptions are usually required to form sequences of dictionary words. In addition, it is common practice to use a language model to weight the probabilities of particular sequences of words.

We can express these constraints by altering the label sequence probabilities in (12) to be conditioned on some probabilistic grammar G, as well as the input sequence $\mathbf{x}$:

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{x}, G). \tag{14}$$

Note that absolute requirements, for example, that $\mathbf{l}$ contains only dictionary words, can be incorporated by setting the probability of all sequences that fail to meet them to zero. Applying the basic rules of probability, we obtain

$$p(\mathbf{l}|\mathbf{x}, G) = \frac{p(\mathbf{l}|\mathbf{x})p(\mathbf{l}|G)p(\mathbf{x})}{p(\mathbf{x}|G)p(\mathbf{l})}, \tag{15}$$

where we have used the fact that $\mathbf{x}$ is conditionally independent of G given $\mathbf{l}$. If we assume that $\mathbf{x}$ is independent of G, (15) reduces to

$$p(\mathbf{l}|\mathbf{x}, G) = \frac{p(\mathbf{l}|\mathbf{x})p(\mathbf{l}|G)}{p(\mathbf{l})}. \tag{16}$$

Note that this assumption is, in general, false since both the input sequences and the grammar depend on the underlying generator of the data, for example, the language being spoken. However, it is a reasonable first approximation and is particularly justifiable in cases where the grammar is created using data other than that from which $\mathbf{x}$ was drawn (as is common practice in speech and handwriting recognition, where separate textual corpora are used to generate language models).

If we further assume that, prior to any knowledge about the input or the grammar, all label sequences are equally probable, (14) reduces to

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} p(\mathbf{l}|\mathbf{x})p(\mathbf{l}|G). \tag{17}$$

Note that, since the number of possible label sequences is finite (because both L and $|\mathbf{l}|$ are finite), assigning equal prior probabilities does not lead to an improper prior.

4.8 CTC Token Passing Algorithm

We now describe an algorithm, based on the *token passing algorithm* for HMMs [48], that allows us to find an approximate solution to (17) for a simple grammar.

Let G consist of a dictionary D containing W words and a set of W^2 bigrams $p(w|\hat{w})$ that define the probability of making a transition from word $\hat{w}$ to word w. The probability of any label sequence that does not form a sequence of dictionary words is zero.

For each word w, define the modified word w' as w with blanks added at the beginning and end and between each pair of labels. Therefore, $|w'| = 2|w| + 1$. Define a token $tok = (s, h)$ to be a pair consisting of a real-valued score s and a history h of previously visited words. In fact, each token corresponds to a particular path through the network outputs, and the token score is the log probability of that path. The transition probabilities are used when a token is passed from the last character in one word to the first character in another. The output word sequence is then given by the history of the highest scoring end-of-word token at the final time step.

At every time step t of the length-T output sequence, each character c of each modified word w' holds a single token $tok(w, c, t)$. This is the highest scoring token reaching that segment at that time. In addition, we define the *input token* $tok(w, 0, t)$ to be the highest scoring token arriving at word w at time t and the *output token* $tok(w, -1, t)$ to be the highest scoring token leaving word w at time t.

The pseudocode is provided in Algorithm 1.

Algorithm 1. CTC token passing algorithm

1: **Initialization:**
2: **for** all words $w \in D$ **do**
3: $tok(w, 1, 1) = (\ln y^1_b, (w))$
4: $tok(w, 2, 1) = (\ln y^1_{w_1}, (w))$
5: **if** $|w| = 1$ **then**
6: $tok(w, -1, 1) = tok(w, 2, 1)$
7: **else**
8: $tok(w, -1, 1) = (-\infty, ())$
9: $tok(w, c, 1) = (-\infty, ())$ for all other l
10:
11: **Algorithm:**
12: **for** $t = 2$ to T **do**
13: sort output tokens $tok(w, -1, t-1)$ by rising score
14: **for** all words $w \in D$ **do**
15: $w^* = \arg\max_{\hat{w}}[tok(\hat{w}, -1, t-1).s + \ln p(w|\hat{w})]$
16: $tok(w, 0, t).s = tok(w^*, -1, t-1).s + \ln p(w|w^*)$
17: $tok(w, 0, t).h = tok(w^*, -1, t-1).h + w$
18: **for** $c = 1$ to $|w'|$ **do**
19: $P = \{tok(w, c, t-1), tok(w, c-1, t-1)\}$
20: **if** $w'_c \neq blank$ and $c > 2$ and $w'_{c-2} \neq w'_c$ **then**
21: add $tok(w, c-2, t-1)$ to P
22: $tok(w, c, t) =$ token in P with highest score
23: $tok(w, c, t).s \mathrel{+}= \ln y^t_{w'_c}$
24: $R = \{tok(w, |w'|, t), tok(w, |w'|-1, t)\}$
25: $tok(w, -1, t) =$ token in R with highest score
26:
27: **Termination:**
28: $w^* = \arg\max_w tok(w, -1, T).s$
29: output $tok(w^*, -1, T).h$

The CTC token passing algorithm has a worst-case complexity of $O(TW^2)$ since line 15 requires a potential search through all W words. However, because the output tokens $tok(w, -1, T)$ are sorted in order of score, the search can be terminated when a token is reached whose score is less than the current best score with the transition included. The typical complexity is therefore considerably lower, with a lower bound of $O(TWlogW)$ to account for the sort. If no bigrams are used, lines 15-17 can be replaced by a simple search for the highest scoring output token and the complexity reduces to $O(TW)$.

5 Experiments and Results

The aim of our experiments was to evaluate the complete RNN handwriting recognition system, illustrated in Fig. 13, on both online and offline handwriting. In particular, we wanted to see how it compared to a state-of-the-art HMM-based system. The online and offline databases used were the IAM-OnDB and the IAM-DB, respectively. Note that these do not correspond to the same handwriting samples: the IAM-OnDB was acquired from a whiteboard (see Section 2), while the IAM-DB consists of scanned images of handwritten forms (see Section 3).

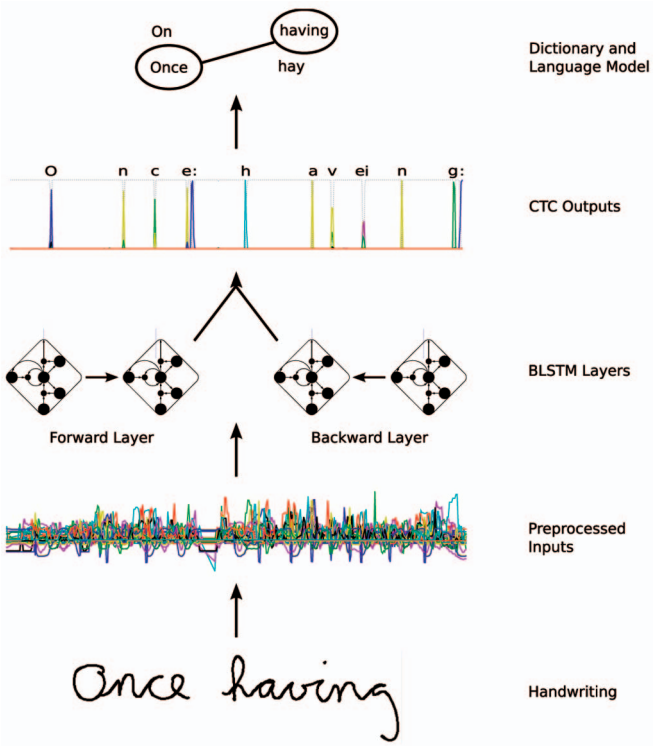


Fig. 13. The complete RNN handwriting recognition system. First, the online or offline handwriting data is preprocessed with the techniques described in Sections 2 and 3. The resulting sequence of feature vectors is scanned in opposite directions by the forward and backward BLSTM hidden layers. The BLSTM layers feed forward to the CTC output layer, which produces a probability distribution over character transcriptions. This distribution is passed to the dictionary and language model, using the token passing algorithm, to obtain the final word sequence.

To make the comparisons fair, the same online and offline preprocessing was used for both the HMM and RNN systems. In addition, the same dictionaries and language models were used for the two systems—see Section 5.2 for further details. As well as the main comparisons, extra experiments were carried out on the online database to determine the effect of varying the dictionary size and of disabling the forward and backward hidden layers in the RNN.

For all of the experiments, the task was to transcribe the text lines in the test set using the words in the dictionary. The basic performance measure was the *word accuracy*:

$$acc = 100 * \left(1 - \frac{\text{insertions} + \text{substitutions} + \text{deletions}}{\text{total length of test set transcriptions}}\right),$$

where the number of word insertions, substitutions, and deletions is summed over the whole test set. For the RNN system, we also recorded the *character accuracy*, defined as above except with characters instead of words.

5.1 Data Sets

For the online experiments we used the IAM-OnDB, a database acquired from a "smart" whiteboard [49]. The database was divided into four disjoint sets, as defined by the IAM-OnDB-t2 benchmark task: a training set containing 5,364 lines; a first validation set containing 1,438 lines; a second validation set containing 1,518 lines, which can be used, for example, to optimize a language model; and a test set containing 3,859 lines. After preprocessing, the input data consisted of 25 inputs per time step.

For the offline experiments, we used the IAM-DB, a database acquired from scanned images of handwritten forms [36]. The IAM-DB consists of 13,040 fully transcribed handwritten lines, containing 86,272 instances of 11,050 distinct words. The division into data sets was defined by the benchmark "large writer-independent text line recognition task."[3] The training set contains 6,161 lines from 283 writers, the validation set contains 920 lines from 56 writers, and the test set contains 2,781 lines from 161 writers. After preprocessing, the input data consisted of nine inputs per time step.

Both databases were created using texts from the Lancaster-Oslo/Bergen (LOB) corpus [50] as prompts. Note, however, that the online and offline prompts were not the same.

Both the online and offline transcriptions contain 81 separate characters, including all lowercase and capital letters, as well as various other special characters, e.g., punctuation marks, digits, a character for garbage symbols, and a character for the space. Note that, for the RNN systems, only 80 of these were used since the garbage symbol is not required for CTC.

5.2 Language Model and Dictionaries

Dictionaries were used for all the experiments where word accuracy was recorded. All dictionaries were derived from three different text corpora, the LOB (excluding the data used as prompts), the Brown corpus [52], and the Wellington corpus [53]. The "standard" dictionary we used for our main results consisted of the 20,000 most frequently occurring words in the three corpora. The figure 20,000 was chosen because it had been previously shown to give best results for HMMs [51]. Note that this dictionary was "open," in the sense that it did not contain all the words in either the online or offline test set. To analyze the dependency of the recognition scores on the lexicon, we carried out extra experiments on the online data using open dictionaries with between 5,000 and 30,000 words (also chosen from the most common words in LOB). In addition, we tested both systems with two dictionaries that were "closed" with respect to the online data (i.e., that contained every word in the online test set).

The bigram language model, used for some of the experiments, was based on the same corpora as the language model. It was then optimized for the online and offline experiments separately, using data from the validation sets. Note that, for the RNN system, the effect of the language model is to directly multiply the existing word sequence probabilities by the combined transition probabilities given by the bigrams. This contrasts with HMMs, where the language model weighting factor must be found empirically because HMMs are known to underestimate acoustic scores [54].

3. Both databases are available for public download, along with the corresponding task definitions, at http://www.iam.unibe.ch/fki/databases/iam-handwriting-database and http://www.iam.unibe.ch/fki/databases/iam-on-line-handwriting-database.

TABLE 1
Main Result for Online Data

System	Word Accuracy	Char. Accuracy
HMM	65.0%	—
CTC	79.7 ± 0.3%	88.5 ± 0.05%

TABLE 2
Main Result for Offline Data

System	Word Accuracy	Char. Accuracy
HMM	64.5%	—
RNN	74.1 ± 0.8%	81.8 ± 0.6%

5.3 HMM Parameters

The HMM system used for the online and offline experiments was similar to that described in [35]. One linear HMM was used for each character. For the online data, every character model contained eight states, while, for the offline data, the number of states was chosen individually for each character [51]. The observation probabilities were modeled by mixtures of diagonal Gaussians. Thirty-two Gaussians were used for online data and 12 for the offline data. In both cases, the number of Gaussians was optimized by incrementally splitting the Gaussian component with the highest weight. The language model weighting factor and the word insertion penalty were determined empirically on the validation set.

5.4 RNN Parameters

The RNN had a BLSTM architecture with a CTC output layer (see Section 4 for details). The forward and backward hidden layers each contained 100 LSTM memory blocks. Each memory block contains a memory cell, an input gate, an output gate, a forget gate, and three peephole connections. Hyperbolic tangent was used for the block input and output activation functions, and the gate activation function was the logistic sigmoid. The CTC output layer had 81 nodes (one for each character occurring in the training set and one extra for "blank"). The size of the input layer was determined by the data: For the online data, there were 25 inputs and, for the offline data, there were nine. Otherwise, the network was identical for the two tasks. The input layer was fully connected to both hidden layers, and these were fully connected to themselves and to the output layer. This gave 117,681 weights for the online data and 105,082 weights for the online data.

The network weights were initialized with a Gaussian distribution of mean zero and standard deviation 0.1. The network was trained using online gradient descent with a learning rate of 0.0001 and a momentum of 0.9. The error rate was recorded every five epochs on the validation set, and training was stopped when performance had ceased to improve on the validation set for 50 epochs. Because of the random initialization, all RNN experiments were repeated four times, and the results are stated as the mean ± the standard error.

5.5 Main Results

As can be seen in Tables 1 and 2, the RNN substantially outperformed the HMM on both databases. To put these results in perspective, the Microsoft tablet PC handwriting recognizer [55] gave a word accuracy score of 71.32 percent on the online test set. This result is not directly comparable to our own, since the Microsoft system was trained on a different training set and uses considerably more sophisticated language modeling than the systems we implemented. However, it suggests that our recognizer is competitive with the best commercial systems for unconstrained handwriting.

5.6 Influence of Dictionary Size

We carried out two sets of experiments on the online database to gauge the effect of varying the dictionary size. In one, we generated open dictionaries with between 5,000 and 30,000 words by taking the n most common words in the LOB, Brown, and Wellington corpora. In the other, we took a closed dictionary containing the 5,597 words in the online test set and measured the change in performance when this was padded to 20,000 words. Note that the language model was not used for the RNN results in the section, which would be substantially higher otherwise (for example, the accuracy for the 20,000-word open lexicon was 79.7 percent with the language model and 74.0 percent without).

The results for the first set of experiments are shown in Table 3 and plotted in Fig. 14. In all cases, the RNN system significantly outperforms the HMM system, despite the lack of language model. Both RNN and HMM performance increased with size (and test set coverage) up to 25,000 words. Note, however, that the RNN is less affected by the dictionary size and that, for the 30,000-word dictionary, performance continues to increase for the RNN but drops for the HMM. The tendency of HMMs to lose accuracy for very large handwriting lexicons has been previously observed [51].

The results of the second set of experiments are shown in Table 4. Unsurprisingly, the closed dictionary containing only the test set words gave the best results for both systems. The scores with the 20,000-word closed dictionary were somewhat lower in both cases, due to the increased perplexity, but still better than any recorded with open dictionaries.

In summary, the RNN retained its advantage over the HMM, regardless of the dictionary used. Moreover, the differences tended to be larger when the dictionary had higher perplexity. We believe this is because the RNN is better at recognizing characters and is therefore less dependent on a dictionary or language model to constrain its outputs.

TABLE 3
Online Word Accuracy with Open Dictionaries

Dictionary Size	Coverage(%)	RNN(%)	HMM(%)
5,000	86.1	66.2 ± 0.4	47.7
10,000	90.7	70.3 ± 0.4	54.8
15,000	92.9	72.5 ± 0.3	60.6
20,000	94.4	74.0 ± 0.3	63.7
25,000	95.1	74.6 ± 0.3	65.0
30,000	95.7	75.0 ± 0.3	61.5

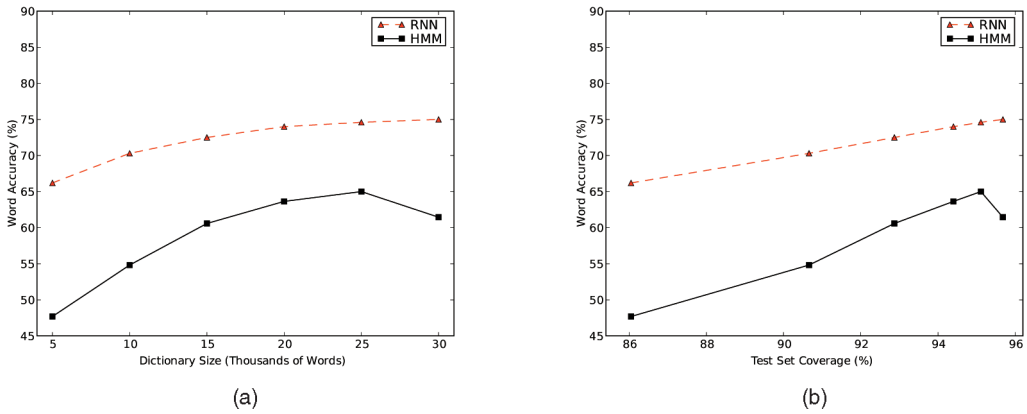


Fig. 14. HMM and RNN word accuracy plotted against (a) dictionary size and (b) test set coverage.

5.7 Influence of RNN Hidden Layers

To test the relative importance of the forward and backward hidden layers, we evaluated the RNN system on the online database with each of the hidden layers disabled in turn. We give the character error rate only, since this is the clearest indicator of network performance. The results are displayed in Table 5. It is interesting to note that significantly higher accuracy was achieved with a reverse layer only than with a forward layer only (all differences are significant using a standard z-test with $\alpha < 0.001$). This suggests that the right-to-left dependencies are more important to the network than the left-to-right ones.

5.8 Learning Curve

Fig. 15 shows the decrease in training and validation error over time for a typical RNN training run on the online database.

5.9 Use of Context by the RNN

We have previously asserted that the BLSTM architecture is able to access long-range bidirectional context. Evidence of this can be found by analyzing the partial derivatives of the network outputs at a particular time t in a sequence with respect to the inputs at all times t'. We refer to the matrix of these derivatives as the *sequential Jacobian*. The larger the values of the sequential Jacobian for some t and t', the more sensitive the network output at time t is to the input at time t'.

Fig. 16 plots the sequential Jacobian for a single output during the transcription of a line from the online database. As can be seen, the network output is sensitive to information from about the first 120 time steps of the sequence, which corresponds roughly to the length of the first word. Moreover, this area of sensitivity extends in both directions from the point where the prediction is made.

6 Discussion

Our experiments reveal a substantial gap in performance between the HMM and RNN systems, with a relative error reduction of more than 40 percent in some cases. In what follows, we discuss the differences between the two systems and suggest reasons for the RNN's superiority.

First, standard HMMs are generative, while an RNN trained with a discriminative objective function (such as CTC) is discriminative. That is, HMMs attempts to model the conditional probability of the input data given the internal state sequence and then use this to find the most probable label sequence, while the RNN directly models the probability of the label sequence given the inputs. The advantages of the generative approach include the possibility of adding extra models to an already trained system and being able to generate synthetic data. However, for discriminative tasks, determining the input distribution is unnecessary. Additionally, for tasks such as handwriting recognition where the prior data distribution is hard to determine, generative approaches can only provide unnormalized likelihoods for the label sequences. Discriminative approaches, on the other hand, yield normalized label probabilities, which can be used to assess prediction confidence or to combine the outputs of several classifiers. In most cases, discriminative methods achieve better performance than generative methods on classification tasks.

The second difference is that RNNs provide more flexible models of the input features than the mixtures of diagonal Gaussians used in standard HMMs. Gaussians with diagonal covariance matrices are limited to modeling distributions over independent variables. This assumes that the input features are decorrelated, which can be difficult to ensure for real-world tasks such as handwriting recognition. RNNs, on the other hand, do not assume that the features come from a

TABLE 4
Online Word Accuracy with Closed Dictionaries

Dictionary Size	RNN(%)	HMM(%)
5,597	85.3 ± 0.3	70.1
20,000	81.5 ± 0.4	68.8

TABLE 5
Online Character Accuracy with Different RNN Hidden Layers

Architecture	Character Accuracy	Training Iterations
Forward Only	81.3 ± 0.3%	182.5 ± 24.3
Reverse Only	85.8 ± 0.3%	228.8 ± 51.7
Bidirectional	88.5 ± 0.05%	41.25 ± 2.6

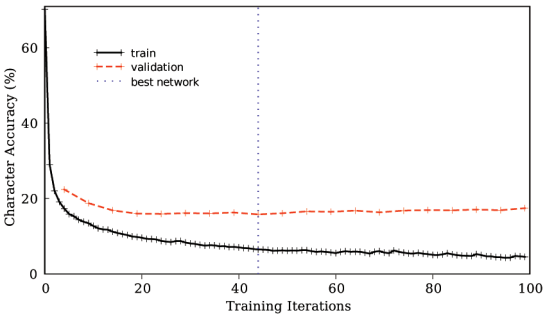


Fig. 15. RNN character error rate during training. The performance ceased to improve on the validation set after 45 passes through the training set (marked with "best network").

particular distribution or that they are independent and can model nonlinear relationships among features. However, it should be noted that RNNs typically perform better using input features with simpler relationships.

The third difference is that the internal states of a standard HMM are discrete and univariate. This means that, for an HMM with n states, only $O(logn)$ bits of information about the past observation sequence are carried by the internal state. RNNs, on the other hand, have a continuous multivariate internal state (the hidden layer) whose information capacity grows linearly with its size.

The fourth difference is that HMMs are constrained to segment the input into a sequence of states or units. This is often problematic for continuous input sequences, since the precise boundary between units can be ambiguous. A further problem with segmentation is that, at least with standard Markovian transition probabilities, the probability of remaining in a particular state decreases exponentially with time. Exponential decay is in general a poor model of state duration, and various measures have been suggested to alleviate this [56]. However, an RNN trained with CTC does not need to segment the input sequence and therefore avoids both of these problems.

The final and perhaps most crucial difference is that, unlike RNNs, HMMs assume that the probability of each observation depends only on the current state. One consequence of this is that data consisting of continuous trajectories (such as the sequence of pen coordinates for online handwriting and the sequence of window positions in offline handwriting) are difficult to model with standard HMMs since each observation is heavily dependent on those around it. Similarly, data with long-range contextual dependencies is troublesome, because individual sequence elements are influenced by the elements surrounding them. The latter problem can be mitigated by adding extra models to account for each sequence element in all different contexts (e.g., using triphones instead of phonemes for speech recognition). However, increasing the number of models exponentially increases the number of parameters that must be inferred, which, in turn, increases the amount of data required to reliably train the system. For RNNs, on the other hand, modeling continuous trajectories is natural, since their own hidden state is itself a continuous trajectory. Furthermore, the range of contextual information accessible to an RNN is limited only by the choice of architecture and, in the case of BLSTM, can in principle extend to the entire input sequence.

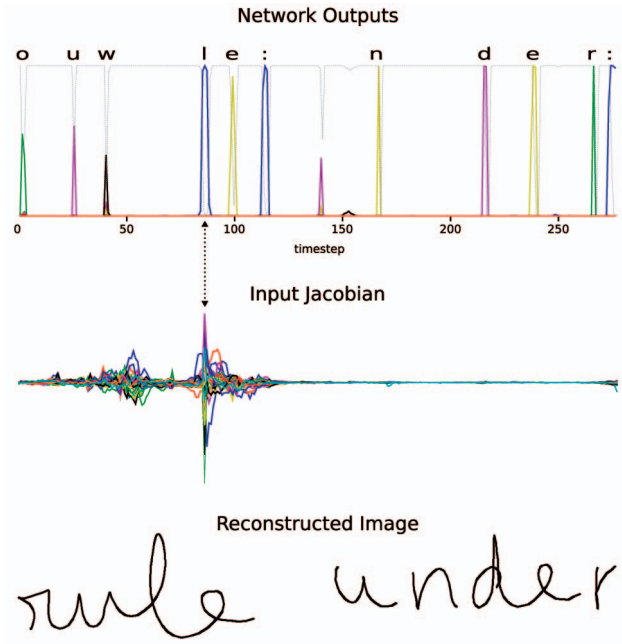


Fig. 16. Sequential Jacobian for a sequence from the IAM-OnDB. The Jacobian is evaluated for the output corresponding to the label "l" at the time step when "l" is emitted (indicated by the vertical dotted line).

In summary, the observed difference in performance between RNNs and HMM in unconstrained handwriting recognition can be explained by the fact that as researchers approach handwriting recognition tasks of increasing complexity, the assumptions HMMs are based on lose validity. For example, unconstrained handwriting or spontaneous speech are more dynamic and show more marked context effects than hand-printed scripts or read speech.

7 Conclusions

We have introduced a novel approach for recognizing unconstrained handwritten text, using an RNN. The key features of the network are the BLSTM architecture, which provides access to long-range bidirectional contextual information, and the CTC output layer, which allows the network to be trained on unsegmented sequence data. In experiments on online and offline handwriting data, the new approach outperformed a state-of-the-art HMM-based system and also proved more robust to changes in dictionary size.

Acknowledgments

This work was supported by the Swiss National Science Foundation Program "Interactive Multimodal Information Management (IM2)" in the Individual Project "Visual/Video Processing" along with SNF Grants 200021-111968/1 and 200020-19124/1. The authors thank Dr. Matthias Zimmermann for providing the statistical language model and James A. Pitman for giving permission to publish the Microsoft results.

REFERENCES

[1] R. Seiler, M. Schenkel, and F. Eggimann, "Off-Line Cursive Handwriting Recognition Compared with On-Line Recognition," *Proc. 13th Int'l Conf. Pattern Recognition,* vol. 4, p. 505, 1996.

[2] C. Tappert, C. Suen, and T. Wakahara, "The State of the Art in Online Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 12, no. 8, pp. 787-808, Aug. 1990.

[3] R. Plamondon and S.N. Srihari, "On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 1, pp. 63-84, Jan. 2000.

[4] A. Vinciarelli, "A Survey on Off-Line Cursive Script Recognition," *Pattern Recognition,* vol. 35, no. 7, pp. 1433-1446, 2002.

[5] H. Bunke, "Recognition of Cursive Roman Handwriting—Past Present and Future," *Proc. Seventh Int'l Conf. Document Analysis and Recognition,* vol. 1, pp. 448-459, 2003.

[6] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "Unipen Project of On-Line Data Exchange and Recognizer Benchmarks," *Proc. 12th Int'l Conf. Pattern Recognition,* pp. 29-33, 1994.

[7] J. Hu, S. Lim, and M. Brown, "Writer Independent On-Line Handwriting Recognition Using an HMM Approach," *Pattern Recognition,* vol. 33, no. 1, pp. 133-147, Jan. 2000.

[8] C. Bahlmann and H. Burkhardt, "The Writer Independent Online Handwriting Recognition System Frog on Hand and Cluster Generative Statistical Dynamic Time Warping," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 26, no. 3, pp. 299-310, Mar. 2004.

[9] C. Bahlmann, B. Haasdonk, and H. Burkhardt, "Online Handwriting Recognition with Support Vector Machines—A Kernel Approach," *Proc. Eighth Int'l Workshop Frontiers in Handwriting Recognition,* pp. 49-54, 2002.

[10] G. Wilfong, F. Sinden, and L. Ruedisueli, "On-Line Recognition of Handwritten Symbols," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 9, pp. 935-940, Sept. 1996.

[11] K.M. Sayre, "Machine Recognition of Handwritten Words: A Project Report," *Pattern Recognition,* vol. 5, no. 3, pp. 213-228, 1973.

[12] L. Schomaker, "Using Stroke- or Character-Based Self-Organizing Maps in the Recognition of On-Line, Connected Cursive Script," *Pattern Recognition,* vol. 26, no. 3, pp. 443-450, 1993.

[13] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis, "An Unconstrained Handwriting Recognition System," *Int'l J. Document Analysis and Recognition,* vol. 4, no. 4, pp. 226-242, 2002.

[14] S. Bercu and G. Lorette, "On-Line Handwritten Word Recognition: An Approach Based on Hidden Markov Models," *Proc. Third Int'l Workshop Frontiers in Handwriting Recognition,* pp. 385-390, 1993.

[15] T. Starner, J. Makhoul, R. Schwartz, and G. Chou, "Online Cursive Handwriting Recognition Using Speech Recognition Techniques," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing,* vol. 5, pp. 125-128, 1994.

[16] J. Hu, M. Brown, and W. Turin, "HMM Based Online Handwriting Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, no. 10, pp. 1039-1045, Oct. 1996.

[17] U.-V. Marti and H. Bunke, "Using a Statistical Language Model to Improve the Performance of an HMM-Based Cursive Handwriting Recognition System," *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 15, pp. 65-90, 2001.

[18] M. Schenkel, I. Guyon, and D. Henderson, "On-Line Cursive Script Recognition Using Time Delay Neural Networks and Hidden Markov Models," *Machine Vision and Applications,* vol. 8, pp. 215-223, 1995.

[19] A. El-Yacoubi, M. Gilloux, R. Sabourin, and C. Suen, "An HMM-Based Approach for Off-Line Unconstrained Handwritten Word Modeling and Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 21, no. 8, pp. 752-760, Aug. 1999.

[20] L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE,* vol. 77, no. 2, pp. 257-286, 1989.

[21] N.G. Bourbakis, "Handwriting Recognition Using a Reduced Character Method and Neural Nets," *Proc. SPIE Nonlinear Image Processing VI,* vol. 2424, pp. 592-601, 1995.

[22] H. Bourlard and N. Morgan, *Connnectionist Speech Recognition: A Hybrid Approach.* Kluwer Academic Publishers, 1994.

[23] Y. Bengio, "Markovian Models for Sequential Data," *Neural Computing Surveys,* vol. 2, pp. 129-162, Nov. 1999.

[24] A. Brakensiek, A. Kosmala, D. Willett, W. Wang, and G. Rigoll, "Performance Evaluation of a New Hybrid Modeling Technique for Handwriting Recognition Using Identical On-Line and Off-Line Data," *Proc. Fifth Int'l Conf. Document Analysis and Recognition,* pp. 446-449, 1999.

[25] S. Marukatat, T. Artires, B. Dorizzi, and P. Gallinari, "Sentence Recognition through Hybrid Neuro-Markovian Modelling," *Proc. Sixth Int'l Conf. Document Analysis and Recognition,* pp. 731-735, 2001.

[26] S. Jaeger, S. Manke, J. Reichert, and A. Waibel, "Online Handwriting Recognition: The NPen+ Recognizer," *Int'l J. Document Analysis and Recognition,* vol. 3, no. 3, pp. 169-180, 2001.

[27] E. Caillault, C. Viard-Gaudin, and A.R. Ahmad, "MS-TDNN with Global Discriminant Trainings," *Proc. Eighth Int'l Conf. Document Analysis and Recognition,* pp. 856-861, 2005.

[28] A.W. Senior and F. Fallside, "An Off-Line Cursive Script Recognition System Using Recurrent Error Propagation Networks," *Proc. Third Int'l Workshop Frontiers in Handwriting Recognition,* pp. 132-141, 1993.

[29] A.W. Senior and A.J. Robinson, "An Off-Line Cursive Handwriting Recognition System," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 3, pp. 309-321, Mar. 1998.

[30] J. Schenk and G. Rigoll, "Novel Hybrid NN/HMM Modelling Techniques for On-Line Handwriting Recognition," *Proc. 10th Int'l Workshop Frontiers in Handwriting Recognition,* pp. 619-623, 2006.

[31] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," *Proc. 23rd Int'l Conf. Machine Learning,* pp. 369-376, 2006.

[32] A. Graves, S. Fernandez, M. Liwicki, H. Bunke, and J. Schmidhuber, "Unconstrained Online Handwriting Recognition with Recurrent Neural Networks," *Advances in Neural Information Processing Systems 20,* J. Platt, D. Koller, Y. Singer, and S. Roweis, eds., 2008.

[33] A. Graves and J. Schmidhuber, "Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures," *Neural Networks,* vol. 18, nos. 5-6, pp. 602-610, 2005.

[34] D. Moore, "The IDIAP Smart Meeting Room," technical report, IDIAP-Com, 2002.

[35] M. Liwicki and H. Bunke, "HMM-Based On-Line Recognition of Handwritten Whiteboard Notes," *Proc. 10th Int'l Workshop Frontiers in Handwriting Recognition,* pp. 595-599, 2006.

[36] U.-V. Marti and H. Bunke, "The IAM-Database: An English Sentence Database for Offline Handwriting Recognition," *Int'l J. Document Analysis and Recognition,* vol. 5, pp. 39-46, 2002.

[37] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies," *A Field Guide to Dynamical Recurrent Neural Networks,* S.C. Kremer and J.F. Kolen, eds., 2001.

[38] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent Is Difficult," *IEEE Trans. Neural Networks,* vol. 5, no. 2, pp. 157-166, Mar. 1994.

[39] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[40] F. Gers, N. Schraudolph, and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *J. Machine Learning Research,* vol. 3, pp. 115-143, 2002.

[41] M. Schuster and K.K. Paliwal, "Bidirectional Recurrent Neural Networks," *IEEE Trans. Signal Processing,* vol. 45, pp. 2673-2681, Nov. 1997.

[42] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, "Exploiting the Past and the Future in Protein Secondary Structure Prediction," *Bioinformatics,* vol. 15, 1999.

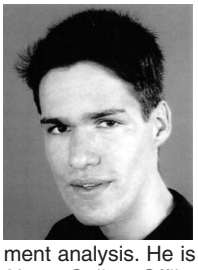
[43] P. Baldi, S. Brunak, P. Frasconi, G. Pollastri, and G. Soda, "Bidirectional Dynamics for Protein Secondary Structure Prediction," *Lecture Notes in Computer Science,* vol. 1828, pp. 80-104, 2001.

[44] T. Fukada, M. Schuster, and Y. Sagisaka, "Phoneme Boundary Estimation Using Bidirectional Recurrent Neural Networks and Its Applications," *Systems and Computers in Japan,* vol. 30, no. 4, pp. 20-30, 1999.

[45] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition," *Proc. Int'l Conf. Artificial Neural Networks,* pp. 799-804, 2005.

[46] J. Bridle, "Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition," *Neurocomputing: Algorithms, Architectures and Applications,* F. Soulie and J. Herault, eds., pp. 227-236, 1990.
[47] R.J. Williams and D. Zipser, "Gradient-Based Learning Algorithms for Recurrent Connectionist Networks," *Backpropagation: Theory, Architectures, and Applications,* Y. Chauvin and D.E. Rumelhart, eds., 1990.
[48] S. Young, N. Russell, and J. Thornton, "Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems," technical report, Eng. Dept., Cambridge Univ., 1989.
[49] M. Liwicki and H. Bunke, "IAM-OnDB—An On-Line English Sentence Database Acquired from Handwritten Text on a Whiteboard," *Proc. Eighth Int'l Conf. Document Analysis and Recognition,* vol. 2, pp. 956-961, 2005.
[50] S. Johansson, *The Tagged LOB Corpus: User's Manual.* Norwegian Computing Centre for the Humanities, 1986.
[51] M. Zimmermann, J.-C. Chappelier, and H. Bunke, "Offline Grammar-Based Recognition of Handwritten Sentences," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 28, no. 5, pp. 818-821, May 2006.
[52] W.N. Francis and H. Kucera, *Brown Corpus Manual, Manual of Information to Accompany a Standard Corpus of Present-Day Edited American English, for Use with Digital Computers.* Dept. of Linguistics, Brown Univ., Providence, R.I., 1979.
[53] L. Bauer, *Manual of Information to Accompany the Wellington Corpus of Written New Zealand English.* Dept. of Linguistics, Victoria Univ., Wellington, New Zealand, 1993.
[54] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm and System Development.* Prentice Hall, 2001.
[55] J.A. Pittman, "Handwriting Recognition: Tablet PC Text Input," *Computer,* vol. 40, no. 9, pp. 49-54, Sept. 2007.
[56] M.T. Johnson, "Capacity and Complexity of HMM Duration Modeling Techniques," *IEEE Signal Processing Letters,* vol. 12, no. 5, pp. 407-410, May 2005.

Alex Graves received the BSc degree in mathematical physics from the University of Edinburgh in 1998, the Certificate of Advanced Study in Mathematics from the University of Cambridge in 2001, and the PhD degree from the Dalle Molle Institute for Artificial Intelligence, Manno, Switzerland, and the Technical University of Munich, where he is currently a postdoctoral researcher. His research interests include recurrent neural networks, speech and handwriting recognition, and general sequence learning.



Marcus Liwicki received the MS degree in computer science from the Free University of Berlin, Germany, in 2004, and the PhD degree from the University of Bern, Switzerland, in 2007. Currently he is a senior researcher and lecturer at the German Research Center for Artificial Intelligence (DFKI). His research interests include knowledge management, semantic desktop, electronic pen-input devices, online and offline handwriting recognition, and document analysis. He is a coauthor of the book *Recognition of Whiteboard Notes Online, Offline, and Combination* (World Scientific, 2008). He has 20 publications, including five journal papers. He is a member of the International Association for Pattern Recognition (IAPR) and a frequent reviewer for international journals, including the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Transactions on Audio, Speech, and Language Processing*, and *Pattern Recognition*. He is a member of the program committee of the IEEE ISM Workshop on Multimedia Technologies for E-learning and a reviewer of several IAPR conferences and the IGS conference.

Santiago Fernández received the PhD degree in physics from the University of Santiago de Compostela, Spain. After this, he held a part-time lecturer position in the Department of Applied Physics from November 2001 to 2003. For the next two years, he was a postdoctoral Marie Curie fellow in the Department of Phonetics and Linguistics at University College London. In 2005, he joined the Dalle Molle Institute for Artificial Intelligence, Switzerland, where he is a postdoctoral researcher. His areas of expertise include speech recognition, machine learning, and human and machine perception. He is a regular contributor to international refereed conferences and journals.

Roman Bertolami received the MS degree in computer science from the University of Bern, Switzerland, in 2004. The topic of his master's thesis was the rejection of words in offline handwritten sentence recognition. He is now employed as a research and lecture assistant in the research group of computer vision and artificial intelligence at the University of Bern. His current research interests include the combination of multiple text line recognizer and confidence measures for handwritten text recognition systems.

Horst Bunke received the MS and PhD degrees in computer science from the University of Erlangen, Germany. In 1984, he joined the University of Bern, Switzerland, where he is a professor in the Computer Science Department. He was the department chairman from 1992 to 1996, the dean of the Faculty of Science from 1997 to 1998, and a member of the Executive Committee of the Faculty of Science from 2001 to 2003. He is a member of the Scientific Advisory Board of the German Research Center for Artificial Intelligence (DFKI). He has more than 550 publications, including 36 authored, coauthored, edited, or coedited books and special editions of journals. He received an honorary doctor degree from the University of Szeged, Hungary. He was on the program and organization committee of many conferences and served as a referee for numerous journals and scientific organizations.

Jürgen Schmidhuber received the doctoral degree in computer science from the Technische Universität München (TUM) in 1991 and the Habilitation degree in 1993, after a postdoctoral stay at the University of Colorado, Boulder. He has been a codirector of the Swiss Institute for Artificial Intelligence (IDSIA) since 1995, a professor of cognitive robotics at TUM since 2004, a professor at the Scuola Universitaria Professionale della Svizzera Italiana (SUPSI) since 2003, and also an adjunct professor of computer science at the University of Lugano, Switzerland, since 2006. His numerous research grants have yielded more than 200 peer-reviewed scientific papers on topics ranging from mathematically optimal universal AI and artificial recurrent neural networks to adaptive robotics and complexity theory, digital physics, and the fine arts.