Generating a Training Corpus for OCR Post-Correction Using Encoder-Decoder Model

Eva D'hondt
LIMSI, CNRS, Univ. Paris-Saclay
F-91405 Orsay
eva.dhondt@gmail.com

Cyril Grouin
LIMSI, CNRS, Univ. Paris-Saclay
F-91405 Orsay
cyril.grouin@limsi.fr

Brigitte Grau
LIMSI, CNRS, ENSIIE, Univ. Paris-Saclay
F-91405 Orsay
bg@limsi.fr

Abstract

In this paper we present a novel approach to the automatic correction of OCR-induced orthographic errors in a given text. While current systems depend heavily on large training corpora or external information, such as domain-specific lexicons or confidence scores from the OCR process, our system only requires a small amount of relatively clean training data from a representative corpus to learn a character-based statistical language model using Bidirectional Long Short-Term Memory Networks (biLSTMs). We demonstrate the versatility and adaptability of our system on different text corpora with varying degrees of textual noise, including a real-life OCR corpus in the medical domain.

1 Introduction

Recently, Optical Character Recognition (OCR) technology has improved substantially, which has allowed for a large-scale digitization of textual resources such as books, old newspapers, ancient hand-written documents, etc. (Romero et al., 2011). The quality of the subsequent digital corpora can vary substantially, depending on factors such as quality of the original paper, ink quality, differences in fonts, etc. The amount of noise in digital collections can have a severe negative impact on the accuracy of subsequent text mining processes (e.g., Named Entity Recognition, Information Extraction, etc.).

OCR post-correction techniques are used to improve the text quality of OCR output. Traditionally, they rely on domain-specific lexicons (de Does and Depuydt, 2013) and character-based errors statistics obtained from a corrected training set (Kumar and Lehal, 2016). However, they have some drawbacks that limit their usefulness for specific, low-resource domains. Such resources are expensive to create and for highly specialized texts (e.g., medical domain) not always possible to obtain. The recent advances in neural network models, based on textual context and needing no external resources, provide new opportunities for OCR post-correction. Character-level sequence modeling architectures are especially suited for this task (Chrupała, 2014; Schmaltz et al., 2016), as they reduce the complexity at output time. Moreover, current systems are often limited to processing texts with a limited degree of OCR corruption, i.e., so-called single-error word corrections (Kissos and Dershowitz, 2016) and correction of OCRed corpora that have been generated by older OCR engines can prove too challenging. The correct recognition of historical texts remains an open challenge (Kluzner et al., 2009). A general-purpose OCR post-correction tool should be adaptable to the ratio of error that is present in the OCR output in order to deal with both types of errors.

In this paper, we propose a novel approach to OCR post-correction using bidirectional recurrent neural networks for learning a robust character-based language model that (i) captures the domain-specific vocabulary of a text and (ii) which is able to detect and correct noise in corrupted text in the same time. In order to overcome the problem of producing manually annotated data, our system requires zero pre-annotated training material. Rather than using a large corrected training corpus to learn an error model, we propose a method of generating our own training material from clean text. This has multiple advantages: it allows us complete control over the learning process, and we can train on larger and more diverse corpora.

Proceedings of the The 8th International Joint Conference on Natural Language Processing, pages 1006–1014,
Taipei, Taiwan, November 27 – December 1, 2017

First, we demonstrate the flexibility of the proposed model by evaluating it on artificially created test sets with varying degrees of noise. We also show how variation inherent in the texts influences training rates. Second, we explore a more realistic setting in which very little clean training data is available to learn a language model. We test our method on a real-life OCRed corpus of French medical reports. Our method outperforms the baseline by 14.3% on domain-specific noisy data, even when the latter is supplemented with a domain-specific lexicon.

2 Background

The problem of OCR post-correction has been studied since the seventies (Kukich, 1992). While traditional OCR error detection systems focused on constructing 'confusion matrices' of likely character (pairs) to detect corruptions of existing words into non-words, recent systems improve accuracy using information on the language context in which the error appears (Evershed and Fitch, 2014), using bigrams (Kissos and Dershowitz, 2016), large-scale word n-grams and character n-grams from the web (Bassil and Alwani, 2012) or associating confusion scores into a Bayesian model (Tong and Evans, 1996) or a HMM model (Borovikov et al., 2004) to select the optimal word candidate. These systems are explicitly or implicitly limited to cases in which an erroneous word appears in an otherwise clean context. For serious degrees of corruption (e.g., historical texts), the common approach aims to optimally combine an ensemble of multiple OCR engines (Nakano et al., 2004; Lund and Ringger, 2009).

'Noisy channel paradigm' aims to learn error models describing the OCR output generation from the reference text, and as such combine error and language models. Kolak and Resnik (2005) used finite state machines on a small set of training material while Llobet et al. (2010) combined all OCR process hypotheses for each recognized character. Such models need a large amount of training material which is costly and not always easily available. In response, the Text-Induced Corpus Clean-up (TICCL) system (Reynaert, 2011) was developed to run with no annotated training data. It takes noisy texts and extracts the high-frequency word variants through statistical analysis and clusters typographical word variants within a user-defined Levenshtein distance.

Recently, Neural Network Language Models have proven to be extremely effective in complex NLP tasks. For spelling errors correction, systems either include auto-encoders to detect nearest neighbor matches of spelling errors with correct words (Raaijmakers, 2013) or learn edit operations from labeled data while incorporating features induced from unlabelled data via character-level neural text embeddings (Chrupała, 2014). Contrary to Azawi (2015) which makes use of LSTM based on character-aligned strings, our method does not require annotated training data (gold standard) to learn the character-based language model. Moreover, our method does not capitalize on learning character transformation rules based on frequently occurring errors, as done in Azawi (2015), but learns a robust character-based language model.

3 Method

The model we proposed consists of a many-to-many character sequence learning network using long short term memory (LSTM) nodes.

3.1 Definition

LSTM are a special type of Recurrent Neural Network (RNN), a neural network hierarchy designed to model times series or other sequences. Standard RNNs have trouble capturing long-distance sequential dependencies, as the error signal during back propagation tends to disperse or blow up over time, which is known as the problem of vanishing or exploding gradients (Hochreiter, 1991; Bengio et al., 1994). This problem is typically addressed by replacing the standard RNN cell with a long short-term memory cell, which allows for a constant error flow along the input sequence (Hochreiter and Schmidhuber, 1997).

Technically, the LSTM architecture is given by the following equations,

$$
\begin{aligned}
i^{(t)} &= \sigma(W^{ix}.x^{(t)}) + W^{ih}.h^{(t-1)} + b_i) \\
f^{(t)} &= \sigma(W^{fx}.x^{(t)}) + W^{fh}.h^{(t-1)} + b_f) \\
o^{(t)} &= \sigma(W^{ox}.x^{(t)} + W^{oh}.h^{(t-1)} + b_o) \\
g^{(t)} &= tanh(W^{gx}.x^{(t)} + W^{gh}.h^{(t-1)} + b_g) \\
c^{(t)} &= f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot g^{(t)} \\
h^{(t)} &= tanh(c^{(t)}) \odot o^{(t)}
\end{aligned}
$$

in which σ is the sigmoid function, i stands for the input gate (a selection of information coming from a previous cell state, given the new informa-

tion contained in x); f stands for the forget gate, which select which parts of the information to forget and which to retain; g creates a vector of new candidate values that are used to update the cell state; o stands for the output gate which decides what parts of the cell state should be outputted.

A bidirectional LSTM (biLSTM) is a combination of two unidirectional LSTM layers, the first of which encodes the input string from left-to-right, the second from right-to-left. This ensures that errors that occur at the beginning of the input string have enough context (on the right-hand side) to be corrected.

3.2 Encoder-Decoder Model: biLSTM model

We stack two bidirectional LSTM layers on top of each other: the first hidden level is an encoder that reads the source character sequence and the other is a decoder that functions as a language model and generates the output. Figure 1 shows the layers in their unrolled forms as they read in the input.

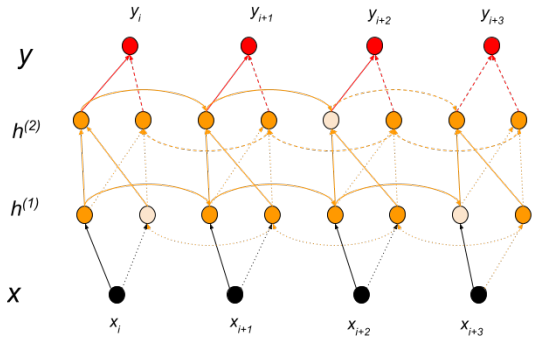


Figure 1: Hierarchy of bidirectional 2-layer many-to-many sequence learning network; lighter nodes in the hidden layers refer to disconnected nodes due to dropout

Architecture We first define a character vocabulary $V = \{v_1, v_2, ..., v_{|V|}\}$ which contains all the characters that are present in the training corpus. Each input string of 20 characters is padded on the right-hand side to a maximum length of 23, which allows for differences in string length due to insertions in the training phase (see Section 3.3). For an input string we define a 23-dimensional vector x in which each element corresponds to the index of the corresponding character in vocabulary V, thus preserving the order. The input x is then passed to an embedding layer that returns the sequence $S = \{c_j | j = x_1, x_2..., x_{23}\}$ where c_j is the character embedding (by a hyperparameter d = 64) for the characters in the initial string. This sequence is then passed to the first biLSTM, which combines two unidirectional LSTM layers that process the input from the left-to-right and right-to-left hand side, respectively. Each LSTM layer H consists of k LSTM memory units, and its output is a matrix $H = \{h_t \epsilon \mathbb{R}_k \mid t = 1, 2, ..., 23\}$. We set k at 512. The output of the individual LSTM layers is concatenated and given as input for the second biLSTM layer. The last hidden layer then projects unto a $|V|$-dimensional output layer. Finally, we apply the softmax function to select the character with the highest probability for a given timestep and obtain a 23-dimensional vector y with the characters indices for the corrected string.

We also added two drop-out layers to the hidden layers, each set at 0.5, since this has been shown to improve performance (Srivastava et al., 2014). The model was implemented in Keras, a python library for deep learning.

3.3 Training strategy

During training, the neural network is fed corrupted input strings and provided the original non-corrupted string as output labels. In this way, the network learns the domain-specific character-based language model that underlies the text in the training documents, while at the same time it learns to detect and eliminate noise introduced by OCR errors. Following the observations that OCR-induced variation is generally much less systematic than spelling errors (Reynaert, 2008), we generate corrupted strings by randomly deleting, inserting and substituting one or two characters for a given string (cf. algorithm 3.1).

We used a random number generator to determine if and which edit operations were selected. Character substitutions were performed at random with characters from the character set. Since a string could be submitted to multiple corrupting edits, this results in both single-error as well as multi-error words and environments in the corrupted string. Since the if-statements are independent, multiple edits on one string are possible, which can result in longer consecutive errors (three or more consecutive characters can be corrupted for a given string). In our script, the ratio of noise is set by the user, which corresponds to a fixed level of corruption in the training data.

Algorithm 3.1: CORRUPTSTRING($str, noiseRatio$)

$len = len(str)$
$chars = set\ of\ characters\ in\ corpus$

comment: Randomly delete a character

if $rand() < (noiseRatio * len)$:
then $\begin{cases} pos = randInt(len) \\ str = str[:pos] + str[pos+1:] \end{cases}$

comment: Randomly insert a character

if $rand() < (noiseRatio * len)$:
then $\begin{cases} pos = randInt(len) \\ str = str[:pos] + randChoice(chars) \\ \quad +str[pos:] \end{cases}$

comment: Randomly replace 1 or 2 characters

if $rand() < (noiseRatio * len)$:
then $\begin{cases} numChars = randChoice([1,2]) \\ pos = randInt(len) \\ do\ numChars\ time \begin{cases} str = str[:pos] \\ \quad +(randChoice(chars)) \\ \quad +str[pos+1:] \\ pos++ \end{cases} \end{cases}$

Table 1 shows the generated input strings for a given string for the different noise ratios we used. In this table, the second row shows the percentage of strings with at least one OCR error in the real-life test set. We split the initial text into windows of 20 characters with an overlap per 3 characters. This length was empirically determined on training experiments w.r.t. accuracy and training speed.

Table 1: Percentage and examples of corrupted strings w.r.t. the noise ratio on the clean corpus. Bold highlights generated errors

noise ratio	corrupted strings	strings (20 characters length) original: n oven for 15 minute
0.001	5%	
0.003	17%	
0.005	30%	
0.01	48%	n o**5**en for 15 minute
0.02	76%	
0.03	92%	**no**ven f**Q**r 15 minute**#**
0.04	99%	n o**e**en for 1**%** min**ü**te

4 Corpora

In our experiments, we used a few corpora, both 'clean' (digital corpora that contain no OCR or orthographic errors) and 'real-life' corpora (that contain varying degrees of OCR errors). They were selected according to two criteria: (i) text genre, either 'structured text' or 'free text', and (ii) domain, 'general domain' or 'specialized domain', more particularly the medical domain.

Table 2 gives an overview of the different corpora and their attributes. We give the sizes of the corpora in number of characters, rather than words since the varying degrees of OCR errors make the latter an unreliable metric.

Table 2: Overview of used corpora

	text style & language	domain	size train/test
CURD	structured English	general	150K 44K
LM	free text French	general	2M 100K
Handbook	free text French	foetopath	1M -
medical reports	structured French	other medical	1M -
foetopath reports	structured French	foetopath	500K 57K

We generate several artificial test sets for two clean corpora. Artificially corrupted strings are created using the same methodology as used for training. We generated two types of artificial test sets:

- test sets with a set noise ratio, which contain both corrupted and uncorrupted strings
- and test sets that contain only corrupted strings of a set Levenshtein Distance (LD)

4.1 Clean corpora

CURD corpus The Carnegie Mellon University Recipe Database corpus (Tasse and Smith, 2008) contains 260 structured cooking recipes in English: a list of ingredients followed by short descriptive sentences. This corpus closely resembles the real-life OCRed corpus of medical reports presented in the next section. Unlike medical reports, we consider it to be 'general domain'. We did not use the semantic annotations in the original CURD corpus but extracted only the plain text.

LM corpus We used documents from the *Le Monde* newspaper from 2000 to 2005 on as training material, and created test sets on a subset of articles from 2006.

Handbook on foetopathology We used an electronic copy of a comprehensive French handbook of foetopathology (Bouvier et al., 2008) to train a domain-specific language model of free text for the foetopathological domain.

Medical reports This set of in-house French medical reports were written in the same reporting style as the real-life OCRed foetopathological reports presented in Section 4.2. While medical, they do not treat the foetopathological domain but rather cancer and gastro-internal illnesses.

4.2 Real-life OCR corpora

The foetopathological reports corpus is a data set of French patient notes from the domain of foetopathology, spanning 22 years. In total, the corpus contains the files from 2476 individual patients. The files were processed with a custom-trained commercial OCR engine, and later de-identified with an in-house de-identification tool (Grouin and Zweigenbaum, 2013).

Since the model of the OCR engine used to convert the entire corpus was trained on a subset of documents of more recent years (implying good paper quality, clear font, etc.), the OCR quality of the OCRed documents decreases substantially for the older documents (D'hondt et al., 2016).

All evaluations in this paper were carried out on an annotated set of 53 files, for which reference texts have been created manually by one annotator in two passes.[1] We extracted two sets of training material from the corpus. One set has a reasonable OCR quality,[2] the second set is taken randomly from the corpus and contains texts with varying degrees of OCR quality. Both training sets have the same size.

5 Evaluation

5.1 Evaluation metrics

For evaluation, we use the CER metric (formula 1), as defined in OCR post-correction evaluations:

$$CER = \frac{S + D + I}{S + D + C} \quad (1)$$

where S refers to the number of substituted characters in the OCR text (w.r.t. the reference texts), D to the number of deleted characters, I to the number of inserted characters and C to the number of 'correct' characters. We use the CER metric when comparing to the baseline system (see Section 6.4), using the `ocrevalUAtion`[3] package (Carrasco, 2014).

Since our models use overlapping[4] windows of 20 characters, a purely character-based metric is not ideal to evaluate. We want to measure the systems performance per input, rather than per character. We therefore introduce two complementary accuracy-based evaluation metrics, which evaluate on the level of the character window:

- detection accuracy (detAcc) shows the proportion of correctly detected errors and non-errors in the evaluated set of 20-character strings
- correction accuracy (corrAcc) reflects the ability of the language model to accurately correct corrupted strings without overgenerating and editing non-corrupted strings

These metrics are calculated as follows:

$$detAcc = \frac{(TP + TN + incorrectEdit)}{(TP + TN + FP + FN + incorrectEdit)}$$

$$corrAcc = \frac{(TP + TN)}{(TP + TN + FP + FN + incorrectEdit)}$$

Table 3 illustrates the different elements of the formulae.

5.2 Baseline model

We compare our system against the only other approach which requires zero annotated training material and no external resources, the TICCL system (Reynaert, 2011). As explained in Section 2, this system is word-based and uses anagram hashing to handle lexical variation in a large, noisy text collection. The system analyses a corpus to select high-frequency word variants, and then aims to map near-neighbours (in terms of edit distance) to those forms in order to reduce global variation in the corpus.

[1] These reference texts were later verified by a second annotator. The role of the second annotator was to check that the existing annotations were correct and consistent. Ergo the annotations were not done independently.

[2] Based on the proportion of out-of-vocabulary words present in a document for a given lexicon.

[3] https://github.com/impactcentre/ocrevalUAtion

[4] Allowing this overlap is a deliberate choice to maximize the ability to learn language models over a small corpus.

Table 3: Example of evaluation categories for the detection accuracy (detAcc) and correction accuracy (corrAcc) metrics, given the reference string 'n oven for 15 minute'

	input string	output string
True Positive (TP)	n o**5**en for 15 minute	n o**v**en for 15 minute
incorrectEdit	n o**5**en for 15 minute	n o**v**en for 15 m**l**nute
True Negative (TN)	n o**v**en for 15 minute	n o**v**en for 15 minute
False Positive (FP)	n o**v**en for 15 minute	n o**v**en for 15 m**l**nute
False Negative (FN)	n o**5**en for 15 minute	n o**5**en for 15 minute

We used the system available for French with pretrained character confusion models with its default settings.[5] For the foetopathological reports test corpora, we provided TICCL with the full available corpora to extract word variants but calculated CER only on the test sets. TICCL performs specific preprocessing to limit the size of its character vocabulary (all numbers and digits are mapped unto the character '3').

6 Experiments and Discussion

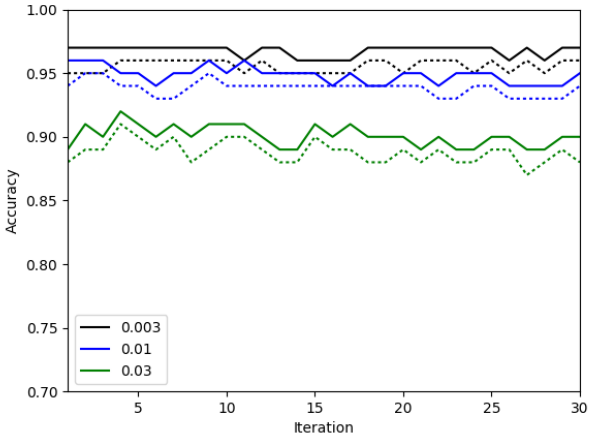
Since our system does not use annotated data, the language and error models learned from the training data are approximations of what will be encountered in the test set. In Sections 6.1 and 6.2, we examine how the error model (the prior probability of encountering an error in a given input string) learned during training corresponds to the (expected) noise ratio in the test set, and how the text genre and size of the training corpus influence performance. Section 6.3 presents how Language Models can be learned when no clean training material is available for a given test set. Section 6.4 shows a comparison with the word-based baseline.

[5]Online TICCL interface for French with default settings: http://ticclops.clarin.inl.nl/ticclops/

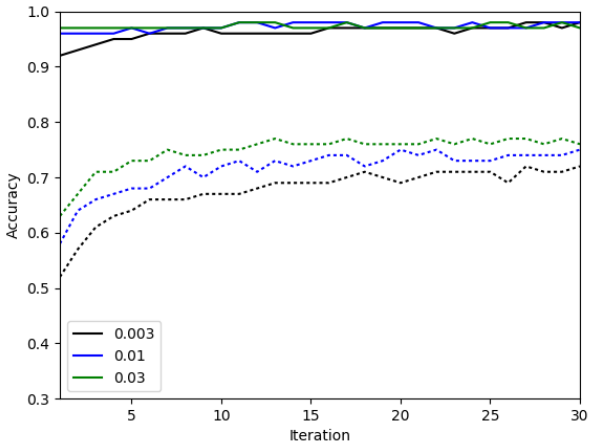
6.1 Adaptability to different degrees of noise

One of the advantages of our approach is its adaptability to different levels of expected noise. It suffices to train the model with a noise ratio that corresponds to the noise found in the test set.

The upper subfigure in Figure 2 shows the detection accuracy (full line) and correction accuracy (dotted line) on a test set with a 0.005 noise ratio for three different models which were trained in 30 iterations on the CURD training data with 0.003, 0.01 and 0.03 noise ratio, respectively.



(a) 0.005 noise ratio



(b) 0.03 noise ratio

Figure 2: Performance of models trained with different noise ratios on the CURD corpus for the 0.005 noise ratio test (upper) and 0.03 noise ratio test (lower), in terms of detection accuracy (full line) and correction accuracy (dotted line)

A 0.005 noise ratio is a fairly easy test set, with few errors, as is evidenced by the overall high scores. We do see that the model which was trained to expect a lot of noise (0.03) underperforms compared to its more conservative counterparts. For the same models on the 0.03 test set,

however, the more aggressively trained model performs the best. The high detection accuracy scores in both figures (full lines) show that the models are correctly identifying errors (i.e., unlikely character sequences) and have little tendency to ignore errors (FN) or incorrectly edit a correct string (FP). The difference between the detection accuracy (full lines) and correction accuracy (dotted lines) in Figure 2 shows the increased difficulty for the language models to make correct edits when the input strings have multiple corruptions.

Figure 3 examines this more closely. Here, the training model has been fixed (trained with noise ratio 0.03) and we examine its performance on test sets were each corrupted string has the same Levenshtein distances to its reference string. Here as well, overall detection accuracy is quite high, but the increasingly lower correction accuracy scores show the increased difficulties of the language model to propose correct edits for a given corrupted string. Since the 0.03 model is trained to expect moderate corruption, it performs best on corrupted strings with a Levenshtein distance of 2.

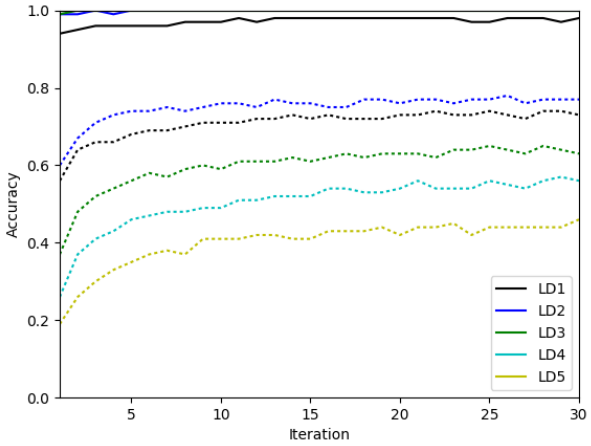


Figure 3: Performance of best model (ratio 0.03) on test sets which contain only corrupted substrings, with a fixed Levenshtein Distance (LD), in terms of detection accuracy (full line) and correction accuracy (dotted line)

6.2 Impact of text variability on training

The previous experiments were carried out on the CURD corpus, a small corpus with relatively fixed vocabulary and structured text, which makes it easy to learn a comprehensive model. Figure 4 shows the performance of a corpus with more inherent variation, the journalistic LM corpus.

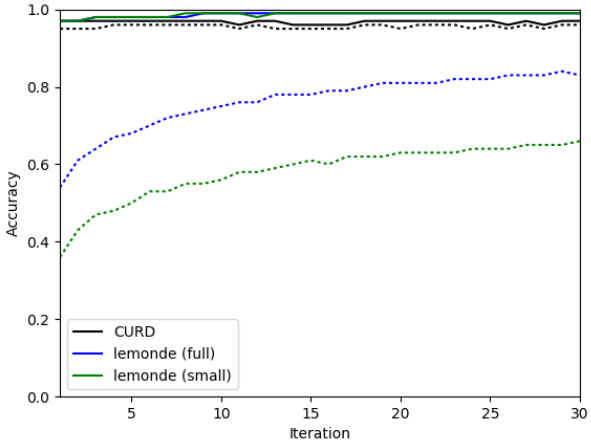


Figure 4: Performance of best-performing models on the CURD and LM 0.005 test set, in terms of detection accuracy (full line) and correction accuracy (dotted line)

As expected, the language model of a more variable corpus has a slower learning rate and needs more training data to achieve good performance. The green lines (lemonde (small)) refer to a subset of the LM corpus of exactly the same size as the CURD corpus (150K characters), which is clearly not enough training material to learn a language model on free text.

6.3 Adaptation to a real-life corpus

While the previous experiments only used artificial test sets, a more realistic setting is that in which an OCRed test set needs to be corrected but no clean data is available to train the system on.

Table 4 presents the results we achieved on the real life test set while training our model on different types of text that are either similar to the test set w.r.t. text genre or domain. We also compared these to the performance of models that were trained on noisy training data from the same corpus as the test set. For the foetopathological reports, we trained models on two test sets of similar size, one which contained data with relatively few OCR errors, and one whose text quality was mixed.

Overall, we find that the language model of the structured foetopathological reports is fairly easy to learn, as evidenced by the high correction scores. The scores in the first two rows of Table 4 show the impact of noisy training data. While relatively clean training data pose little problem, training on the mixed set leads to a more corrupted language model. Interestingly, we find that training

Table 4: Performance of different models trained on different training data (with 0.003 noise ratio) on real-life test set of foetopathological reports

	detAcc	corrAcc
foet. reports (fairly clean)	0.87	0.82
foet. reports (mixed)	0.86	0.76
medical reports (clean)	0.85	0.79
foet. handbook (clean)	0.87	0.81
LM corpus (clean)	0.60	0.52

on similar corpora, both in text genre (medical reports) and domain (Handbook) leads to almost as good correction accuracy as training on the original corpus. The language model trained on the LM corpus, whilst having seen the most training data, has the worst performance since the text in this corpus is too far removed from foetopathological reports, both in domain and text genre.

6.4 Comparison with baseline

To compare against our baseline system, we transformed the output of the two best performing systems from the experiments in Section 6.3 to a single final output string. Where two characters differed in the overlapping output strings from our system, we used majority vote to construct the final output string.

Table 5 shows the results against the TICCL baseline system, using the CER metric. TICCL-lex refers to a setting in which it was provided with a French, domain-specific lexicon of frequent terms in the foetopathology domain. We find that our system significantly outperforms the baseline on the structured, domain-specific data, by 14.3%.

Table 5: Comparison of baseline system (TICCL) with best performing systems on the foetopathological real-life test set

	Character Error Rate (CER)
original text	34.3
biLSTM	7.1
TICCL	34
TICCL-lex	21.4

7 Conclusion

In this paper we proposed a novel zero-annotated data approach to OCR post-correction. We used biLSTMs to build an encoder-decoder model. Rather than learning from annotated data, we developed a method to generate our own training material. Our model is trained on clean or moderately clean data in order to produce a robust character-based language model.

We have evaluated our method on different text genres and domains. We found that our method is especially suited to correct domain-specific, structured text, even when no training text from the same corpus is available.

Acknowledgments

This work was partially supported by the French National Agency for Research under the grant Accordys[6] ANR-12-CORD-0007.

References

Mayce Ibrahim Ali Al Azawi. 2015. *Statistical Language Modeling for Historical Documents using Weighted Finite-State Transducers and Long Short-Term Memory*. Ph.D. thesis, University of Kaiserslautern.

Youssef Bassil and Mohammad Alwani. 2012. OCR context-sensitive error correction based on Google web 1t 5-gram data set. *American Journal of Scientific Research*.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Eugene Borovikov, Ilya Zavorin, and Mark Turner. 2004. A filter based post-OCR accuracy boost system. In *Proc of ACM Work. on Hardcopy document processing*, pages 23–28.

Raymonde Bouvier, Dominique Carles, Marie-Christine Dauge, Pierre Déchelotte, Anne-Lise Delézoide, Bernard Foliguet, Dominique Gaillard, Bernard Gasser, Marie Gonzalès, and Férechté Encha-Razavi. 2008. *Pathologie fœtale et placentaire pratique*. Sauramps Médical.

Rafael C Carrasco. 2014. An open-source OCR evaluation tool. In *Proc of Digital Access to Textual Cultural Heritage*, pages 179–184.

Grzegorz Chrupała. 2014. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proc of ACL*, volume 2, pages 680–686.

Eva D'hondt, Cyril Grouin, and Brigitte Grau. 2016. Low-resource OCR error detection and correction in French Clinical Texts. In *Proc of LOUHI*, pages 61–68, Lisbon, Portugal.

[6] Agrégation de Contenus et de COnnaissances pour Raisonner à partir de cas dans la DYSmorphologie foetale

Jesse de Does and Katrien Depuydt. 2013. Lexicon-supported OCR of eighteenth century Dutch books: a case study. In *Proc. of SPIE*, volume 8658 of *Document Recognition and Retrieval*.

John Evershed and Kent Fitch. 2014. Correcting noisy OCR: Context beats confusion. In *Proc of ICDAR*, pages 45–51.

Cyril Grouin and Pierre Zweigenbaum. 2013. Automatic de-identification of French clinical records: Comparison of rule-based and machine-learning approches. In *Stud Health Technol Inform*, volume 192, pages 476–80, Copenhagen, Denmark.

Sepp Hochreiter. 1991. *Untersuchungen zu dynamischen neuronalen Netzen*. Ph.D. thesis, Technische Universität München.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ido Kissos and Nachum Dershowitz. 2016. OCR error correction using character correction and feature-based word classification. In *Proc of Document Analysis Systems Work*, Santorini, Greece.

Vladimir Kluzner, Asaf Tzadok, Yuval Shimony, Eugene Walach, and Apostolos Antonacopoulos. 2009. Word-based adaptive OCR for historical books. In *Proc of ICDAR*, pages 501–505.

Okan Kolak and Philip Resnik. 2005. OCR post-processing for low density languages. In *Proc of EMNLP*, pages 867–874.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439.

Atul Kumar and Gurpreet Singh Lehal. 2016. Automatic text correction for Devanagari OCR. *Indian Journal of Science and Technology*, 9(45).

Rafael Llobet, Jose-Ramon Cerdan-Navarro, Juan-Carlos Perez-Cortes, and Joaquim Arlandis. 2010. OCR post-processing using weighted finite-state transducers. In *Proc of ICPR*, pages 2021–2024.

William B Lund and Eric K Ringger. 2009. Improving optical character recognition through efficient multiple system alignment. In *Proc of Digital libraries*, pages 231–240.

Yasuaki Nakano, Toshihiro Hananoi, Hidetoshi Miyao, Minoru Maruyama, and Kenichi Maruyama. 2004. A document analysis system based on text line matching of multiple OCR outputs. *Lecture Notes in Computer Science*, pages 463–471.

Stephan Raaijmakers. 2013. A deep graphical model for spelling correction. In *Proc of Benelux Conference on Artificial Intelligence*, Delft, The Netherlands.

Martin Reynaert. 2008. Non-interactive OCR post-correction for giga-scale digitization projects. In *Proceedings of the 9th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'08, pages 617–630, Berlin, Heidelberg. Springer-Verlag.

Martin WC Reynaert. 2011. Character confusion versus focus word-based correction of spelling and OCR variants in corpora. *International Journal on Document Analysis and Recognition*, 14(2):173–187.

Verónica Romero, Nicolás Serrano, Alejandro H. Toselli, Joan Andreu Sánchez, and Enrique Vidal. 2011. Handwritten text recognition for historical documents. In *Proceedings of the Language Technologies for Digital Humanities and Cultural Heritage Workshop*, pages 90–96, Hissar, Bulgaria.

Allen Schmaltz, Yoon Kim, Alexander M Rush, and Stuart M Shieber. 2016. Sentence-level grammatical error identification as sequence-to-sequence correction. *arXiv preprint arXiv:1604.04677*.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Dan Tasse and Noah A Smith. 2008. SOUR CREAM: Toward semantic processing of recipes. *Carnegie Mellon University, Pittsburgh, Tech. Rep. CMU-LTI-08-005*.

Xiang Tong and David A Evans. 1996. A statistical approach to automatic OCR error correction in context. In *Proc of Very Large Corpora Work*, pages 88–100.