Computational Linguistics in the Netherlands Journal 8 (2018) 122–137 Submitted 06/2018; Published 12/2018

Detecting and correcting spelling errors in high-quality Dutch Wikipedia text

Merijn Beeksma[1] M.BEEKSMA@LET.RU.NL
Maarten van Gompel[1] M.VANGOMPEL@LET.RU.NL
Florian Kunneman[2] F.A.KUNNEMAN@UVT.NL
Louis Onrust[1] LOUISONRUST@GMAIL.COM
Bouke Regnerus[3] B.REGNERUS@STUDENT.UTWENTE.NL
Dennis Vinke[3] D.VINKE@STUDENT.UTWENTE.NL
Eduardo Brito[4,5] EDUARDO.ALFREDO.BRITO.CHACON@IAIS.FRAUNHOFER.DE
Christian Bauckhage[4,5] CHRISTIAN.BAUCKHAGE@IAIS.FRAUNHOFER.DE
Rafet Sifa[4,5] RAFET.SIFA@IAIS.FRAUNHOFER.DE

[1] *Radboud University Nijmegen, the Netherlands*

[2] *Tilburg University, the Netherlands*

[3] *University of Twente, the Netherlands*

[4] *Fraunhofer IAIS, Sankt Augustin, Germany*

[5] *Fraunhofer Center for Machine Learning, Germany*

Abstract

For the CLIN28 shared task, we evaluated systems for spelling correction of high-quality text. The task focused on detecting and correcting spelling errors in Dutch Wikipedia pages. Three teams took part in the task. We compared the performance of their systems to that of a baseline system, the Dutch spelling corrector Valkuil. We evaluated the systems' performance in terms of F1 score. Although two of the three participating systems performed well in the task of correcting spelling errors, error detection proved to be a challenging task, and without exception resulted in a high false positive rate. Therefore, the F1 score of the baseline was not improved upon. This paper elaborates on each team's approach to the task, and discusses the overall challenges of correcting high-quality text.

1. Introduction

The popularity of websites such as `DamnYouAutoCorrect.com` illustrate that although spelling correction and word prediction tools are widely used, users recognize that the 'corrections' are not always perfect. Automatic spelling error detection and correction has been the subject of research for decades, but the growing amount of automatically processed data and the increasing capabilities of computers (such as smart phones) motivate continuous development (Berck 2017). Although state of the art spell checkers perform reasonably well in everyday life, reaching high accuracy remains a challenging task. In addition to supporting text writers, spelling correction has become increasingly important because it increases the performance of natural language processing systems, if done well. Search engines and systems for text classification, summarization, semantic concept detection, morphological analysis, and part of speech tagging are all examples of systems for which the performance is influenced by the text quality.

The increased use of digitized databases and the Internet as a source of data has opened up all kinds of opportunities for data mining and machine learning, but also challenges the requirements of natural language processing tools. Different languages, text genres and knowledge domains are char-

acterized by their own words and conventions, thereby impeding the transfer of spelling correction systems across languages, genres and domains.

The Internet popularized new methods of creating text, in the form of wikis. A wiki is a collection of interlinked documents that are created and maintained as a collaborative effort, by several contributors and editors. The best-known and most widely used example of a wiki is Wikipedia. Anyone, not just experts, is allowed to contribute to Wikipedia. Therefore, a broad range of topics is described in many languages, and although the content has been ever-growing since the start of the project, Wikipedia remains to be well maintained. Instead of making it hard to add or edit a page by enforcing rules regarding text and information quality, Wikipedia made it easy to correct existing pages. Therefore, although Wikipedia allows itself to be imperfect (Wikipedia.org 2018), the Wikipedia pages of high-resource languages are generally characterized by high text quality.

In addition to being one of the most popular reference works on the Internet, Wikipedia is popular as a text corpus. Among other things, it is often used as main or background corpus for statistical language models, to align words in different languages, to train word embedding models, to explore semantic relations, and to build knowledge graphs. Spelling correction in Wikipedia articles improves the text quality for users, but also potentially aids the use of Wikipedia as a corpus for data mining and machine learning tasks. Especially in natural language processing pipelines, errors may accumulate quickly if misspellings are not corrected early on.

The CLIN28 shared task aimed to explore spelling detection and correction of high-quality Wikipedia pages. Because this shared task was organized in the context of the CLIN28 conference, and because Dutch is a high-resource language (and therefore one of the best represented languages in Wikipedia) we limited the task to the correction of *Dutch* Wikipedia pages. As an additional challenge, and to mimic common real-world scenarios, we did not provide the teams with labeled training data, but only with a small validation dataset to indicate the format of the test data. We compared the results of the three teams that participated in the task to an existing baseline system, to explore the strengths and weaknesses of the different approaches.

2. Related work

Traditionally, systems for spelling error detection and correction focused on comparing texts to lists of correct words, thereby verifying which words are correct, and which are possible errors. Using a list to detect spelling errors is not a sufficient approach, however, due to the productivity that is inherent to language: a list is never complete.

In the case of Dutch, we deal with a language that is highly productive in terms of deriving and compounding words. Words such as 'pompoenachtig' (*pumpkin-like*) or 'bureaulaconstructie' (*desk drawer construction*) are examples of proper Dutch words that are unlikely to appear on any word list. Early spelling correction systems for Dutch such as CORR^{ie} (Vosse 1994) were equipped with compound analyzers, which check whether an out-of-vocabulary word can be split into multiple known words, while taking into account constraints such as the length and frequency of the individual parts to avoid making implausible splits.

Verb inflection errors, or 'dt' errors, are a different category of errors in Dutch which often result in incorrect use of an existing word. Verb inflection errors are both common and hard to detect automatically. Due to phonological processes such as final devoicing and degemination, differently inflected verbs are often homophonous: words such as 'gebeurt' and 'gebeurd' (present singular second/third person inflection versus past participle of 'gebeuren', *to happen*) are frequently confused, as are 'word' and 'wordt' (present singular first versus second/third person inflection of 'worden', *to become*), or 'melden' en 'meldden' (plural present versus past inflection of 'melden', *to report*). To detect such 'confusibles', the context in which they appear needs to be taken into account. Systems such as CORR^{ie} handled such errors by analyzing the syntactic structure of a sentence in order to detect confusibles.

A drawback of approaches which rely heavily on linguistic rules or linguistic analyses is that they are highly language-specific. Alternatively, statistical approaches emerged to find and correct such errors. Statistical approaches provide a more flexible approach to spelling correction by focusing on detecting incorrect (use of) words rather than verifying whether words are correct. These approaches rely on character or word n-grams, and therefore typically require large amounts of data to derive reliable statistics from.

TISC (Text Induced Spelling Correction) is an example of a statistical, context-sensitive system, which uses a corpus-derived lexicon of word uni- and bigrams to detect errors and rank correction candidates (Reynaert 2004). Because the detection and correction of errors is led by n-gram statistics rather than linguistic rules or linguistic analyses, statistical systems have the potential to be language-independent. However, the most widely used spelling correctors today, such as Hunspell[1], usually combine a statistical approach with language-specific modules or vocabularies to tackle common errors.

Hunspell is integrated in OpenOffice, Mac OS X, Evernote, Firefox, Chrome, and Photoshop, among many other applications. It supports a broad range of languages, and combines generic modules with language-specific dictionaries and rules. Hunspell is incorporated in the modular framework Gecco[2], which too provides a language-independent, hybrid spelling correction system. Gecco offers a generic pipeline for spelling correction, and can be used to train and test a spelling corrector from scratch, given a dataset. Valkuil[3], a spell checker for Dutch, is powered by Gecco. To tune it for Dutch, the system is retrained on subsets of data to tackle common spelling errors in Dutch, such as verb inflection errors and homophone confusions.

Probabilistic methods for error correction follow naturally from statistical approaches (Kukich 1992). Methods for assessing which of the 'correction candidates' fits best given the context of a spelling error usually depend on transition or confusion probabilities, either on the character or word level. Many systems combine statistical data with machine learning algorithms such as classifiers in order to determine which candidate is the optimal candidate.

A leading example of a machine learning approach is the work of Golding and Roth (1999) who applied the Winnow algorithm to the task of correcting confusibles. The Winnow algorithm performs especially well on large feature sets, and generalizes well to unfamiliar data. A different machine learning approach is used in the work of Berck (2017) and Van den Bosch and Berck (2013), who used a memory-based language model in combination with a decision tree algorithm. The system predicts a distribution of words given a certain context, and uses this distribution to both detect and correct words: if a word is unexpected given the distribution, it may be an error.

In addition to general-purpose spelling correctors, a large number of special-purpose spell checkers has been created, for specific genres, domains, or users. Special-purpose spell checkers are trained to handle large amounts of noise, jargon, abbreviations, proper names, archaic spelling, expressions, or other features specific to the text type. As an example of a text type-specific spelling corrector, Text-Induced Corpus Clean-up (TICCL) (Reynaert 2010, Reynaert 2014) (the main component of PICCL[4], a workflow for corpus building), is designed specifically to correct noise that results from OCR. With regards to specific domains, spelling correction software has for example been created for clinical text (Fivez et al. 2017). We find examples of user-specific specialization in auto-correction software applications on smartphones, which typically learn from a user's input, and for example in Ghotit's dyslexia software (Ghotit 2011), which is specifically designed for people with dyslexia and other learning disabilities. Effort has been put into the creation of specialized systems that deal with specific spelling problems as well. We find an example of such work in the current issue of this journal: Heyman et al. use neural networks to detect and correct context-dependent dt errors in Dutch text (Heyman et al. 2018).

1. `https://github.com/hunspell`
2. `https://github.com/proycon/gecco`
3. `valkuil.net`
4. `https://github.com/LanguageMachines/PICCL`

Although the use of neural networks has been explored for subtasks of spelling correction, such as OCR correction, text-to-speech correction, and proper name correction (see Kukich (1992) for an extensive literature overview), so far neural networks, and deep learning in particular, have not received much attention in the domain of spelling correction, in contrast to many other domains. Examples of the use of sequential neural networks includes the work of Xie et al. (2016), who use an encoder-decoder recurrent neural network (RNN), which operates on the character level in order to handle out-of-vocabulary words. Less common is the use of convolutional neural networks (CNNs), as used by Cholempatt (2018), who shows that the network effectively captures local context with the use of word- or character-level embeddings as input features. Ghosh and Kristensson (2017) combine a character-level CNN with a gated recurrent unit (GRU) encoder along with a word-level GRU-based decoder to tackle the tasks of text correction and completion in keyboard decoding simultaneously. As a sequence-to-sequence network, the model encodes character-level hidden representations of an erroneous input sequence and then decodes the revised sequence, resulting in a corrected output string. Although the examples illustrate that deep learning techniques are in fact explored and seem to be feasible for the task at hand, the use of these techniques has yet to become a standard solution for spelling correction.

3. Method

For the annotated data, preprocessing and evaluation scripts, instructions for annotators, a detailed task description including examples of spelling errors, and team submissions, we refer the reader to `https://github.com/LanguageMachines/CLIN28_ST_spelling_correction`.

3.1 Data

To create an annotated corpus of Dutch Wikipedia pages, we downloaded the most recent data dump of Dutch Wikipedia pages (the data dump of 01-02-2018). Text edits are collected by Wikipedia in log files in order to keep track of changes. We searched the log files for strings indicating that a word was edited to correct a spelling error (e.g. Dutch equivalents of 'spelling correction' and 'spelling error') in a previous version of the page, and marked these edits as possible spelling corrections in both the edited version of the file and the pre-edited version of the file.

To obtain a clean set of data for annotation, we assessed the quality of the pages until we collected roughly 100 pages of good quality. We excluded pages which had no marks for possible spelling corrections, and pages with unusual amounts of urls, markup text, or other noise. A page could occur multiple times in the sample, because spelling edits could have been made at multiple moments in time. We accepted multiple versions of a page only in the cases in which substantial changes were made, compared to a different version of the same page.

3.2 Annotation

We enlisted a total of 6 annotators to mark and categorize the spelling errors in the Wikipedia pages. We annotated the documents in two rounds: in the first round, we annotated 20 documents to provide annotated validation data to the shared task participants, and in the second round, we annotated 50 documents as test data, which were released to the shared task participants without annotations. The documents were evenly distributed over the annotators. Each document was annotated by two annotators separately, and the annotations were checked and merged by a third annotator. In cases of clashing annotations (e.g. the annotation spans differed, or the assigned spelling categories for an error did not match), the third annotator consistently solved the conflicts. To further promote consistency, the third annotator was always the same person.

To facilitate the annotation process, we used FLAT[5] (FoLiA Linguistic Annotation Tool), a web-based linguistic annotation tool built upon an XML-based data format (Gompel and Reynaert 2013, Gompel 2017). The interface shows one document at a time, in which the possible spelling corrections that were found automatically during the data collection phase were marked to support the annotators. In many cases, a document contained more spelling errors than were initially marked, and some of the marked edits were not actually spelling errors. For this reason, the annotators were requested to read the full documents and highlight all spelling errors they could find, as long as they fitted one of the predefined spelling error categories. Annotators could highlight one word or a span of words, and were prompted to choose which type of spelling error they had marked.

3.3 Evaluation

We discriminated between two different tasks: error detection and error correction. We evaluated the participating systems for their performance on these two tasks separately. We evaluated the teams in terms of precision (i.e. true positive / (true positive + false positive)), recall (i.e. true positive / (true positive + false negative)), and F1 score (i.e. 2 * ((precision * recall) / (precision + recall))).

Systems were allowed to deliver multiple correction candidates ranked by a measure of certainty (instead of returning one solution per spelling error). For each case in which *one* correct solution was returned by a system, we upped the true positive count of that system by 1. In case *multiple* solutions were returned by a system, the measure of certainty corresponding to the correct solution was summed to a system's true positive count instead of 1. To rank the participating systems and determine which system performed best, we used the overall F1 score for all spelling categories and for detection and correction together. Finally, we compared the performance of the participating systems to the baseline system Valkuil, a spelling correction system for Dutch.

3.4 Spelling error categories

We aimed to focus spelling correction in general (rather than just non-words or confusibles for example), therefore we included a broad range of spelling errors in this task. We based the list of error categories on the list of errors our baseline system Valkuil is trained to solve, but made some adjustments to the list to better fit the goals of this task: 1) the original categories of missing and redundant punctuation touch upon a broad range of spelling phenomena which depend to a smaller or larger degree on personal preference. Therefore, we retain the category, but use it to annotate missing or redundant hyphens relating to compound words only. 2) We excluded the category of remaining/uncertain errors.

The annotation guidelines therefore instructed the annotators to mark and correct words or word groups which containing the following errors:

- archaic spelling: outdated spelling (e.g. *'aktie' → 'actie');
- capitalization error: incorrect use of capital letters (e.g. *'amsterdam' → 'Amsterdam');
- confusible: word would be a properly spelled word in another context, but is incorrectly used (e.g. *'ik wordt' → 'ik word');
- missing word: sentence is ungrammatical because a word is missing (e.g. *'met vrouw die' → 'met de vrouw die');
- redundant word: sentence is ungrammatical as the result of an extra/redundant word (e.g. *'door doordat' → 'doordat');

5. https://github.com/proycon/FLAT

- missing punctuation: a hyphen is missing (e.g. *'autoongeluk' → 'auto-ongeluk');
- redundant punctuation: there is a redundant hyphen (e.g. *'co-assistent' → 'coassistent');
- split error: incorrect spaces occur in a compound word (e.g. *'lang durig' → 'langdurig');
- run-on error: words are incorrectly linked together (e.g. *'etcetera' → 'et cetera');
- non-word error: word is not an existing Dutch word and would not be a proper result of productive linguistic processes (e.g. *'assrtief' → 'assertief').

The annotators were instructed *not* to mark the following cases:

- the word is correctly spelled, but alternative spelling variations would also be possible;
- the error is intentional (for example when a historic text is quoted);
- incorrect use of punctuation (other than hyphens) or spaces;
- markup text and symbols (HTML).

Table 1 lists the spelling error categories of interest, and the number of occurrences in the test data.

Category	**Occurrences**
archaic spelling	5
capitalization error	165
confusible	40
missing punctuation	18
missing word	21
non-word error	62
redundant punctuation	53
redundant word	2
run-on error	10
split error	123
Total	499

Table 1: Error types and corresponding frequencies in the test data.

4. System descriptions

Three teams participated in the shared task: Team FRAUNHOFER, Team COCOCLINSPCO, and Team ENSCHEDE. We compared the performance of the teams to a baseline system: VALKUIL. The following sections provide background information about the baseline system and the participating systems.

4.1 VALKUIL

We used VALKUIL, a statistical spelling corrector for Dutch, as a baseline system. Trained on a large amount of Dutch text, it applies several memory-based classification modules to an n-gram representation of the text data. It operates on the word level: the system scans the text word by word, and determines whether a correction is required based on the focus word and the surrounding context words. The system returns a distribution of possible corrections, which are either a word deletion, insertion, or replacement. The set of correction candidates is ranked by frequency of

occurrence in the training data. For a more detailed description, we refer the reader to Van den Bosch & Berck (2013) and to valkuil.net.

4.2 FRAUNHOFER

Most spell checkers generate a set of correction candidates by finding the subset of words that differ from the incorrect word less than a determined threshold for edit distance (Tijhuis 2014). In order to keep the computation time low enough for real-word applications, most of them find only the words that have an edit distance of 1 to the incorrect word. This strong limitation significantly reduces the probability of selecting a suitable correction candidate.

Team FRAUNHOFER proposes a novel approach[6] which entails the use of embeddings which are computed with kernel principal component analysis (KPCA). KPCA embeddings are continuous vector representations for non-numeric entities (such as words).

The approach enables the use of other similarity functions than just the edit distance, which only need to be computed for a small subset of the vocabulary. Therefore, the computation time is reduced (which keeps the computation time suitable for near real-time applications), while increasing the maximum 'allowed' edit distance.

The vocabulary consists of all the words belonging to *wdutch*, a word list from the OpenTaal project (available as a Debian package) which uses the official spelling of 2005, and which is officially approved by the TaalUnie. The team took the 3000 most frequent words from the list as the 'representative vocabulary' for the training phase.

The team trained the embedding model by using the homogeneous polynomial kernel of degree 2 on the representative vocabulary. They selected the 2000 principal components with highest eigenvalues as the embeddings for the representative vocabulary. To compute the embeddings, the team used KPCA, for which the dot product between data points is replaced by a predefined similarity function (Brito et al. 2017, Schölkopf et al. 1997). They used these vector representations together with the resulting projection matrix to generate KPCA embeddings for the rest of the vocabulary. The team found that using bigram similarity between two given words (Kondrak 2005) was the most suitable approach for this shared task.

The approach considered any word token *not* belonging to the vocabulary as a misspelling, unless any of the following conditions applied:

- the token is a number;
- the token ends with a dot (because of the assumption that it is likely to be an abbreviation);
- the token contains special characters other than '-' or '".

An example of KPCA embeddings can be seen in Figure 1. The team exploited the fact that a KPCA embedding can be inferred for any out-of-vocabulary word once the language model is learned: a KPCA embedding was inferred for each potential misspelling during the detection phase. The word that is the closest neighbor to the misspelling embedding (according to the cosine distance between the word vectors) is selected as correction candidate. Although the team only selects one word as correction candidate, the approach can be extended to provide a ranking of possible corrections by listing additional nearest neighbors sorted by the distance to the misspelling vector.

To prevent the system from regarding all out-of-vocabulary proper nouns as spelling errors, correction candidates were discarded when the cosine distance between the correction embedding and the presumed error was too large (> 0.2).

The most obvious limitation of this approach is that it only works for error categories related to the word form. No contextual information is taken into consideration, which makes the detection of confusibles impossible. Additionally, this approach leads to many false positives because

6. `https://github.com/fraunhofer-iais/kpca_embeddings`

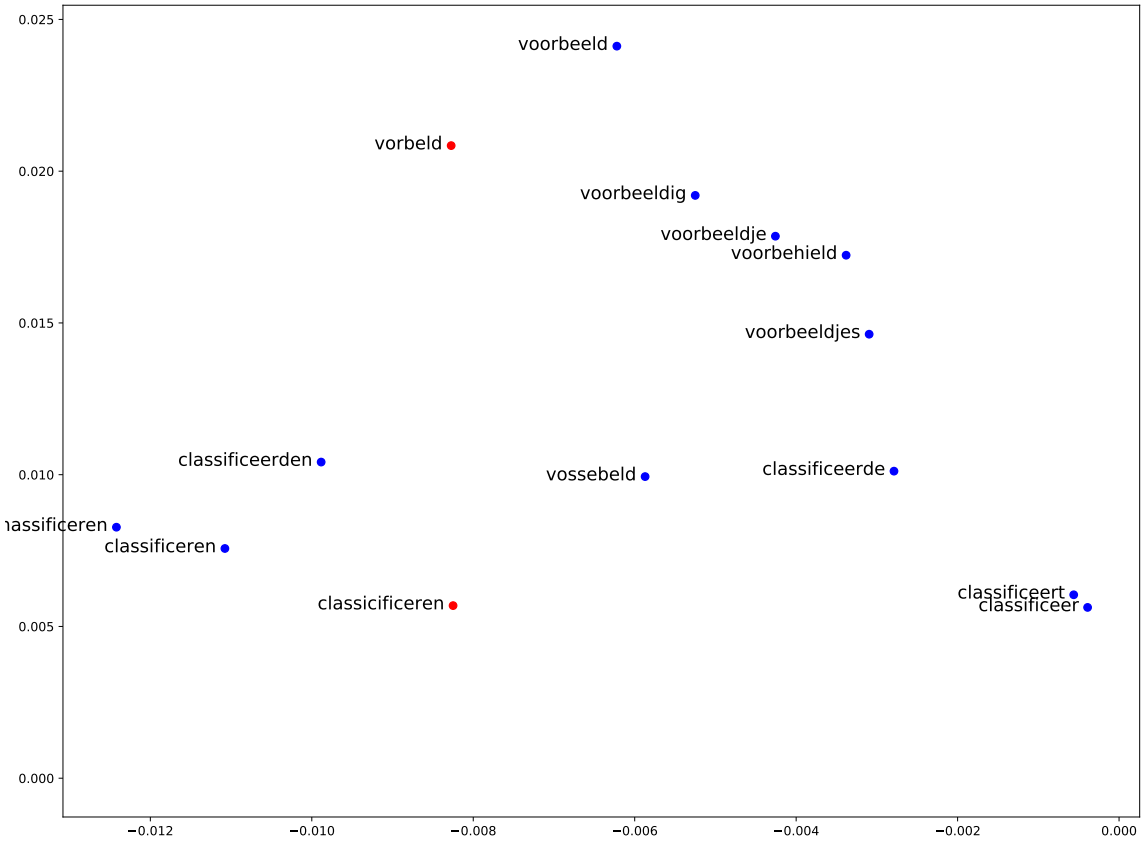

Figure 1: Visualization of the misspellings *'vorbeld' and *'classificiferen' and their nearest neighbors by plotting the two first components of their respective KPCA embeddings. The closest neighbors ('voorbeeld' (*example*) and 'classificeren' (*to classify*)) are in these cases indeed proper corrections of the misspelled words.

of capitalization errors. Integrating named entity recognition to ignore named entities and thereby circumventing the need of a threshold for corrections which are too distant (in terms of cosine similarity) from the misspelling, could potentially improve the results. Additionally, different ways to incorporate contextual information could be considered, in order to detect confusibles.

4.3 ENSCHEDE

Team ENSCHEDE explored the use of a character-level sequence-to-sequence neural network to approach the task (Vinke and Regnerus 2018). The network consisted of long short-term memory (LSTM) units, and used a sequence of input characters, such as letters, spaces and punctuation marks to convert a text with spelling errors to a corrected text. The network takes all the characters from an input sentence into consideration when determining what the most likely output of a predetermined character set should be. This means the characters put into the network before the current character influence the outcome of the network for the current character used as input. While similar LSTM architectures have been used for similar problems, such as translating archaic spellings to modern words (Bollmann and Søgaard 2016), it has not successfully been used as a basis for a spelling error detector and corrector.

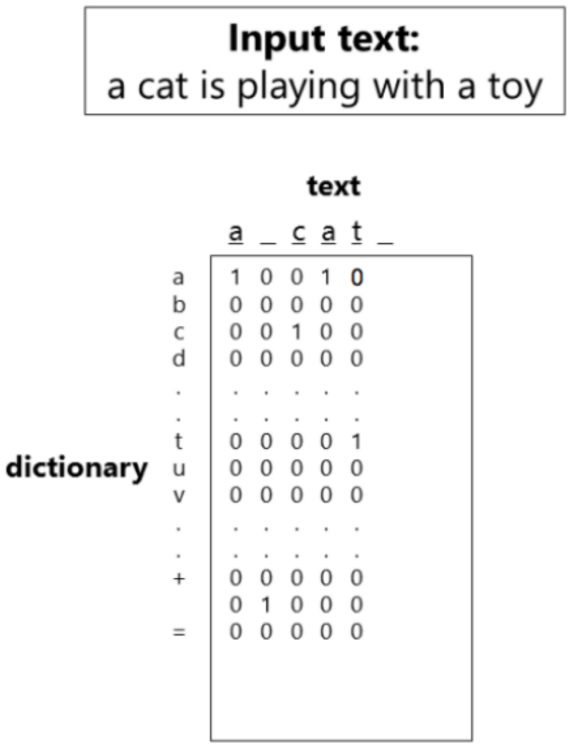

Figure 2: An example of one-hot encoded input features.

Training an LSTM network requires labeled data. Therefore, to train and test the LSTM-based error detector and corrector, the team used a Wikipedia dump containing roughly four million text documents. They divided the documents in sentences, removed the sentences that contained infrequent words, shuffled the sentences of all the documents, and extracted a sample of 100.000 sentences. It should be noted that tokenisation of the data set removed capital letters, numbers and punctuation marks, therefore the network is not capable of correcting errors regarding those types of characters.

To label the data, the team generated artificial errors and inserted them into the sentences. First, they simulated errors that are likely to be made when a keyboard is used to generate text (Grudin 1983, MacNeilage 1964). The most common error type for keyboard-generated text is that the target letter is replaced by a neighboring letter on the keyboard. As MacNeilage (1964) explains, when generating errors, the errors should not be introduced in every sentence, therefore the team introduced errors in 1% of the data. This approach mostly generated non-word errors, which is only one of the error categories in the challenge. Therefore, the team additionally generated errors by scraping Woorden.org (*Nederlands woordenboek* 2018), a Dutch website containing around 6000 grammar and spelling mistakes and corresponding corrections.

In order to use the data as input for the LSTM model, the team defined a character set, and used multiple one-hot encoded arrays (one array for each character) to represent the data, as illustrated in Figure 2. They used a sliding window of 60 characters that moved 20 characters per time step to handle sentences of different lengths. The probabilities from these windows were combined to calculate the probability of being a spelling error for each character.

The team tested several variations of an LSTM model with different parameter settings. The best-performing model contained two LSTM encoding and two decoding layers, each containing 128

LSTM units. No dropout was applied. The model was implemented with the neural network library Keras with Python.

The team notes that the system performance is negatively influenced by two obstacles: 1) the way the data is preprocessed (tokenization removed capital letters, numbers and punctuation marks), and 2) the way the network is trained. The LSTM network learns words based on the way characters are ordered. The network learns which characters are likely to follow each other in a word combination, but it does not know if the created word exists. The network does not treat spaces differently from other characters - spaces have no special status as word separators. As a result of this implementation, the network cannot learn the length of a word, and may learn character compositions which do not represent well-formed Dutch words. Resolving these obstacles remains a task for future work.

4.4 COCOCLINSPCO

COCOCLINSPCO[7] is a system for spelling correction based on a language model. Since the task provided validation, but not training material, and because this type of training material is generally not (publicly) available for Dutch, team COCOCLINSPCO resorted to an unsupervised method for spelling correction. Creating a training set had several drawbacks, according to the team: it is a time consuming process, and it would have been unlikely to result in a set of training material large enough to capture the delicacies of spelling errors. Injecting artificial spelling errors would create a bias, regarding both the error types, and the error frequency. By using unannotated material, the team could use an unrestricted amount of text. The approach assumes that the spelling errors inherent to large amounts of text are accompanied by a larger number of counter-examples.

Because the test set contained Wikipedia articles, the team decided to incorporate Wikipedia articles in the training material. They used two Dutch Wikipedia dumps, from 2013 and 2015. Additionally, they used material from the Flemish Mediargus corpus. Mediargus contains texts from news papers and magazines in Flanders from the last two decades[8]. Although there are some small differences between Flemish and Dutch, it was assumed that those would not be significant in terms of spelling errors. The dataset consisted of a total of 4.5 billion tokens.

To build a language model, the team used Colibri Core (Gompel and van den Bosch 2016), a C++ library that supports binary representations and informed iterative counting by means of word n-grams. They used a 5-gram language model, in which the first word pair formed the left context, the middle word was the focus word, and the last word pair formed the right context. For Colibri Core the team used a unigram threshold of 100, an n-gram threshold of 2, word n-grams of length 1 to 5, and an unindexed pattern model. This resulted in a model with 3 million types (and a runtime memory use of around 70GB).

This language model formed the foundation of four different correctors: a replacer, a splitter, an inserter, and an attacher. To find and correct the spelling errors, the team used stacked corrections. Each sentence is read from left to right with a sliding window of size 5. Within each window, each corrector aims to find and apply corrections. If an error is corrected, the system starts at the beginning of the sentence, and repeats the procedure.

The replacer finds a replaceable word c' in the window $a\_b\_c\_d\_e$ ('_' denotes a space) such that the frequency (normalized occurrence count for the length of the pattern) of $c' > tc$, where t is a manually chosen threshold. If no such a c' exist, the system backs off to smaller context windows: b_c_d and ultimately c without any context. Because this approach typically yielded many correction candidates, and given that most of these candidates were incorrect suggestions, the maximum allowed Levenshtein edit distance between the incorrect word and the correction candidate was set to 1. To find run-on errors, the splitter corrector was used to find c and d in the window $a\_b\_cd\_e$ such that

7. `https://github.com/naiaden/COCOCLINSPCO`
8. Mediargus is not publicly available. Although the Dutch Twente Newscorpus could have been used alternatively, this corpus is less up to date, and smaller in size. The team therefore opted to use Mediargus.

the frequency of $c\_d > cd$. Similarly, the inserter corrector finds a word f in the window $a\_b\_c\_d\_e$ such that the frequency of $c\_f\_d > c\_d$. The last corrector is the attacher, which looks for a word cd in the window $a\_b\_c\_d\_e$ such that the frequency of $cd > c\_d$.

5. Results

Tables 2 and 3 show the results per system and per spelling category in detail, for the detection and correction tasks respectively. The overall results are summarized in Tables 4 and 5. Although we evaluated the systems' performance in terms of precision, recall and F1 score, these evaluation metrics are not suitable to describe the results per error category in Tables 2 and 3, for the following reasons: 1) the teams were not asked to classify the errors into error categories, therefore we cannot assign false positives to error categories; 2) words marked as spelling errors during the detection phase are always corrected during the correction phase. Therefore, recall for the correction task would always be 1.0. Alternatively, we did not include false positives in Tables 2 and 3, but include the total numbers of false positives per system in Table 4. Additionally, we described the results of the correction task in terms of accuracy (i.e. the proportion of detected errors which was accurately corrected) in Table 3.

Category	**VALK**		**FRAU**		**ENSC**		**COCO**		**Average**		
(nr. of errors)	TP	FN	TP	FN	TP	FN	TP	FN	TP	FN	recall
non-word (62)	21	41	39	23	4	58	35	27	24.75	37.25	0.40
run-on (10)	3	7	5	5	0	10	6	4	3.50	6.50	0.35
capitalization (165)	0	165	117	48	4	161	58	107	44.75	120.25	0.27
split (123)	28	95	1	122	0	123	68	55	24.25	98.75	0.20
mis. punct. (18)	1	17	6	12	2	16	5	13	3.50	14.50	0.19
red. punct. (53)	3	50	13	40	1	52	20	33	9.25	43.75	0.17
confusible (40)	2	38	7	33	6	34	9	31	6.00	34.00	0.15
archaic (5)	0	5	1	4	0	5	1	4	0.50	4.50	0.10
mis. word (21)	0	21	0	21	0	21	1	20	0.25	20.75	0.01
red. word (2)	0	2	0	2	0	2	0	2	0.00	2.00	0.00
Total (499)	58	441	189	310	17	482	203	296	116.75	382.25	

Table 2: Performance per system in the detection phase per spelling error category. The 'average' columns summarize the performance of all systems, to enable comparison of performance per error category. In the 'total' line, the raw frequencies are summed. The error categories are sorted by decreasing recall. TP = true positive, FN = false negative.

As Table 2 shows, non-word errors are the best detected errors, followed by run-on errors and capitalization errors, while archaic spellings and missing and redundant words prove to be the hardest to detect. However, as shown by Table 3, the error categories that are most successfully detected, are not necessarily the easiest to correct. Split, capitalization and run-on errors have been solved most accurately by the systems on average.

The systems tend to miss a lot of errors (FN) in the detection phase. The amount of true errors that were actually found (TP) deviates a lot between systems, and also between error categories. For some of the error categories in the detection task, the systems of teams FRAUNHOFER and COCOCLINSPCO found more than half of the errors, in contrast to the other systems. In the correction task however, more error categories were accurately handled than in the detection task by all teams except team ENSCHEDE. COCOCLINSPCO and FRAUNHOFER in particular achieved high accuracy on multiple error categories.

When looking at the overall results in Table 4, it is striking that VALKUIL achieves the highest F1 score ($F1 = 0.17$), given that FRAUNHOFER ($F1 = 0.09$) and COCOCLINSPCO ($F1 = 0.06$) corrected more errors than VALKUIL, as is summarized in Table 5. VALKUIL's relatively high F1 score can however be explained by the amount of falsely detected errors (FP) of the systems. While COCOCLINSPCO and FRAUNHOFER corrected more errors successfully than VALKUIL, each participating team erroneously determined that hundreds or even thousands of *correctly* spelled words contained spelling errors, while VALKUIL only delivered 73 false positives. Due to a correction accuracy of 0, ENSCHEDE finally reached an F1 score of 0.

Category	**VALK**		**FRAU**		**ENSC**		**COCO**		**Average**		
(nr. of errors)	TP	FN	TP	FN	TP	FN	TP	FN	TP	FN	accuracy
split (123)	27	1	1	0	0	0	52.50	3.5	20.13	1.13	0.95
capitalization (165)	0	0	105	12	0	4	56	2	40.25	4.50	0.90
run-on (10)	3	0	3	2	0	0	6	0	3.00	0.50	0.86
non-word (62)	21	0	24	15	0	4	32	3	19.25	5.50	0.78
red. punct. (53)	0	3	8	5	0	1	14	4.5	5.50	3.38	0.62
mis. punct. (18)	1	0	3	3	0	2	2	2.5	1.50	1.88	0.44
confusible (40)	1	1	3	4	0	6	5	4	2.25	3.75	0.38
archaic (5)	0	0	1	0	0	0	1	0	0.50	0.00	1.00*
mis. word (21)	0	0	0	0	0	0	1	0	0.25	0.00	1.00*
red. word (2)	0	0	0	0	0	0	0	0	0.00	0.00	0.00
Total (499)	53	5	148	41	0	17	169.50	19.50	92.63	20.63	

Table 3: Performance per system in the correction phase per spelling error category. The 'average' columns summarize the performance of all systems, to enable comparison of performance per error category. 'Accuracy' denotes the number of true positive corrections divided by the number of detected errors for each spelling category. The error categories are sorted by decreasing accuracy. TP = true positive, FN = false negative.

* *We gave these results a low rank in the table, to avoid distorting the overall picture: the results are based on only one (in case of missing words) or two (in case of archaic spelling) cases.*

System	**Detection**						**Correction**			**Overall**		
	TP	FN	FP	rec.	prec.	F1	TP	FN	acc.	rec.	prec.	F1
VALK	58	441	73	0.12	0.44	0.18	53.00	5.00	0.91	0.11	0.40	0.17
FRAU	189	310	2693	0.38	0.07	0.11	148.00	41.00	0.78	0.30	0.05	0.09
COCO	203	296	5388	0.41	0.04	0.07	169.50	19.50	0.90	0.34	0.03	0.06
ENSC	17	482	932	0.03	0.02	0.02	0.00	17.00	0.00	0.00	0.00	0.00
Average	116.75	382.25	2271.5	0.23	0.14	0.10	92.63	20.63	0.65	0.19	0.12	0.08

Table 4: Summarization per system of the results per task (detection and correction) and averaged over the tasks. TP = true positive, FP = false positive, FN = false negative.

6. Conclusion and discussion

Three teams participated in this shared task about detecting and correcting several types of spelling errors in Dutch Wikipedia pages, which are typically characterized by high quality text. As an additional challenge, we did not provide annotated training data as part of the task. Even though

two of the three competing systems, FRAUNHOFER and COCOCLINSPCO, were able to recognize and correct a significant amount of errors, the baseline system VALKUIL delivered the highest F1 score. The following paragraphs discuss some concerns regarding prominent error categories included in this task, followed by concluding remarks about the individual systems.

System	Corrected (abs)	Corrected (rel)
COCO	169.5	34%
FRAU	148	30%
VALK	53	11%
ENSC	0	0%
Total / average	499	19%

Table 5: Successfully corrected errors per system.

6.1 A difficult task

To simulate a real-world scenario, we used Wikipedia text and included a broad range of spelling errors in this task. The most prevalent errors in the test data were capitalization errors (33%) and split errors (25%). The rules for capitalization and compounding as stated by the Woordenlijst Nederlandse Taal (Nederlandse Taalunie 2015), (which constitutes the official Dutch spelling), also called 'Groene Boekje' (*green book*), are notoriously complex. The total number of errors related to compounding and capitalization make up a large proportion of the total number of errors: capitalization errors (165), run-on errors (10), split errors (123), redundant punctuation (53), and missing punctuation (18), together form a total of 369 errors - 74% of the total number of errors.

Although the general 'rules of thumb' for capitalization and compounding may not seem complex, they are characterized by many exceptions, and are highly specific in some cases. The past two decades, spelling rules (and the revisions thereof) have led to controversy, and even to the creation of an alternative spelling guide: 'Spellingwijzer Onze Taal', or the 'Witte Boekje' (*white book*) (Genootschap Onze Taal 2015). This alternative spelling guide argues for the simplification of certain complex rules, and deviates from the rules of the official spelling in a number of ways - many of which regarding compounding and capitalization.

To give insight in the complexity of the rules, the reader may consider the following examples (explanation and examples were translated/paraphrased from the Woordenlijst Nederlandse Taal by the authors):

- Rule 16.3 states about capitalization that words that are used to refer to subpopulations typically are capitalized, even if a word is not derived from a geographical location, e.g. 'Nederlander' (*Dutch person*), 'Eskimo' (*Eskimo*). The words 'indianen' (*indigenous peoples of the Americas*) and 'zigeuners' (*gypsies*) are not capitalized, because these overarching terms refer to broader ethnical groups. When a reference is made to a religious group, such as 'christen' (*Christian*), the word is not capitalized either. Therefore, 'een dialoog tussen **christenen** en **joden**' (*a dialogue between Christians and Jews*) would be written without capitals, while 'een dialoog tussen **Joden** en Palestijnen' (*a dialogue between Jews and Palestines*) requires capitalization.

- Rule 16.4 states about capitalization that 'Middeleeuwen' (*Middle Ages*) is either to be capitalized or not, depending on the context: 'the word we use to refer to a period in time is not capitalized', the official rules state, but: 'this rule applies to common texts. Exceptions to the rule may occur in specialized publications'.

- Rule 6 for compounding in Dutch discusses the cases in which word groups and/or compound words should be written all together, or linked with spaces or hyphens. Rule 6.1 states to write

spaces between words in word groups ('vijftig euro' - *fifty euro*, '80 jaar' - *80 years*), while rule 6.2 states that no spaces should be used in compound words or derivations ('eurobiljetten' - *euro bills*, '80-jarige' - *someone who's 80 years old*). According to rule 6.7, the space in a word group is removed once a word group subsumes a compound word: one writes 'vijftig euro' and 'eurobiljetten', but 'vijftigeurobiljetten'. At least - in case one wants to refer to multiple euro bills which are all worth fifty euro. When one refers to fifty euro bills regardless of their worth, one would correctly spell 'vijftig eurobiljetten'. One might also opt for a different spelling, and use '50' instead of 'vijftig'. Exceptions to rule 6.7 are made when numerals are involved (rule 6.7.4.1): a space needs to be inserted in the compound if the numeral is written with numbers. Therefore, the compound must be written as '50 eurobiljetten' (*fifty euro bills*), regardless of whether the writer intends to speak of fifty bills worth x euro, or x bills worth 50 euro. To disambiguate between the different meanings, the alternative spelling of the Witte Boekje suggests to spell the word with a hyphen in case of the latter meaning: '50-eurobiljetten'. Finally, an exception to the exception is made (rule 6.7.4.4) in case a symbol is used in addition to the numeral, e.g. '€50': in this case, a hyphen is added to form the official spelling '€50-biljetten'.

Given the complexity of the official spelling and widely used alternative rules, it is to be expected that the Wikipedia pages contain many capitalization and compound errors: on the one hand such errors are common, and on the other hand they may be hard to detect. Additionally, it explains at least part of why none of the systems performed particularly good in this task: it was a hard task.

To determine whether or not the task was representative for a real-world setting depends on whether or not we consider the mentioned cases as important spelling errors or not. In case the goal is to develop a system which corrects texts in such a way that they adhere more closely to the official rules for spelling, the task can be considered representative. In case the goal is to correct text in such a way that most readers would consider the text to be correct however, we believe this task was not representative for a real-word setting. We believe that errors such as non-word errors and confusibles would be considered as more serious violations by most readers, because they result from breaking more systematic spelling rules (e.g. verb inflection errors), or are relatively easy to detect (e.g. non-words), compared to complex cases of compounding or capitalization.

6.2 Open data

Wikipedia is a open source of data. We could not (nor wanted to) restrict its use by the participating teams. Some of the teams indeed used Wikipedia data to train their systems. We did not try to check whether or not the data they extracted from Wikipedia overlapped with our test data, for three reasons: 1) we have no real means to check which data the systems are trained on, 2) Wikipedia pages are not annotated for spelling errors, and 3) the test set consisted of 50 Wikipedia pages, while the Wikipedia dumps used by the teams consisted of millions of documents. Therefore, we do not expect that any overlap between the Wikipedia data used by the teams and the test data impacted the systems' performance.

6.3 System performance

The higher overall performance of VALKUIL can be attributed mainly to the fact that the teams' systems without exception incurred many more false positives than VALKUIL did. This deviation results from the classic trade-off between high precision and high recall - a trade-off that should be kept in mind when tuning a system to a certain task.

The participating teams used a broad range of techniques to approach the task. Team ENSCHEDE explored the use of a character-level sequence-to-sequence neural network to approach the task, in line with recent developments in neural spelling correction. To handle the lack of labeled training data, the team generated artificial errors to train their model. Team COCOCLINSPCO

used a more traditional, unsupervised statistical approach using word n-gram representations of the text data. The team circumvented the need for labeled training data by exploiting language statistics derived from a large amount of text data. This approach however required significant computational memory and run time, thereby making it less suitable for practical applications on devices with limited computational capacities, such as smartphones. Team FRAUNHOFER reached the highest F1 score of the three participating teams, using novel techniques that make use of similarity rather than edit distance to correct words, thereby limiting computation time to a large extent and enabling real-world applicability.

On the one hand many errors were accurately corrected by all systems once they were found, but on the other hand the large amount of missed errors (false positives) and especially the high number of false alarms (false positives) lead to doubts about the feasibility of these systems in real-world applications. Depending on the language, genre, domain, and type of errors to correct, spelling correctors are more or less able to provide support to writers - they alert users of possible errors, and the users determine whether or not action is required. For NLP purposes however, which typically aim to reduce the need for manual intervention as much as possible, automatic spelling correction is far from a guarantee for error-free text. Although VALKUIL proved to be a strong baseline, its F1 score of 0.17 clearly illustrates that spelling correction is not a solved problem yet. Spelling correction therefore remains an interesting and important field of research, and still leaves room for the exploration of novel techniques.

7. Acknowledgments

We want to express many thanks to the annotators, whose meticulous analyses of the Wikipedia pages enabled this shared task.

References

Berck, Peter (2017), *Memory-based text correction*, PhD thesis.

Bollmann, Marcel and Anders Søgaard (2016), Improving historical spelling normalization with bi-directional LSTMs and multi-task learning.

Bosch, Antal van den and Peter Berck (2013), Memory-based Grammatical Error Correction, *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task* (August 2013), pp. 102–108. http://aclweb.org/anthology/W13-3614.

Brito, Eduardo, Rafet Sifa, and Christian Bauckhage (2017), KPCA embeddings: an unsupervised approach to learn vector representations of finite domain sequences, *LWDA*, pp. 87–96.

Chollampatt, Shamil and Hwee Tou Ng (2018), A Multilayer Convolutional Encoder-Decoder Neural Network for Grammatical Error Correction. http://arxiv.org/abs/1801.08831.

Fivez, Pieter, Simon Šuster, and Walter Daelemans (2017), Unsupervised context-sensitive spelling correction of English and Dutch clinical free-text with word and character n-gram embeddings, *Computational Linguistics in the Netherlands Journal* **7** (2015), pp. 39–52.

Genootschap Onze Taal (2015), *Spellingwijzer Onze Taal*, Prisma.

Ghosh, Shaona and Per Ola Kristensson (2017), Neural networks for text correction and completion in keyboard decoding, **14** (8), pp. 1–14. http://arxiv.org/abs/1709.06429.

Ghotit (2011), Ghotit Dyslexia Contextual Spell Checker. http://www.ghotit.com/home.shtml.

Golding, Andrew R. and D. Roth (1999), Applying Winnow to context-sensitive spelling correction, *Machine Learning* (34), pp. 107–130.

Gompel, Maarten van (2017), FoLiA: Format for Linguistic Annotation. Documentation, *Language and Speech Technology Technical Report Series LST-14-01* pp. 0–157. https://proycon.github.io/folia/.

Gompel, Maarten van and Antal van den Bosch (2016), Efficient n-gram, skipgram and flexgram modelling with Colibri Core, *Journal of Open Research Software*, Ubiquity Press.

Gompel, Maarten van and Martin Reynaert (2013), Folia: A practical xml format for linguistic annotation-a descriptive and comparative study, *Computational Linguistics in the Netherlands Journal* **3**, pp. 63–81, Citeseer.

Grudin, Jonathan T. (1983), *Error Patterns in Novice and Skilled Transcription Typing.*

Heyman, Geert, Ivan Vuliç, Yannick Laevaert, and Marie-Francine Moens (2018), Automatic detection and correction of context-dependent dt-mistakes using neural networks, Vol. 8.

Kondrak, Grzegorz (2005), N-gram similarity and distance, *String processing and information retrieval*, Springer, pp. 115–126.

Kukich, Karen (1992), Technique for automatically correcting words in text, *ACM Computing Surveys* **24** (4), pp. 377–439. http://portal.acm.org/citation.cfm?doid=146370.146380.

MacNeilage, Peter F. (1964), Typing errors as clues to serial ordering mechanisms in language behaviour., *Language and speech* (7), pp. 144–159.

Nederlandse Taalunie (2015), *Het groene boekje: Woordenlijst Nederlandse taal*, Van Dale Uitgevers.

Nederlands woordenboek (2018). http://www.woorden.org.

Reynaert, Martin (2004), Text induced spelling correction, *Proceedings of the 20th international conference on Computational Linguistics*, Association for Computational Linguistics, p. 834.

Reynaert, Martin (2010), Character confusion versus focus word-based correction of spelling and ocr variants in corpora, *International Journal on Document Analysis and Recognition* pp. 1–15, Springer Berlin / Heidelberg. 10.1007/s10032-010-0133-5. http://dx.doi.org/10.1007/s10.

Reynaert, Martin (2014), TICCLops: Text-Induced Corpus Clean-up as online processing system, *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations* pp. 52–56. http://repository.ubn.ru.nl/bitstream/handle/2066/93611/93611.pdf?sequence=1.

Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller (1997), Kernel principal component analysis, *International Conference on Artificial Neural Networks*, Springer, pp. 583–588.

Tijhuis, Lars (2014), Context-based spelling correction for the Dutch language: Applied on spelling errors extracted from the Dutch Wikipedia revision history, *Unpublished master's thesis.*

Vinke, Dennis and Bouke Regnerus (2018), Spelling error detection and correction using bidirectional long short-term memory networks.

Vosse, Theo G (1994), *The word connection*, PhD thesis, Doctoral dissertation, University of Leiden.

Wikipedia.org (2018), Wikipedia: Introduction. https://en.wikipedia.org/wiki/Wikipedia:Introduction.

Xie, Ziang, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng (2016), Neural Language Correction with Character-Based Attention. http://arxiv.org/abs/1603.09727.