Unconstrained On-line Handwriting Recognition with Recurrent Neural Networks

Abstract

On-line handwriting recognition is unusual among sequence labelling tasks in that the underlying generator of the observed data, i.e. the movement of the pen, is recorded directly. However, the raw data can be difficult to interpret because each letter is spread over many pen locations. As a consequence, sophisticated pre-processing is required to obtain inputs suitable for conventional sequence labelling algorithms, such as HMMs. In this paper we describe a system capable of directly transcribing raw on-line handwriting data. The system consists of a recurrent neural network trained for sequence labelling, combined with a probabilistic language model. In experiments on an unconstrained on-line database, we record excellent results using either raw or pre-processed data, well outperforming a state-of-the-art HMM in both cases.

1 Introduction

Handwriting recognition is traditionally divided into off-line and on-line recognition [12]. Off-line recognition is performed on images of handwritten text. In on-line handwriting the position of the pen-tip on a surface is recorded at regular time intervals, and the task is to map from the sequence of pen positions to the sequence of words.

At first sight, it would seem straightforward to label the raw on-line data directly. Unlike off-line handwriting, a compact and complete feature set is already explicit in the sensor readings, and does not have to be extracted from an image. However, the fact that each letter or word is distributed over many pen positions poses a problem for conventional sequence labelling algorithms, which have difficulty processing data with long-range interdependencies. The problem is especially acute for unconstrained handwriting, where the writing style may be either cursive or printed. The standard solution is to pre-process the data into a set of localised features. These features typically include geometric properties of the trajectory in the vicinity of every data point, off-line information from a generated image, and character level shape characteristics [7, 8]. Delayed strokes (such as the crossing of a "t" or the dot of an "i") require special treatment because they split up the characters and therefore interfere with localisation. HMMs [7] and hybrid systems incorporating time-delay neural networks and HMMs [8] are commonly trained with such features.

The issue of classifying pre-processed versus raw data has broad relevance to machine learning, and merits further discussion. Using hand-crafted features often yields superior results, and in some cases can render classification essentially trivial. However, there are three points to consider in
favour of raw data. Firstly, designing an effective pre-processor requires considerable time and expertise. Secondly, hand-coded features tend to be more task specific. For example, features designed for English handwriting could not be applied to languages with substantially different alphabets, such as Arabic or Chinese. In contrast, a system trained directly on pen movements could be re-trained on any language. Thirdly, using raw data allows feature extraction to be built into the classifier, and the whole system to be trained together. For example, convolutional neural networks, in which a globally trained hierarchy of network layers is used to extract progressively higher level features, have proved effective at classifying raw images [14, 10].

In this paper, we apply a recurrent neural network (RNN) to the task of on-line handwriting recognition. Experiments are carried out on the IAM database (IAM-OnDB) [11] which contains forms of unconstrained English text acquired from a whiteboard. The RNN architecture is bidirectional Long Short-Term Memory (BLSTM) [4], chosen for its ability to process data with long time dependencies. The RNN is trained with connectionist temporal classification (CTC) [3], an objective function specifically designed for labelling unsegmented sequence data. An algorithm is introduced for applying grammatical constraints to the CTC outputs, thereby providing word level transcriptions. The performance of the RNN system using both raw and pre-processed input data is compared to that of an HMM using pre-processed data only [1]. To the best of our knowledge, this is the first time whole sentences of unconstrained handwriting have been directly transcribed from raw on-line data.

Section 2 describes BLSTM and CTC and introduces the algorithm for integrating CTC with grammatical constraints. Section 3 provides experimental results, and conclusions are given in Section 4.

2 Method

2.1 Bidirectional Long Short-Term Memory

One of the key benefits of recurrent neural networks (RNNs) is their ability to make use of previous context. However, for standard RNN architectures, the range of context that can in practice be accessed is 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 recurrent connections. This is usually referred to as the vanishing gradient problem [5].

Long Short-Term Memory (LSTM) [6] is an RNN architecture specifically designed to address the vanishing gradient problem. An LSTM hidden layer consists of multiple recurrently connected subnets, known as memory blocks. Each block contains a set of internal units, known as cells, whose activation is controlled by three multiplicative ‘gate’ units. The effect of the gates is to allow the cells to store and access information over long periods of time, thereby avoiding the vanishing gradient problem.

For many tasks it is useful to have access to future as well past context. Bidirectional RNNs [13] achieve this by presenting the input data forwards and backwards to two separate hidden layers, both of which are connected to the same output layer. Bidirectional LSTM (BLSTM) [4] combines the above architectures to provide maximal access to long-range, bidirectional context.

2.2 Connectionist Temporal Classification

Connectionist temporal classification (CTC) [3] is an objective function designed for sequence labelling with RNNs. Unlike previous objective functions it does not require pre-segmented training data, or post-processing to transform the network outputs into transcriptions. Instead, it trains the network to map directly from input sequences to an estimate of the conditional probabilities of the possible labellings.

A CTC output layer contains one more unit than there are elements in the alphabet \( L \) of labels for the task. The output activations are normalised with the softmax activation function [2]. The first \( |L| \) outputs are used to estimate the probabilities of observing the corresponding labels at particular times. The extra output estimates the probability of observing a ‘blank’, or no label. The combined output sequence estimates the joint probability of all possible alignments of the input sequence with labels and blanks. The probability of a particular labelling can then be estimated by summing over the probabilities of the alignments that correspond to it.
More precisely, for an input sequence \( x \) of length \( T \), choosing a label (or blank) at every timestep according to the probabilities implied by the network outputs defines a probability distribution over the set of length \( T \) sequences of labels and blanks. We denote this set \( L^{T} \), where \( L = L \cup \{blank\} \) is the extended alphabet with blank included. To distinguish them from labellings, we refer to the elements of \( L^{T} \) as paths.

Since the probabilities of the labels at each timestep are conditionally independent given \( x \), the conditional probability of a path \( \pi \in L^{T} \) is given by:

\[
p(\pi | x) = \prod_{t=1}^{T} y_{\pi_{t}}^{t}.
\]

where \( y_{\pi_{t}}^{t} \) is the activation of output unit \( k \) at time \( t \).

Paths are mapped onto label sequences \( l \in L^{\leq T} \) by an operator \( B \) that removes first the repeated labels, then the blanks. For example, both \( B(a, - , a, b, -) \) and \( B(-, a, a, - , - , a, b, b) \) yield the labelling \((a,a,b)\). Since the paths are mutually exclusive, the conditional probability of a given labelling \( l \in L^{\leq T} \) is the sum of the probabilities of all paths corresponding to it:

\[
p(l | x) = \sum_{\pi \in B^{-1}(l)} p(\pi | x).
\]

Although a naive calculation of the above sum would be unfeasible, it can be efficiently evaluated with a graph-based algorithm [3], similar to the forward-backward algorithm for HMMs.

To allow for blanks in the output paths, for each label sequence \( l \in L^{\leq T} \) consider a modified label sequence \( l' \in L'^{\leq T} \), with blanks added to the beginning and the end and inserted between every pair of labels. The length of \( l' \) is therefore \( |l'| = 2|l| + 1 \).

For a labelling \( l \), define the forward variable \( \alpha_{t}(s) \) as the summed probability of all path beginnings reaching index \( s \) of \( l' \) at time \( t \), i.e.

\[
\alpha_{t}(s) = P \left( \pi_{1:t} : B(\pi_{1:t}) = 1_{1:s/2}, \tau_{t} = l'_{s}, \pi_{t} = 1_{s/2}^{t} \right) = \sum_{\pi_{1:t}, \pi_{t} = 1_{s/2}^{t}} y_{\pi_{t}}^{t},
\]

where, for some sequence \( s, a:b \) is the subsequence \((s_{a}, s_{a+1}, \ldots, s_{b-1}, s_{b})\). Note that index \( s \) of \( l' \) corresponds to index \( s/2 \) (rounded down) of the original label sequence \( l \).

The backward variables \( \beta_{t}(s) \) are defined as the summed probability of all path endings that would complete the labelling \( l \) if the path beginning had reached index \( s \) of \( l' \) at time \( t \):

\[
\beta_{t}(s) = P \left( \pi_{t+1:T} : B(\pi_{t:T}) = 1_{s/2:|l|}, \tau_{t} = l'_{s}, \pi_{t} = 1_{s/2}^{t+1} \right) = \sum_{\pi_{t+1:T}, \pi_{t} = 1_{s/2}^{t+1}} y_{\pi_{t}}^{t}.
\]

Both the forward and backward variables are calculated recursively [3]. The label sequence probability is given by the sum of the products of the forward and backward variables at any timestep:

\[
p(l | x) = \sum_{s=1}^{|l'|} \alpha_{t}(s) \beta_{t}(s).
\]

The objective function for CTC is the negative log probability of the network correctly labelling the entire training set. Let \( S \) be a training set, consisting of pairs of input and target sequences \((x, z)\), where the length of sequence \( z \) is less than or equal to the length of the input sequence \( x \). Then the objective function is:

\[
O^{CTC} = - \sum_{(x,z) \in S} ln \left( p(z | x) \right).
\]

The network can be trained with gradient descent by first differentiating \( O^{CTC} \) with respect to the outputs, then using backpropagation through time 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 labelling \( l \), we define the set of positions where label \( k \) occurs as \( lab(1, k) = \{ s : l^*_s = k \} \), which may be empty. We can then set \( I = z \) and differentiate (5) with respect to the network outputs to obtain:

\[
- \frac{\partial \ln p(z|x)}{\partial u_k^t} = y_k^t - \frac{1}{p(z|x)} \sum_{s \in lab(z, k)} \alpha_t(s) \beta_t(s),
\]

where \( u_k^t \) and \( y_k^t \) are respectively the unnormalised and normalised outputs of the softmax layer.

Once the network is trained, we would like to use it to label some unknown input sequence \( x \) by choosing the labelling \( l^* \) with the highest conditional probability:

\[
l^* = \arg \max_l p(l|x).
\]

Using the terminology of HMMs, we refer to the task of finding this labelling 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 labelling:

\[
l^* \approx B \left( \arg \max_\pi p(\pi|x) \right).
\]

### 2.3 Integration with an External Grammar

For some tasks we want to constrain the output labellings 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 (8) to be conditioned on some probabilistic grammar \( G \), as well as the input sequence \( x \):

\[
l^* = \arg \max_l p(l|x, G).
\]

Note that absolute requirements, for example that \( l \) contains only dictionary words, can be incorporated by setting the probability of all sequences that fail to meet them to 0. Applying the basic rules of probability, we obtain:

\[
p(l|x, G) = \frac{p(l|x)p(l|G), p(x)}{p(x|G)p(l)}
\]

where we have used the fact that \( x \) is conditionally independent of \( G \) given \( l \). If we assume that \( x \) is independent of \( G \), (11) reduces to:

\[
p(l|x, G) = \frac{p(l|x)p(l|G)}{p(l)}
\]

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 \( x \) 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, (10) reduces to:

\[
l^* = \arg \max_l p(l|x)p(l|G).
\]

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

We now describe an algorithm, based on the token passing algorithm for HMMs [15], that allows us to find an approximate solution to (13) for a simple grammar.

Let \( G \) consist of a dictionary \( D \) containing \( W \) words, and a set of \( W^2 \) bi-grams \( p(w|\hat{w}) \) 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 0.
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 $\text{tok}(w, -1, T)$ 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 $O(TW \log W)$ to account for the sort. If no bi-grams are used, lines 15-17 can be replaced by a simple search for the highest scoring output token, and the complexity reduces to $O(TW)$. 

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 $\text{tok} = \{\text{score}, \text{history}\}$ as a pair consisting of a real valued score and a history of previously visited words. Each token corresponds to a particular path through the network outputs, and the token score is the log probability of that path. Define a state $s$ of $w'$ as an index into the sequence of labels and blanks ($1 \leq s \leq |w'|$).

The basic idea of the token passing algorithm is to pass along the highest scoring tokens at every word state, then maximise over these to find the highest scoring tokens at the next state. The transition probabilities are used when a token is passed from the last state in one word to the first state in another. The output word sequence is then given by the history of the highest scoring end-of-word token at the final timestep.

At every timestep $t$ of the length $T$ output sequence, each state $s$ of each modified word $w'$ holds a single token $\text{tok}(w, s, t)$. This is the highest scoring token reaching that state at that time. In addition we define the input token $\text{tok}(w, 0, t)$ to be the highest scoring token arriving at word $w$ at time $t$, and the output token $\text{tok}(w, -1, t)$ to be the highest scoring token leaving word $w$ at time $t$.

Algorithm 1: CTC Token Passing Algorithm

1: Initialisation:
2: for all words $w \in D$ do
3:   $\text{tok}(w, 1, 1) \leftarrow \ln(y^1_{w1}, (w))$
4:   $\text{tok}(w, 2, 1) \leftarrow \ln(y^2_{w1}, (w))$
5: if $|w| = 1$ then
6:   $\text{tok}(w, -1, 1) \leftarrow \text{tok}(w, 2, 1)$
7: else
8:   $\text{tok}(w, -1, 1) \leftarrow (-\infty, ())$
9: $\text{tok}(w, s, 1) \leftarrow (-\infty, ())$ for all other $s$
10: end
11: Algorithm:
12: for time $t = 2$ to $T$ do
13:   sort output tokens $\text{tok}(w, -1, t - 1)$ by ascending score
14: for all words $w \in D$ do
15:   $\text{tok}(w, 0, t) \leftarrow \arg \max_{\text{tok}(w, -1, t - 1)} \text{tok}.\text{score} + \ln(p(w|\hat{w}))$
16:   $\text{tok}(w, 0, t).\text{score} += \ln(p(w|\hat{w}))$
17:   $\text{tok}(w, 0, t).\text{history} += w$
18: for segment $s = 1$ to $|w'|$ do
19:   $P \leftarrow \{\text{tok}(w, s, t - 1), \text{tok}(w, s - 1, t - 1)\}$
20: if $w'_s \neq \text{blank}$ and $s > 2$ and $w'_{s-2} \neq w'_s$ then
21:   $P += \text{tok}(w, s - 2, t - 1)$
22: $\text{tok}(w, s, t) \leftarrow \text{highest scoring token in } P$
23: $\text{tok}(w, s, t).\text{score} += \ln(y'^s_{w1})$
24: $\text{tok}(w, -1, t) \leftarrow \text{highest scoring of } \{\text{tok}(w, |w'1|, t), \text{tok}(w, |w'| - 1, t)\}$
25: end
26: Termination:
27: find output token $\text{tok}^*(w, -1, T)$ with highest score at time $T$
28: output $\text{tok}^*(w, -1, T).\text{history}$
3 Experiments

The experimental task was on-line handwriting recognition, using the IAM-OnDB handwriting database [11]. IAM-OnDB can be downloaded from http://www.iam.unibe.ch/fki/iamondb/

For the CTC experiments, we record the character error rate using standard best-path decoding, and the word error rate using Algorithm 1 with a language model and a dictionary. For the HMM system, the word error rate is quoted from the literature [1]. Both the character and word error rate are defined as the combined number of insertions, deletions and substitutions in the algorithm’s transcriptions of the input sequences in the test set, divided by the combined length of the target sequences in the test set.

We compare results using two different input representations, one hand-crafted for HMMs, the other consisting of raw data direct from the pen sensor.

3.1 Data and Preprocessing

IAM-OnDB consists of pen trajectories collected from 221 different writers using a ‘smart white-board’ [11]. The writers were asked to write forms from the LOB text corpus [9], and the position of their pen was tracked using an infra-red device in the corner of the board. The original input data consists of the $x$ and $y$ pen co-ordinates, the points in the sequence when individual strokes (i.e. periods when the pen is pressed against the board) end, and the times when successive position measurements were made. Recording errors in the $x, y$ data was corrected by interpolating to fill in for missing readings, and removing steps whose length exceeded a certain threshold.

IAM-OnDB is divided into a training set, two validation sets, and a test set, containing respectively 5364, 1438, 1518 and 3859 written lines taken from 775, 192, 216 and 544 forms. For our experiments, each line was used as a separate sequence (meaning that possible dependencies between successive lines were ignored).

The character level transcriptions contain 80 distinct target labels (capital letters, lower case letters, numbers, and punctuation). A dictionary consisting of the 20,000 most frequently occurring words in the LOB corpus was used for decoding, along with a bi-gram language model optimised on the training and validation sets [1]. 5.6% of the words in the test set were not in the dictionary.

Two input representations were used for this task. The first consisted simply of the offset of the $x, y$ co-ordinates from the top left of the line, along with the time from the beginning of the line, and an extra input to mark the points when the pen was lifted off the whiteboard. We refer to this as the raw input representation.

The second representation required a large amount of sophisticated pre-processing and feature extraction [1]. We refer to this as the pre-processed input representation. Briefly, in order to account for the variance in writing styles, the pen trajectories were normalised with respect to such properties as the slant, skew and width of the letters, and the slope of the line as a whole. After that, two sets of input features were extracted, the first consisting of ‘online’ features, such as pen position, pen speed, line curvature etc., and the second consisting of ‘offline’ features created from a two dimensional window of the image created by the pen.

3.2 Experimental Setup

The CTC network used the BLSTM architecture [4]. The forward and backward hidden layers each contained 100 single cell memory blocks. The input layer was fully connected to the hidden layers, which were fully connected to the output layer. The output layer contained 81 units (80 characters plus the blank label). For the raw input representation, there were 4 input units, giving 100,881 weights in total. For the pre-processed representation, there were 25 input units, giving 117,681 weights in total. Hyperbolic tangent was used for the cell activation functions and logistic sigmoid in the range $[0, 1]$ was used for the gates. For both input representations, the data was normalised so that each input had mean 0 and standard deviation 1 on the training set. The network was trained with stochastic steepest descent, using a learning rate of $10^{-4}$ and a momentum of 0.9. Training was stopped after no improvement was recorded on the validation set for 50 training epochs.
Table 1: **Word Error Rate (WER) on IAM-OnDB.** LM = language model. CTC results are a mean over 4 runs, ± standard error. All differences were significant ($p < 0.01$)

<table>
<thead>
<tr>
<th>System</th>
<th>Input</th>
<th>LM</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM</td>
<td>pre-processed</td>
<td>✓</td>
<td>35.5% [1]</td>
</tr>
<tr>
<td>CTC</td>
<td>raw</td>
<td>×</td>
<td>30.1 ± 0.5%</td>
</tr>
<tr>
<td>CTC</td>
<td>pre-processed</td>
<td>×</td>
<td>26.0 ± 0.3% [1]</td>
</tr>
<tr>
<td>CTC</td>
<td>raw</td>
<td>✓</td>
<td>22.8 ± 0.2%</td>
</tr>
<tr>
<td>CTC</td>
<td>pre-processed</td>
<td>✓</td>
<td>20.4 ± 0.3%</td>
</tr>
</tbody>
</table>

The HMM setup [1] contained a separate, left-to-right HMM with 8 states for each character ($8 \times 81 = 648$ states in total). Diagonal mixtures of 32 Gaussians were used to estimate the observation probabilities. All parameters, including the word insertion penalty and the grammar scale factor, were optimised on the validation set.

### 3.3 Results

The character error rate for the CTC network with the pre-processed inputs was 11.5 ± 0.05%. From Table 1 we can see that with a dictionary and a language model this translated into a mean word error rate of 20.4%, which is a relative error reduction of 42.5% compared to the HMM. Without the language model, the error reduction was 26.8%.

With the raw input data CTC achieved a character error rate of 13.9 ± 0.1%, and word error rates that were comparable to those recorded with the pre-processed data. This result corroborates the ability of the LSTM architecture to access long-range contextual information. When we attempted to train a CTC network with a standard bidirectional RNN on the same task, the character error rate never dropped below 90%.

A useful indicator of contextual sensitivity is provided by the derivatives of the output $y_k^t$ at a particular point $t$ in the data sequence with respect to the inputs $x_{k}^{t'}$ at all points $1 \leq t' \leq T$ in the sequence. We refer to the sequence of these derivatives as the *sequential Jacobian*. Looking at the relative magnitude of the sequential Jacobian at different timesteps gives an idea of the range of context that is being used.

Figure 1 shows a sequential Jacobian for CTC networks using both the raw and pre-processed inputs for a phrase from IAM-OnDB. For both input representations the region of sensitivity extends forwards and backwards from the given timestep, which supports the use of bidirectional networks. For the pre-processed representation, the Jacobian is sharply peaked around the time when the output is emitted. For the raw inputs, the Jacobian is more spread out, which suggests that the network does indeed make more use of long-range contextual information. Note the high sensitivity to the very end of the input sequence: this corresponds to the delayed dot of the letter “i”.

### 4 Conclusion

We have combined a BLSTM CTC network with a probabilistic language model. We have applied this system to an on-line handwriting database and obtained results that substantially improve on a state-of-the-art HMM. We have also shown that the network’s performance with raw sensor data is comparable to that with specially pre-processed data. As far as we are aware, our system is the first to successfully recognise unconstrained on-line handwriting using raw data only.

### References


Figure 1: Sequential Jacobian for a sequence from the IAM-OnDB, with raw inputs (left) and pre-processed inputs (right). For ease of visualisation, only the derivative with highest absolute value is plotted at each timestep. The reconstructed image was created by plotting the pen co-ordinates recorded by the sensor. The individual strokes are alternately coloured red and black. For both representations, the Jacobian is plotted for the output corresponding to the label ‘i’ at the point when ‘i’ is emitted (indicated by the vertical dashed lines).


