Kwery Korrection

IR 646 Project Proposal

Sanjay Reddy S
University of Massachusetts Amherst
Amherst, MA, USA
ssatti@umass.edu

Twinkle Tanna
University of Massachusetts Amherst
Amherst, MA, USA
ttanna@umass.edu

ABSTRACT

In this project we establish a benchmark for query spelling correction using multiple techniques. Three datasets are used: Webis-Qspell-17 (54,772 queries), JDB 2011 (11,134 queries) and Microsoft Speller Challenge TREC (5892 queries). We also report metrics for an end to end retrieval task, where the same methods are applied on Robust04 TREC dataset (249 Queries with relevance judgements on a large index). The aim is to bring together deep learning approaches aimed at spelling correction into the field of information retrieval (specifically for query correction tasks) and see how they stand against the older/currently-used ones. Our experiments, workings, results, contributions and conclusions are presented in this report.

KEYWORDS

Query Correction, Information Retrieval, Neural Networks

1 INTRODUCTION

Search is now a ubiquitous feature on the Web, and is often the main way to access and interact with the enormous data that companies possess — be it website results on Bing and Google, or products on Amazon. Providing relevant results is vital for good user experience, and has the potential for boosting user engagement and revenue for the company. Unsurprisingly, users have high expectations from search — they want the system to guess what they mean and not what they actually type, they want synonyms to be replaced appropriately, they want their results to be personalized, they want to know what other people have been searching and of course they expect retrieval systems to automatically handle spelling errors in their queries

There has been a recent surge in deep learning techniques being applied to correct spelling errors [27] [28] [15]. As far as we could find there haven't been any benchmarks comparing these state of the art techniques to the older ones specifically in the domain of Information Retrieval (IR). [12] puts forth a new dataset but uses older techniques (from 2011). A lot has happened since then. The data at companies like Google is proprietary. Our hope with this paper is to instigate new curiosity in the research community by putting out this benchmark and giving our critique of these methods.

There are actually services like [4] which actually corrects any given input sentence. Another motivating factor could be: The methods described here can be extended to general settings also. For example, it can combined with the autocorrect feature of keyboards in mobiles. Since more and more people are now spending time typing on a digital system, this could be prove very fruitful.

2 SPELL CORRECTION TECHNIQUES

What follows is a technical discussion of the methods used, how they were trained and some of their strengths — weaknesses. At the end of the each discussion 1-2 examples are also provided. The last four are neural methods.

2.1 Peter Norvig's method

This method [22] [27] works with simple edits that comprise of deletion (remove one letter, n such modifications), transposition (swap two adjacent letters-1 such types, n), replacement (change one letter to another, 26 such types) or insertion (add a letter, 26(n+1) such types). (As is evident from the edits mentioned, this method is a direct extension over levenshtein distance). In total, for a word of length n, we can have 54n+25 modifications. These are all the possible candidates. The language model is basically the word frequency as a probability derived from [21] which is built from Project Gutenberg, Wiktionary and British National Corpus. The error model returns the word if found in the dictionary else returns candidates that are 1 edit away. If the word in found, the method stops else goes on to return candidates 2 edits away. This method is not optimal and much preprocessing can be done to make it faster. During testing, the time it took to generate predictions was almost as that of neural methods. The input to Peter Spell is a word and hence every query was evaluated word by word. Any misspellings in the query due to incorrect or omitted spaces can not be handled directlt here.

For the original query Q1: *'2006 form 1040es'* the corrected result obtained is A1: *'2006 form 1040e6'*. For Q2: *'number of military personel killed in training'*, peter spell corrects it to A2: *'number of military personal killed in training'*.

2.2 Symspell's method

Symspell's method [10] of spelling corrections make improvements upon naive search from the dictionary and Peter Norvig's method of edits based on insertions, deletions, alterations and transpositions. Symspell performs a preprocessing step where it generates terms with edit distance deletes for every word in the dictionary. And at run-time, edits for the input terms are searched for in the dictionary. This method compromises on storage requirements and is a O(1) algorithm. It also supports splitting or decompounding of a string wherein wrongly inserted spaces or wrongly ommitted spaces are also corrected as another layer on top of the edits. It also supports word segmentation of texts by inserting spaces at appropriate positions using a triangular matrix approach. For the spell correction experiments on the three dataset, the edit distance dictionary for pre-calculation was set to 2 and compound lookup was performed for every query. This is geared towards fast performance.

For the original query Q1: *'2006 form 1040es'* the corrected result obtained is A1: *'of of form of does'*. For Q2: *'number of military personel killed in training'*, symspell corrects it to A2: *'number of military personal killed in training'*. What we can see here is even though this model does quite well, it needs to work on how to incorporate numbers present in queries and also chose between two candidates wisely as *'personal'* and *'personnel'* were probably two candidates generated and the former was picked probably due to higher frequency of occurrence. Note the similarity with Peter Norvig's method (they are based on similar principles).

2.3 Pyauto correct

A spelling error can be treated as a classification problem trying to identify the most probable candidate of all the words. If w is the incorrect(maybe) word and c is a candidate, then P(c|m) is the probability that the user typed w instead of c. $P(c|w) = P(w|c) * P(c)/P(w)$ Here P(w|c) denoted the probability of making a mistake to type w instead of c and P(c) is the prior that user intended to type c. The error model is calculated as the error probability of typing one letter wrongly (omitted,extra,replaced) to the power of the number of such mistakes. P(c) is the frequency of the word in the language corpus. This method has its foundations in Bayes Probabilities [26]. Unlike the earlier two which do character by character by check this works at word level.

For the original query Q1: *'peart time benexfits'* the corrected result obtained is A1: *'heart time benefits'*. The method got *'benefits'* but not *'part'*. It's probably because the word *'heart'* appeared in the training set a lot more than *'part'* or might also be that it prefers deletions over additions or transposes.

2.4 Using dictionaries

For another simple evaluation method, words that are commonly misspelled were taken from the dictionaries available at Corpora of misspellings [19]. Birbeck, Holbrook, Aspell and Wikipedia all put together account for 36,109 misspelled words. Direct substitution of a word if found was applied.

For the original query Q1: *'world bank cirticism'* the corrected result obtained is A1: *'world's dining cirticism'*. This method is not expected to work well as it is direct substitution assuming that the word given to it is incorrect. But it works quite fast. (especially if we can load the whole dictionary into main memory).

2.5 Word2Vec dictionaries

The drawback with earlier method is that the dictionaries are generated in a static fashion. One has to manually fill the dictionary with commonly made mistakes and map it to their original word. Using word2vec methods we can ask a system to automate that phase. Another added benefit, maintaining a dynamic dictionary is possible. Just feed latest data periodically and create new embeddings.

Word Embeddings are texts converted into vector of numbers (typically using Neural Networks). Also referred to as Word2Vec [18] the input here is a text corpus and output is a set of vectors: feature vectors for all words in that corpus. Word2vec groups vectors of similar words together in vector space (Here similarity of words is infered from the context in which they appear. It detects these mathematically and does so without human intervention).

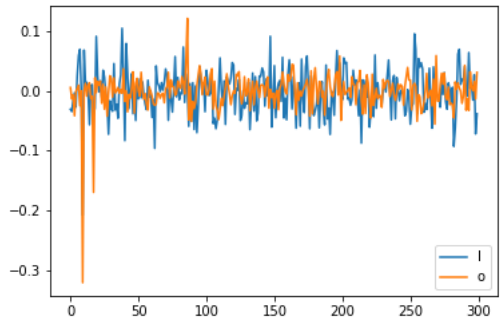

Figure 1: Showing the spelling transformation vector

The easiest way to show the effectiveness of word embeddings:
$\vec{v}(\text{"}King\text{"}) - \vec{v}(\text{"}man\text{"}) + \vec{v}(\text{"}Queen\text{"}) = \vec{v}(\text{"}woman\text{"})$.
$\vec{v}(\text{"}chauffeur\text{"}) - \vec{v}(\text{"}chauffer\text{"}) + \vec{v}(word\text{"}X\text{"}) = \vec{v}(word\text{"}X\text{"})$.
The second example shows the approach used for spelling correction. Even though one is a misspelling of another, they still will appear in similar contexts. We can take many misspellings and generate an average 'spelling transformation vector' which is an aggregate of $\{\vec{v}(correctly\_spelled\_word) - \vec{v}(incorrect\_version)\}$. For the final benchmark, we took the taking Stanford's Glove Word Embeddings [24] which is trained on 840 billion words and each word embedding is of 300 dimensions. To illustrate the above spelling transformation vector in a plot, take Oxford's dataset [3]. This has 102 commonly occurring spelling errors. Figure 1 shows $\vec{v}(\text{"}chauffer\text{"}) - \vec{v}(\text{"}chauffeur\text{"})$ in blue and the aggregate vector in orange. It's a simple line graph of 300 dimensions.

Another key aspect of this, by applying the vector in reverse we can actually generate more noisy data. We use the same Stanford Glove dataset to generate these mistakes (But ensure that the edit distance doesn't cross 3). We ultimately create a dictionary by applying the above method to transform misspelt words to correct ones. Since during output, this dictionary is used the testing is quite fast.

For the original query Q1: *'apparantly'* the corrected result obtained is A1: *'apparently'* and for Q2: *'legal pa Bike 103'* we get A2: *'legal pay Bike 103'*. Out of all the neural methods this works the best.

2.6 Machine Translation Approach

This is sort of like a new spin on an existing idea. Seq2Seq [25] models are very effective in Machine Translation tasks. But instead of using it to translate from say French to English, what if we could leverage the same network and convert broken/incorrect English to properly written English.

A Seq2Seq model is shown in Figure 2. It has two recurrent neural networks working in tandem to transform one sequence to another. An encoder network condenses an input sequence into a single vector, and a decoder network unfolds that vector into a new sequence. The encoder reads an input sequence one item at a time, and outputs a vector at each step. The final output of the encoder is stored as the context vector. The decoder uses this context vector

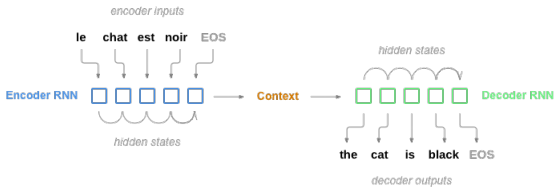

Figure 2: Encoder - Decoder used in Seq2Seq

to produce a sequence of outputs one step at a time. By encoding many inputs into one single context vector, and decoding from that into many outputs, we are freed from the constraints of sequence order and length of source language. We expect the context vector to capture the 'meaning' of the input sentence.

Attention [6] is also another common feature in machine translation and is used here.

For the original query Q1: *'pirac'* the corrected result obtained is A1: *'pirac'* and for Q2: *'legal pa dam 103'* we get A2: *'legal pay Bike 103'*. Easy examples such as the first one were also not corrected and in the second example, *'dam'* was translated to *'Bike'*. One reason that this isn't working could be imbalance between both sides of data: there are a lot of ways a correct word can be misspelt. All these are mapped to the correct word.

2.7 Curie

This is very similar to the earlier Machine Translation Approach. Instead of using a direct pre-trained machine translator [16] and changing the source and target language (after some pre-processing of the files), here a Bi-LSTM RNN network built and trained specifically for this task is used. This model is trained on 20 Gutenberg books [5] (3024075 words in total).

For the original query Q1: *'Spellin is difficult, whch is wyh you need to study everyday'* the corrected result obtained is A1: *'Whilst paliarism tic paliarism miste'* and for Q2: *'teacphing duisalbed chicldren'* we get A2: *'teaching dusiled children chil'*. From these examples we notice that even after the sentence is over, the model tends to predict extra words or letters influenced by what the model observes in parts of the sentence earlier.

2.8 Tal Character level encoding

All the three neural methods do word to word correction. But they have a common problem: They can't deal with OOV (Out of Vocabulary word). So we can try at character level. This sequence-to-sequence model is trained on a modified version of the Google One Billion Word dataset [8]. The noise (as is the case above) is added randomly in words. The inputs were capped at a length of 60 characters and again an encoder-decoder (each 2 LSTM layers) model is used. Any input query greater than length of 60 characters was skipped and no changes were made to that query.

Although Character level encoding has lot of advantages like: no out-of- vocabulary issues, can learn morphology [13], and can move us closer to completely end-to- end translation systems they are very sensitive to noise [7]. So it actually doesn't help in spell check as well as expected. [This took very long to train and after some point we actually had to stop (Left it to run for 7 days on UMass gypsum cluster].

For the original query Q1: *'intenrattionoal ogrnaized rime'* the corrected result obtained is A1: *'interrationalll organized rime'*. So we see that the model is definitely learning. The part of the word *'interrationalll'* comprises *'inter'* and *'rational'* which independently do make sense.

3 EXPERIMENTAL SETTING

3.1 Dataset Statistics

The public dataset released with Microsoft Speller Challenge is an adaptation of TREC Million Queries with ground truth available [17]. The second dataset, JDB2011 [9] is sampled from AOL query logs and 2009 Million Query Task and contains 11,134 queries. This dataset is also of the form original query and alternate query suggestions. There are two variants of these: V1 (only tested on their train.txt of 6000 queries) and V2 (tested on train.txt+3 test.txt files). The Webis Query Spelling Corpus 2017 [12] has 54,772 web queries which are extracted from AOL query log. The data comprises of the query id, query, original query (correct spelling) and alternative correct spellings for 9170 of the queries. All queries are within 3-10 words long. Compared to the earlier two dataset, this contains a lot more queries and covers a lot more kind of spelling errors as well. Earlier benchmarks have performed worse on this dataset compared to the other two. All neural network based approaches have used larger datasets for example Statistical Machine Translation Corpus [2], Google's Billion Word Dataset [8], Stanford Glove reprsentation [24]. For incorrectly spelled words, we randomly inserted noise in correct ones.

Robust04 TREC [1] is a commonly used dataset to run end-to-end relevance experiments. The index is a large set of documents (upwards of 1 GB).

3.2 Evaluation Criterion

- The trouble with spelling corrections is that you can get many false positives thrown in even appropriate corrections [14]. The risk is that one might end up changing the meaning of a sentence by changing the spelling of an unknown - rather than missspelt - word. (For example, with regards to proper nouns, i.e. names). So to simulate a real world setting, we *always* keep the given input query typed in. All the accuracies (for all the 8 methods) thus will always be above the baseline.
- The first metric we use is vanilla accuracy. Exact string match only is considered correct. The way to reason about this: It's the worst case performance. (Note that this metric is same as Expected precision [9] when all methods given equal weights to suceed).
- The second metric we use is the levenshtein distance. A distance of 0 indicates an exact match. Lower the distance, better it is. Each prediction is compared to all the candidates and the least distance is considered. This is averaged over all the input queries. The earlier one says how good can our methods be in worst case. But this says when we fail, how bad can the predictions go.

Table 1: Accuracy Results on the 3 datasets

Methods	Microsoft	JDBv1	JDBv2	QSpell
Baseline	94.72	90.58	90.18	87.68
Peter	95.89	92.48	92.18	89.83
Symspell	**96.05**	**93.06**	**92.90**	**89.84**
Pyauto	94.94	91.06	90.64	88.25
Dictionaries	94.96	90.78	90.4	88.04
Word2Vec	**95.75**	**91.55**	**91.09**	**89.54**
NMT	94.77	90.61	90.21	87.78
Curie	94.82	90.95	90.59	87.79
Tal	94.91	90.75	90.38	87.94
Hybrid	**96.57**	**93.76**	**93.77**	**90.98**

Table 2: Distance measures on the 3 datasets

Methods	Microsoft	JDBv1	JDBv2	QSpell
Baseline	0.062	0.124	0.127	0.179
Peter	**0.248**	**0.412**	**0.41**	**0.179**
Symspell	0.845	0.7155	0.703	0.85
Pyauto	2.06	1.435	1.436	2.032
Dictionaries	5.02	1.788	1.876	3.556
Word2Vec	**0.0541**	**0.114**	**0.118**	**0.159**
NMT	1.255	0.911	3.528	1.407
Curie	3.376	2.409	2.383	3.164
Tal	1.79	1.206	1.201	1.786
Hybrid	**0.038**	**0.079**	**0.077**	**0.133**

- In the end to end measure MAP, nDCG and Precision@10 are reported on retreival on an index using query likelihood model with dirichlet smoothing and Krovetz Stemmer.

4 RESULTS

The codebase is at: github link (Refer the ReadME).

Table 1 has the accuracy metrics on the three datasets. Table 2 has average minimum levenshtein distance numbers. Hybrid (as expected) model performs the best. In neural networks, Word2Vec performs the best.

Table 3 has the retrieval metrics on Robust04 TREC. Peter Norvig's method works the best. In neural networks, Word2Vec performs the best.

5 ANALYSIS AND DISCUSSION

As is evident, Neural Network Methods aren't working as well as expected. The older methods are not only working pretty well but also are much more faster (This is true during both training time and testing time). Neural methods take lot of time. In the older methods, we do not expect Dictionaries and PyAuto to work well. The dictionaries data is quite small and the PyAuto relies a lot on the frequencies of observed words in its data. Symspell performs well in terms of absolute string match whereas PeterSpell

Table 3: Relevance metrics on end-to-end relevance task

Methods	MAP	nDCG	P@10
Actual Queries	0.254	0.517	0.430
Noisy Queries	0.094	0.244	0.181
Peter	**0.187**	**0.415**	**0.322**
Symspell	0.168	0.387	0.290
Pyauto	0.079	0.213	0.156
Dictionaries	0.086	0.214	0.159
Word2Vec	**0.102**	**0.259**	**0.199**
NMT	0.089	0.227	0.174
Curie	0.071	0.189	0.123
Tal	0.081	0.216	0.137

performs best when you look at the distance metric. This can be reasoned well because the PeterSpell method is designed to reduced the levenshtein distance (It is optimized on that metric directly). In the neural methods, Curie performs the worst. We notice that the Bi-LSTM model requires some changes as multiple instances have trailing letter sequences that were previously observed in the same query. Perhaps performing more regularization could improve this further. Also, the number of layers are too small to effectively capture the 'meaning'. There's also the hyper-parameter tuning. NMT chooses the best options by doing automatic random search on validation set. But Curie didn't have any such option. Since the training took lot of time, we didn't have the option to tune for better performance. The NMT approach probably needs to be trained more or requires better kind of data to be trained. Tal which is the Deep Spell approach also uses an encoder-decoder style but is character level model. We see that it did get many letters of words right but ends up with multiple extra characters towards the end of the word like the Curie model does extra words. The Word2Vec approach by far performs the best in neural methods. In terms of the levenshtein distance, it is infact the least and goes even better than the baseline. This brings the question that why doesn't it perform better in terms of absolute string match or the end to end task. Symspell goes wrong when it comes to numbers and abbreviations as in *'california assembly 43 district'*, *'ada high'* where W2V got *'california assembly of district'*, *'fda hgh'*. On the end to end task, we notice that on performing all the evaluations on the 249 query data, Word2Vec doesn't perform as well. This was a synthetic dataset with 85% random noise. With this level of noise and multiple mistakes even in the same word, the Word2Vec model loses it's power of better cosine similarity in the distributed space. Word2Vec approach still has some ways that can be improved. There are cases where prepositions like *'before* were corrected to *'about'* or *'of'*. Even though they are all prepositions, this is not desired. For example in *'maximum benefit for social security disability michigan'*, Word2Vec changes it to *'maximum benefit before social curiosity disability michigan'*.

6 FUTURE WORK

- Try a new CNN model for translation [11].

- Instead of spellings one can focus on the grammar of the sentences (or even both) [29] [23] and [28].
- As mentioned training data became the main issue for neural methods. Surprisingly, they aren't many datasets available for this domain and for Neural Networks to work we need abundance of data. The way we generate noise is using random functions. This doesn't exactly model real world mistakes. So both: better training data and better ways to model real world noise are two possible avenues to pursue. Also, hyper parameter tuning is of vital importance for Neural methods. We need to have proper validation sets so that we can experiment.
- Word Embeddings, as mentioned, capture lot of semantic knowledge about the word. So one way to improve accuracy, instead of words feed the vectors as input to the neural networks. For NMT in particular, this is particularly straight forward [16].
- Once we have good amount of data, we can start tweaking the neural network methods. All four presented here are all standard architectures (that too with the bare minimum number of layers). We can start increasing the number of layers. Bring in new architectures like Siamese RNNs [20].

REFERENCES

[1] 2004. TREC 2004 Robust Track Guidelines. https://trec.nist.gov/data/robust/04.guidelines.html

[2] 2014. NINTH WORKSHOP ON STATISTICAL MACHINE TRANSLATION. http://www.statmt.org/wmt14/training-monolingual-news-crawl/

[3] 2018. Common misspellings. https://en.oxforddictionaries.com/spelling/common-misspellings

[4] 2018. Perfect Tense. https://www.perfecttense.com/docs/#introduction

[5] 2018. Project Gutenberg. https://www.gutenberg.org/wiki/Main_Page

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014). arXiv:1409.0473 http://arxiv.org/abs/1409.0473

[7] Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and Natural Noise Both Break Neural Machine Translation. *CoRR* abs/1711.02173 (2017). arXiv:1711.02173 http://arxiv.org/abs/1711.02173

[8] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling*. Technical Report. Google. http://arxiv.org/abs/1312.3005

[9] Yasser Ganjisaffar, Andrea Zilio, Sara Javanmardi, Inci Cetindil, Manik Sikka, Sandeep Katumalla, Narges Khatib, Chen Li, and Cristina M P V Lopes. 2011. qSpell : Spelling Correction of Web Search Queries using Ranking Models and Iterative Correction.

[10] Wolf Garbe. 2017. 1000x faster Spelling Correction. https://towardsdatascience.com/symspellcompound-10ec8f467c9b

[11] Jonas Gehring and Michael Auli. 2017. A novel approach to neural machine translation. https://code.fb.com/ml-applications/a-novel-approach-to-neural-machine-translation/?utm_campaign=Artificial%2BIntelligence%2Band%2BDeep%2BLearning%2BWeekly&utm_medium=email&utm_source=Artificial_Intelligence_and_Deep_Learning_Weekly_13

[12] Matthias Hagen, Martin Potthast, Marcel Gohsen, Anja Rathgeber, and Benno Stein. 2017. A Large-Scale Query Spelling Correction Corpus. In *40th International ACM Conference on Research and Development in Information Retrieval (SIGIR 2017)*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 1261–1264. https://doi.org/10.1145/3077136.3080749

[13] Dr Peter Hancox. 1996. Morphological analysis. https://www.cs.bham.ac.uk/~pjh/sem1a5/pt2/pt2_intro_morphology.html

[14] Jeremy Howard. 2018. NLP: Any libraries/dictionaries out there for fixing common spelling errors? https://forums.fast.ai/t/nlp-any-libraries-dictionaries-out-there-for-fixing-common-spelling-errors/16411

[15] Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. 2017. A Nested Attention Neural Hybrid Model for Grammatical Error Correction. In *ACL*.

[16] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. ACL*. https://doi.org/10.18653/v1/P17-4012

[17] Gord Lueck. 2011. A data-driven approach for correcting search queries. *Spelling Alteration for Web Search Workshop 2011* (2011).

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781 http://arxiv.org/abs/1301.3781

[19] Roger Mitton. 1996. Spellchecking by computer. https://www.dcs.bbk.ac.uk/~ROGER/corpora.html?fbclid=IwAR3CWF2yeyOKrByRj9a_4NyfBRLeDyVDqIdit9evN4zLme20aTyBN_VLCaU

[20] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese Recurrent Architectures for Learning Sentence Similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, 2786–2792. http://dl.acm.org/citation.cfm?id=3016100.3016291

[21] Peter Norvig. 2007. Big Text. https://norvig.com/big.txt

[22] Peter Norvig. 2007. How to Write a Spelling Corrector. https://norvig.com/spell-correct.html

[23] Alex Paino. 2017. Deep Text Corrector. http://atpaino.com/2017/01/03/deep-text-corrector.html

[24] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *CoRR* abs/1409.3215 (2014). arXiv:1409.3215 http://arxiv.org/abs/1409.3215

[26] Synced. 2017. Applying Multinomial Naive Bayes to NLP problems a practical explanation. https://syncedreview.com/2017/07/17/applying-multinomial-naive-bayes-to-nlp-problems-a-practical-explanation/

[27] Tal Weiss. 2017. Deep Spelling. https://machinelearnings.co/deep-spelling-9ffef96a24f6?fbclid=IwAR2l369XqNR_5xq62a5ARYvws5XlY6c1pNKBgAILVyZ_44YJoUgqIXyda1k

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

[29] Zheng Yuan and Ted Briscoe. 2016. Grammatical error correction using neural machine translation. 380–386. https://doi.org/10.18653/v1/N16-1042