Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), pages 2424–2430
Marseille, 11–16 May 2020

GM-RKB WikiText Error Correction Task and Baselines

Gabor Melli, Abdelrhman Eldallal, Bassim Lazem, Olga Moreira
gmelli@acm.org, abdelrhman.d@aucegypt.edu, lazem@live.com, olga.moreira@gmail.com

Abstract

We introduce the *GM-RKB WikiText Error Correction Task* for the automatic detection and correction of typographical errors in WikiText annotated pages. The included corpus is based on a snapshot of the GM-RKB domain-specific semantic wiki consisting of a large collection of concepts, personages, and publications primary centered on data mining and machine learning research topics. Numerous Wikipedia pages were also included as additional training data in the task's evaluation process. The corpus was then automatically updated to synthetically include realistic errors to produce a training and evaluation ground truth comparison. We designed and evaluated two supervised baseline *WikiFixer* error correction methods: (1) a naive approach based on a maximum likelihood character-level language model; (2) and an advanced model based on a sequence-to-sequence (seq2seq) neural network architecture. Both error correction models operated at a character level. When compared against an off-the-shelf word-level spell checker these methods showed a significant improvement in the task's performance – with the seq2seq-based model correcting a higher number of errors than it introduced. Finally, we published our data and code at https://github.com/GM-RKB/LREC-2020.

Keywords: Wiki Text, Typographical Error Correction, Character-level Language Models, seq2seq-based Error Correction

1. Introduction

Wikis are one of the most popular resources for online collaboration and knowledge exchange on which users can write and edit content directly through a web browser. Because wiki software engines are easy to use and deploy, they have become the natural choice for creating and growing a large number of modern knowledge bases, dictionaries, academic libraries, and encyclopedias. Mediawiki, the wiki engine that runs the well-known Wikipedia (Junghans et al., 2008; Krötzsch et al., 2006), for instance, currently powers a multitude of interactive websites including the online semantic wiki that this paper is focused on, the GM-RKB [1].
Similar to the creation of websites in the early days of the Web; when websites content was crafted with HTML code by hand than transformed by a browser rendering engine into web pages, wikis content is created using WikiText (Dohrn and Riehle, 2011) written and edited predominantly by humans then parsed and rendered by a wiki engine (Dohrn and Riehle, 2011; Junghans et al., 2008). WikiText is a simplified markup language that facilitates annotation of text documents. To create an internal link between annotated words (concept mentions) and a target wiki pages in Mediawiki markup, the editor simply needs to use double square brackets ("[[" and "]]"). For instance, the following wikitext:

```
"A [[Character-Level Seq2Seq Training
Algorithm|character-level seq2seq
algorithm]] is a [[seq2seq algorithm]]
that is a [[character-level NNet
algorithm]]."
```

will be displayed as "*A character-level seq2seq algorithm is a seq2seq algorithm that is a character-level NNet algorithm*" and link the concept mentions to the corresponding wiki entries.

[1] http://GM-RKB.com

GM-RKB is a semi-structured, domain-specific, and linguistically-rich online semantic wiki primarily focused on describing concepts, researchers, and methodologies reported in scientific literature on machine learning, data mining, statistics, mathematics, and physics. It systematically describes concepts related to computing tasks, systems and algorithms, along with their input/output data type and requirements. GM-RKB concept pages are designed to improve human and machine readability using a controlled English vocabulary and the structure proposed in (Melli and McQuinn, 2008).
The text of scientific publications included in the GM-RKB corpus have been semantically annotated and concept mentions linked to their respective wiki entries using Mediawiki markup language. Up to recently, the GM-RKB text annotation task was performed by 2 main agents:

1. a SDOI system's mention recognizer that automatically pre-annotated a text input by applying a trained conditional random field (CRF)-based chunker that (Melli, 2012);

2. a human editor who reviewed each wiki page to remove any WikiText errors and add domain-specific repairs.

Despite WikiText's intended simplicity, human editors are prone to introduce typographical mistakes in their annotations resulting in misspelled words and markup language errors. Since first introduced by Melli (2010), GM-RKB has grown to a semantic wiki that interlinks over 10,000 domain-specific concepts to a corpus containing over 1,000 publications. Correcting wikitext errors manually without the aid of automatic procedures was destined to be a laborious and time-consuming task. Developing a wikifixer tool promised to be not only time-saving but crucial for GM-RKB's continuous growth.
We designed and implemented the *GM-RKB WikiText Error Correction (WEC) Task* to benchmark systems that attempt to automatically recognize and repair simple typographi-

cal errors in WikiText based on frequent patterns observed in the corpus. The task consisted in conducting a series of experiments on benchmark datasets to find the best performing WEC system. We adopted a precision-based performance metric because we were interested in measuring of the balance between the welcome benefit a WEC system succeeding in repairing an error correctly against the significant cost of it introducing an error which requires to be repaired manually. We compared the relative performance of a character MLE Language Model-based and a sequence-to-sequence (seq2seq) neural network-based WEC, as well as two spelling error correction systems trained on GM-RKB and Wikipedia corpora datasets. Because of the difficulty in logging real wikitext errors introduced by human editors, we developed a sub-system that artificially can add human-like editing errors to the original text and convert it to training data.
The paper is structured as follow as:

- Section 2 provides a brief review of previous related research and lays down the foundations for the work presented in this paper;
- Section 3 describes the different stages of our task, as well as datasets and the precision-based performance metric used in our evaluation of the selected WEC systems;
- Section 4 describes the speller checkers, character-level MLE Language Model-based and seq2seq neural network-based models;
- Section 5 reports the series of experiments we conducted for evaluation of these WEC systems and respective results;
- Section 6 analyses and discusses our results;
- Section 7 draws conclusions about our findings and how they contribute to the recent and future development of WEC systems.

Our data and code can be found at https://github.com/GM-RKB/LREC-2020.

2. Related Work

Numerous research studies have been done on typographical spelling error correction (Mays et al., 1991). The most classical approach is the manual design of grammar rules as shown in Mozgovoy (2011). These hand-crafted rules are created by linguistic experts and used to match text to spot and correct any error. More advanced techniques are introduced through the use of machine learning for tackling this task. Mays et al. (1991) presented one of the earliest approaches using language models and by performing statistical analysis such as maximum likelihood techniques. These models use an annotated corpus which in turn is used on the text to automatically detect and correct spelling errors (Soni and Thakur, 2018).
Recently, many studies formulated the task of spelling correction as machine translation task, where erroneous text is considered as the source language and the correct text as the target language. This opens the gate to utilizing more advanced techniques such as neural networks. Notably the sequence to sequence encoder-decoder model which is mainly used for machine translation (Sutskever et al., 2014) is successfully utilized in several grammatical error correction tasks (Chollampatt and Ng, 2018). While effective, this supervised approach requires a large amount of clean annotated parallel training corpus to yield an adequate performance. That is why the method of augmenting the data by incorporating synthetic noise has been widely studied (Etoori et al., 2018).
Although the GM-RKB WEC Task is influenced by the task of spelling correction, the wiki markup is different from plain natural language text. The work done in (Junghans et al., 2008) shows the challenges dealing with such corpus from processing natural text. Moreover, the semi-structured nature of the WikiText markup is a fertile ground to utilize natural language processing (NLP) techniques for other similar tasks. For example, Dang and Ignat (2016) used deep neural networks for assessing the quality of Wikipedia articles, whereas Chisholm et al. (2017) used the generative ability of language modeling combined with the sequence-to-sequence architecture to generate one-sentence biographies from Wikipedia text.

3. Task Description

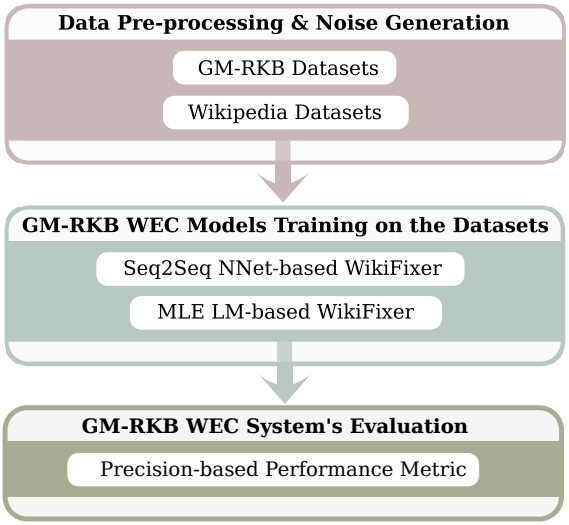

Figure 1: GM-RKB WikiText Error Correction Task

The GM-RKB WEC Task can be divided in 3 main subtasks: (1) the creation and preparation of the training dataset; (2) the training of WEC models on the datasets; and (3) the analysis of the relative performance of WEC systems. Fig. 1 shows a schematic view of our benchmarking process.

3.1. Datasets

The *GM-RKB WikiText Error Correction Task* focuses on fixing errors in Wiki pages. In order to help with such task, we created multiple datasets for both training error-fixing systems and for evaluating these systems. There are two main types of datasets. One type is based on snapshot of GM-RKB and the other type is based on Wikipedia pages.

Pre-processing and Error Generation
The process of creating a new dataset for the task starts by parsing the XML file containing Wiki pages. These XML snapshots contains WikiText source (language markup) and the metadata of each page. After parsing the original content of each Wiki page, we start adding noise to the text of each page which is written in WikiText markup language. Given the absence of manually corrected examples, we have created a tool to synthetically introduce errors. Four types of character-level errors are introduced: "Changed Character", "Deleted Character", "Inserted Character", and "Swapped Characters". When a new character is inserted as noise, it is selected at random from an ASCII non-numeric alphabet. Over time this random selection will be adjusted to better simulate naturally occurring errors.
We created a Python library to automatically parse the XML Wiki snapshot, extract the needed information and the original content, as well as add noise to the original content and store all the data in a "Parquet" file. These files became our main data repository. The "Parquet" file contains the original content of each page, the version with noise added, a unique ID number, the size of the page in number of characters, and a 'k' value from 0 to 10 to be used with 10-fold cross validation. The noise can be controlled by multiple variables by indicating the level of total noise and the ratio of each type of noise added.

GM-RKB Datasets
The main dataset of the task is the GM-RKB Wiki dataset. It is created using a snapshot of GM-RKB wiki pages in MediaWiki XML format. The snapshot was extracted in October 2019. These pages contain predominantly English-written text in Mediawiki annotation format. The current version of corpus consists of 106,111 MediaWiki-formatted pages from the GM-RKB site. It has an average page size of 976 characters.

Wikipedia Datasets
We used MediaWiki API, a web service that allows us to download Wikipedia pages as XML files. It provides multiple actions and filters to allow users to control the content downloaded. We have created two Wikipedia datasets. The first dataset is the Wikipedia training dataset containing a total of 35,000 wiki pages. We run the API to create a list of random pages using a special query. We did not specify any other filters or constraints. The dataset has an average page size of 7,252 characters and a total size of 253,835,323 characters. The second Wikipedia dataset was downloaded for testing purposes. We extracted a list of 5000 random page titles from Wikipedia using the API and extracted the content of the pages as well. The data set has an average page size of 6,164 characters, and the total size of 34,700,225 characters. The two Wikipedia datasets are different as both were created using two different random lists of pages.

Datasets Features
GM-RKB and Wikipedia datasets both have English Wiki pages written in Wiki markup language. They both have similar annotation syntax, which can mean they have the same notion of corrected text. On the other hand, they have different average page size. The Wikipedia dataset has an average page size of 7,252 characters, while, GM-RKB dataset has an average page size of 976 characters. Out of the 10,000 most frequent space-separated texts (words), only 3,700 are common between the two datasets. These words are often ordered completely different in the files.

3.2. Evaluation

We considered a special evaluation metric that measures the changes in text repairs performed on the Wiki pages by rewarding correctly fixed errors and penalizing system added errors. Ultimately, the metric should be well-suited at quantifying the detection of orthographic errors rather than grammatical errors. It should be able to operate at the character-level (rather than at a token or word-level) because errors sometimes include space ' ' characters and newline ' \n' characters ("H_ere is tw\no errors."). The adopted evaluation metric treats the task as a (near-binary) classification problem. Each character in the WEC system's output text is grouped into one of the following five outcomes:

- **True Positive (TP):** the source character was an error; the model correctly fixed it.
- **False Positive (FP):** although the source character was not an error; however, the model tried to fix it introducing a new error instead.
- **True Negative (TN):** the source character was not an error; the model correctly disregarded it as an error and did not try to fix it.
- **False Negative (FN):** the source character was an error; however, the model failed to detect it and processed the error as the correct character.
- **Detected Not-Fixed (DN):** the source character was an error; the model successfully detected the error but failed to correct it, or used the wrong character to fix it. (e.g. the correct repair was changing 'a' to 'b' but the model changed 'b' to 'c' instead of 'a').

We focused on only two of the five classes of TPs and FPs. That is, we focused on the two classes that can help us measure the model's precision. The evaluation score is calculated as follows.

$$Evaluation\ Score = 1 \times TP\ count - 5 \times FP\ count \quad (1)$$

In principle, when training a new model using a labeled dataset, this metric should reward correctly fixed errors and penalize incorrectly introduced errors.
We have developed a system to help with the evaluation process by comparing any two texts and creating logs representing the differences between the two texts. For evaluation, we compare the logs representing the differences between the original and noise text, to the logs between fixed and noise text . For example, to compare the sentences `A Character-Level Seq2Seq Training Algorithm` and `A Character-Leel Seq2Seq Trianing Algorithm` where we deleted one character from the word `Level` and swapped two

characters in the word `Training`, the system outputs following log: `[{'type': delete, 'pos': 14, 'chars': ['v']}, {'type': swap, 'pos': 28, 'chars': ['a', 'i']}]`. The *type* attribute represents the specific of difference in text files, for example, *delete* means a character was deleted. The attribute *pos* is the new position of a character or the first character changed in case of *swap* difference. The attribute *chars* is the list of the characters changed, swapped, deleted, or inserted. We use this model to create logs between the original text and the noisy text, then compare these logs to the ones created from comparing text with noise to fixed ones. These logs allow us to accurately evaluate and analyze the WEC models.

4. WikiText Repairing Tools

4.1. Spelling Checkers

Although the task of correcting natural language human-written text is different from that of correcting Wiki pages, we tested and compared spelling correction tools for performance evaluation purposes. We tested JamSpell, a Python library that checks and corrects spelling in text, and Pypyenchant similar spelling tool. JamSepll library has the full sentence as input and considers the context.

4.2. A Character-level MLE Language Model-based WikiFixer

WikiFixer is a tool that automatically repairs simple typos in WikiText based on patterns found in a related corpus. Currently, this system is focused on solving the GM-RKB WikiText Error Correction Task. The *Character-level MLE Language Model-based WikiFixer* is a data-driven implementation of this tool that is based on a simple function based on a character-level language model (in the form of a lookup table) that predicts the likelihood score for short sub-strings of characters. Currently, the function is trained on using the GM-RKB dataset. This statistical approach assumes that *N-gram* is a sequence of *N* characters. The trained language model can both predict the probability of certain *N-gram* appearing in the text or the probability of certain character appearing after certain *N-gram* (Norvig, 2007). The MLE WikiFixer uses number of similarity probabilities to detect noise and selects the error correction candidates with the highest probability. The model is controlled by setting thresholds for detecting noise and accepting an error correction action. This system is being used as a baseline for more sophisticated approaches such as the use of Neural Networks methods. Additional information of this baseline method can be found in an online document [2].

4.3. A Seq2Seq NNet-based WikiFixer

The final baseline WikiFixer included in this task was based on the sequence-to-sequence (seq2seq) neural networks model which 'translates' partially noisy sequences to corrected ones. Our seq2seq model can map a fixed-length input sequence with a fixed-length output. Input and output text lengths may differ. We were motivated by the approach proposed by Chollampatt and Ng (2018) for grammatical error correction, but at the character-level as described in Weiss (2016). The model consists of 3 main components: Encoder, Encoder Vector, and Decoder as illustrated in Fig. 2. Encoder is a stack of several recurrent layers where each accepts a single element of the input sequence, collects information for that element, and propagates it forward. In the WikiFixer problem, the input sequence is a collection of all characters from the noisy WikiText. The final hidden state produced from the encoder component is called Encoder Vector. This vector is constructed to represent the information for all input elements in order to help the decoder make accurate predictions. It acts as the initial hidden state of the decoder component of the model. Decoder is a stack of several recurrent layers where each predicts an output element at a time step. Each recurrent unit accepts a hidden state from the previous unit, produces an output as well as its own hidden state. In the WikiFixer problem, the output sequence is a collection of all characters of the fixed text. The recurrent units used are LSTMs. It's an extension for recurrent neural networks built to enhance their memory capacity.

5. Experiments

5.1. Description

We conducted several experiments to benchmark the performance of all WikiText repairing tools described in Sec.4. We trained the MLE Language Model-based WikiFixer on the GM-RKB dataset and 4 different versions of Seq2Seq NNet-based WikiFixer. WikiFixer NNet-GM-RKB is a version trained from scratch using GM-RKB dataset only. WikiFixer NNet-Wikipedia was trained using Wikipedia Dataset in similar manner. We adopted a transfer learning technique to train one of the NNet based WikiFixer versions. Transfer learning have been used mainly with Convolutional Neural Networks, where a pre-trained model with a large dataset is used as initialization for a different model (Singh and Garzon, 2015). We used the Wikipedia model as the largest dataset to create a pre-trained model. We used this model as initialization and retrained the same network using the GM-RKB dataset. Data Augmentation is another method that can improve NNet models when there is insufficient training data available (Singh and Garzon, 2015). We augmented the GM-RKB dataset with 7,000 random wiki pages from Wikipedia. The final WikiFixer NNet model was trained from scratch using this augmented dataset. We tested each model using the part of the GM-RKB dataset that was not included in the training process of any of the models. We also tested the models using a large dataset created with 5,000 random Wikipedia pages. The wiki pages of Wikipedia test set are different from the pages used in our training process.

5.2. Results

Tab 1 and 2 summarizes the task results. It shows the number of TPs, FPs and the Eq.1 performances score for all the WikiText repairing tools described in Sec. 3.1 trained and tested on GM-RKB and Wikipedia datasets described in Sec.4

[2] https://tinyurl.com/mle-wikifixer

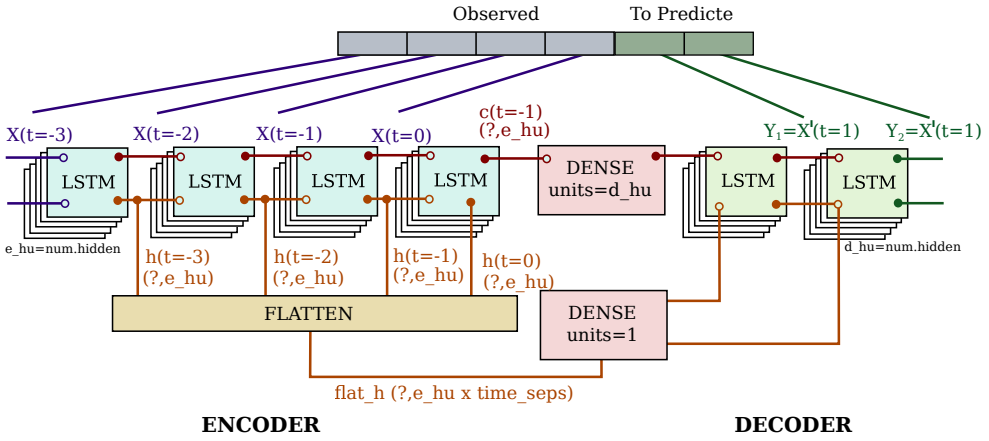

Figure 2: A Seq2Seq NNet-based WikiFixer Architecture.

Model	TP	FP	Score
JamSpell	18,324	460,916	−2,286,256
Pyenchant	18,630	4,717,170	−23,567,220
WikiFixer MLE	9,838	449	7,593
WikiFixer NNet GM-RKB	16,061	696	12,581
WikiFixer NNet Wikipedia	8,678	524	6,058
Wikifixer NNet Wikipedia pre-trained + GM-RKB	13,841	490	11,391
Wikifixer NNet Wikipedia 7,000 pages+GM-RKB	16,003	652	12,743

Table 1: GM-RKB Testing Dataset Results

Model	TP	FP	Score
JamSpell	11,479	312,809	−1,552,566
Pyenchant	9,656	8,351,825	−41,749,469
WikiFixer MLE	252	166	−578
WikiFixer NNet GM-RKB	3,954	287	2,519
WikiFixer NNet Wikipedia	6,385	211	5,330
Wikifixer NNet Wikipedia pre-trained + GM-RKB	3,284	160	2,484
Wikifixer NNet Wikipedia 7,000 pages+GM-RKB	6,056	277	4,671

Table 2: Wikipedia Testing Dataset Results

6. Discussion and Analysis

A straightforward application of the spell checkers to wiki markup text introduced a significant number of errors because these checkers rely on know-words dictionaries as well as on whitespace-based tokenization. The absence or addition of whitespaces produced additional nuanced errors as we demonstrate in the following example:

“<BExample(s):</B>”
⇓
“<B>Example:</B>”

Since the word ”Example” is not separated with white spaces, the normal spelling checkers cannot correctly tokenize and process such text.This is an example of the problem that would be created in case of the absence of white spaces. All the errors that were introduced to the wiki text in the training and testing datasets were created randomly, in terms of error type and position in the text, and they were also labeled automatically. We did not control the errors created to ensure the generality of the model.

Tables 1 and 2 show that spelling correction tools Jamspell, and Enchant failed to achieve acceptable results for the task. Although spelling tools had on average the highest rates of True Positives, they also had the highest rates for False Positives. This means that they added considerably more errors than the ones they fixed. They were good at fixing space-separated text (words), but this was not the goal of the task. That is why they failed to attain the same performance as the baseline WikiFixer MLE model. The score of both spelling correction tools, when tested with the two datasets, was negatively large which suggests these introduced more noise to the WikiText data. This result demonstrates how distinct the task of spelling correction is from that of fixing noise in Wiki pages.

The following example illustrates the limitation of these spelling correction tools when dealing with WikiText errors:

Original WikiText:

```
A [[semantic wiki]]
```

WikiText After Adding Noise:

```
A [s[emantic wiki]]
```

Spelling Tools Correction:

```
A [s[semantic wiki]]
```

They often fail to correct the error in the WikiText by swapping 's' and '['.

The Wikifixer MLE model scored positive when tested on the GM-RKB datasets, but it dropped to negative values when tested on the Wikipedia datasets. Nevertheless, it performed significantly better than spelling correction tools overall. The MLE model fixed errors more than it added noise, but not sufficiently high enough to score positive in case of Wikipedia data. Since MLE is a statistical model, this proves that GM-RKB pages, although having some statistical similarities and other similar features, are different from Wikipedia pages. This means each dataset would have some unique features as described in Section 3.1. This means each dataset described in Section 3.1 has some unique features that need to be captured to develop a good predictive model.

WikiFixer NNet models performed exceptionally well overall, scoring remarkably higher TP than FP instances. The model trained on GM-RKB performed considerably better but also produced a higher number of FPs than the WikiFixer MLE model. This was mainly due to overfitting on the training data. For this same reason, the model's performance dropped when tested on Wikipedia datasets scoring the highest FP number among all the WikiFixer models. The NNet model trained on Wikipedia datasets achieved the highest performance score but lower score on GM-RKB datasets. As in the previous case, the WikiFixer was overfitted on the Wikipedia dataset. When pretrained on Wikipedia and retrained on GM-RKB datasets, the WikiFixer NNet model achieved higher scores than MLE tested on these two datasets. However, these scores are still slightly lower compared to the other NNet-based models. The NNet model trained on GM-RKB datasets along with 7,000 pages from Wikipedia, scored the highest when tested on GM-RKB datasets and the second-highest when tested on the Wikipedia dataset. The 7,000 Wikipedia pages inclusion solved the overfitting problem during training. Moreover, the model became more versatile as GM-RKB and Wikipedia pages have distinct features.

7. Conclusions and Future Work

We benchmarked two baseline models for the detection and correction of typographical errors in WikiText annotated pages: Seq2Seq NNet-based; and Character-level MLE Language Model-based WikiFixers, along with JamSpell and Pyenchant spelling checker training on datasets from Wikipedia and domain-specific GM-RKB corpora. We artificially added character-level noise to data to simulated WikiText human-editing errors (Changed Character, Deleted Character, Inserted Character, and Swapped Characters). To compare the relative performance of each system, we adopted a precision-based performance metric that is essentially a measure of the balance between correctly detected error repairs (True Positive) and the errors introduced by the system (False Positives).

JamSpell and Pyenchant spelling checkers are inadequate for correcting text errors at character-level, these performed equally poorly, introducing significantly more errors than they fixed. On the contrary, the seq2seq-based neural model was our best performing WEC system, it proved to be very effective, correcting significantly more errors than it introduced. As future work, we plan to enhance the realism of our artificial noise addition method by considering a more natural distribution based on the input corpus; to explore other Neural Network-based WEC systems, moving beyond character-level to subwords.

Neural network-based models have gained a considerable amount of attention within the Natural Language Processing (NLP) community recently as they can be trained to be excellent at detecting and correcting grammatical errors in written (plain) text documents. We showed in this paper that a Seq2Seq encoder-decoder neural network is able to efficiently detect and character-level errors in Wikitext, demonstrating it is possible to translate and extended neural network-based Grammatical Error Correction systems to WEC systems.

Wikis creation is becoming the norm for collaborating exchange knowledge. We have developed an automatic WikiText repairing tools that can easily be implemented into a wiki software engine, the GM-RKB Wikifixer.

8. References

Chisholm, A., Radford, W., and Hachey, B. (2017). Learning to generate one-sentence biographies from wikidata. *arXiv preprint arXiv:1702.06235*.

Chollampatt, S. and Ng, H. T. (2018). A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Dang, Q. V. and Ignat, C.-L. (2016). Quality assessment of wikipedia articles without feature engineering. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries*, pages 27–30. ACM.

Dohrn, H. and Riehle, D. (2011). Design and implementation of the sweble wikitext parser: unlocking the structured data of wikipedia. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, pages 72–81. ACM.

Etoori, P., Chinnakotla, M., and Mamidi, R. (2018). Automatic spelling correction for resource-scarce languages using deep learning. In *Proceedings of ACL 2018, Student Research Workshop*, pages 146–152.

Junghans, M., Riehle, D., Gurram, R., Kaiser, M., Lopes, M., and Yalcinalp, U. (2008). A grammar for standardized wiki markup. In *Proceedings of the 4th International Symposium on Wikis*, page 21. ACM.

Krötzsch, M., Vrandečić, D., and Völkel, M. (2006). Semantic mediawiki. In *International semantic web conference*, pages 935–942. Springer.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Information Processing & Management*, 27(5):517–522.

Melli, G. and McQuinn, J. (2008). Requirements specification using fact-oriented modeling: A case study and generalization. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 738–749. Springer.

Melli, G. (2010). Concept mentions within kdd-2009 abstracts (kdd09cma1) linked to a kdd ontology (kddo1). In *LREC*.

Melli, G. (2012). Identifying untyped relation mentions in a corpus given an ontology. In *Workshop Proceedings of TextGraphs-7 on Graph-based Methods for Natural Language Processing*, pages 30–38. Association for Computational Linguistics.

Mozgovoy, M. (2011). Dependency-based rules for grammar checking with languagetool. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 209–212. IEEE.

Norvig, P. (2007). How to write a spelling corrector.

Singh, D. and Garzon, P. (2015). Using convolutional neural networks and transfer learning to perform yelp restaurant photo classification.

Soni, M. and Thakur, J. S. (2018). A systematic review of automated grammar checking in english language. *arXiv preprint arXiv:1804.00540*.

Sutskever, I., Vinyals, O., and Le, Q. (2014). Sequence to sequence learning with neural networks. *Advances in NIPS*.

Weiss, T. (2016). Deep spelling - rethinking spelling correction in the 21st century. In *machinelearnings.co post*.