Effective search space reduction for spell correction using character neural embeddings

Harshit Pande
Smart Equipment Solutions Group
Samsung Semiconductor India R&D
Bengaluru, India
pandeconscious@gmail.com

Abstract

We present a novel, unsupervised, and distance measure agnostic method for search space reduction in spell correction using neural character embeddings. The embeddings are learned by skip-gram word2vec training on sequences generated from dictionary words in a phonetic information-retentive manner. We report a very high performance in terms of both success rates and reduction of search space on the Birkbeck spelling error corpus. To the best of our knowledge, this is the first application of word2vec to spell correction.

1 Introduction

Spell correction is now a pervasive feature, with presence in a wide range of applications such as word processors, browsers, search engines, OCR tools, etc. A spell corrector often relies on a dictionary, which contains correctly spelled words, against which spelling mistakes are checked and corrected. Usually a measure of distance is used to find how close a dictionary word is to a given misspelled word. One popular approach to spell correction is the use of Damerau-Levenshtein distance (Damerau, 1964; Levenshtein, 1966; Bard, 2007) in a noisy channel model (Norvig, 2007; Norvig, 2009). For huge dictionaries, Damerau-Levenshtein distance computations between a misspelled word and all dictionary words lead to long computation times. For instance, Korean and Japanese may have as many as 0.5 million words[1]. A dictionary further grows when inflections of the words are also considered. In such cases, since an entire dictionary becomes the search space, large number of distance computations blows up the time complexity, thus hindering real-time spell correction. For Damerau-Levenshtein distance or similar edit distance-based measures, some approaches have been tried to reduce the time complexity of spell correction. Norvig (2007) does not check against all dictionary words, instead generates all possible words till a certain edit distance threshold from the misspelled word. Then each of such generated words is checked in the dictionary for existence, and if it is found in the dictionary, it becomes a potentially correct spelling. There are two shortcomings of this approach. First, such search space reduction works only for edit distance-based measures. Second, this approach too leads to high time complexity when the edit distance threshold is greater than 2 and the possible characters are large. Large character set is real for Unicode characters used in may Asian languages. Hulden (2009) proposes a Finite-State-Automata (FSA) algorithm for fast approximate string matching to find similarity between a dictionary word and a misspelled word. There have been other approaches as well using FSA, but such FSA-based approaches are approximate methods for finding closest matching word to a misspelled word. Another more recent approach to reduce the average number of distance computations is based on anomalous pattern initialization and partition around medoids (de Amorim and Zampieri, 2013).

In this paper, we propose a novel, unsupervised, distance measure agnostic, highly accurate, method of search space reduction for spell correction with a high reduction ratio. Our method is unsupervised because we use only a dictionary of correctly spelled words during the training process. It is distance measure agnostic because once the search space has been reduced then any distance measure of spell correction can be used. It is novel because to the best of our knowledge, it

[1] http://www.lingholic.com/how-many-words-do-i-need-to-know-the-955-rule-in-language-learning-part-2/

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, pages 170–174,
Valencia, Spain, April 3-7, 2017.

is the first application of neural embeddings learning word2vec techniques (Mikolov et al., 2013a; Mikolov et al., 2013b) to spell correction. The goal of this paper is not to find a novel spell correction algorithm. Rather, the goal is to reduce the time complexity of spell correction by reducing the search space of words over which the search for correct spelling is to be done. The reduced search space contains only a fraction of words of the entire dictionary, and we refer to that fraction as reduction ratio. So, our method is used as a filter before a spell correction algorithm. We discuss a closely related work in Section 2, which is followed by description of our method in Section 3. Then we present our experiments and results in Section 4, which demonstrates the effectiveness of our approach.

2 Related Work

As discussed in Section 1, there have been studies to reduce the time complexity of spell correction by various methods. However, the recent work of de Amorim and Zampieri (2013) is closest to our work in terms of the goal of the study. We briefly describe their method and evaluation measure, as it would help us in comparing our results to theirs, though the results are not exactly comparable.

De Amorim and Zampieri (2013) cluster a dictionary based on anomalous pattern initialization and partition around medoids, where medoids become the representative words of the clusters and the candidacy of a good cluster is determined by computing the distance between the misspelled word and the medoid word. This helps in reducing the average number of distance computations. Then all the words belonging to the selected clusters become candidates for further distance computations. Their method on average needs to perform 3,251.4 distance calculations for a dictionary of 57,046 words. This amounts to 0.057 reduction ratio. They also report a success rate of 88.42% on a test data set known as Birkbeck spelling error corpus.[2] However, it is important to note that they define success rate in a rather relaxed manner - one of the selected clusters contains either the correct spelling or contains a word with a smaller distance to the misspelled word than the correct word. Later in Section 4, we define a stricter and natural definition of success rate for our studies. This difference in relaxed vs strict success rates along with the inherent differences in approach render their method and our method not entirely comparable.

[2] http://www.dcs.bbk.ac.uk/ ROGER/corpora.html

3 Method

Recent word2vec techniques (Mikolov et al., 2013a; Mikolov et al., 2013b) have been very effective for representing symbols such as words in an n-dimensional space $\mathbb{R}^n$ by using information from the context of the symbols. These vectors are also called neural embeddings because of the one hidden layer neural network architecture used to learn these vectors. In our method, the main idea is to represent dictionary words as n-dimensional vectors, such that with high likelihood the vector representation of the correct spelling of a misspelled word is in the neighborhood of the vector representation of the misspelled word. To quickly explore the neighborhood of the misspelled word vector, fast k-nearest-neighbor (k-NN) search is done using a Ball Tree (Omohundro, 1989; Liu et al., 2006; Kibriya and Frank, 2007). A Ball Tree (aka Metric Tree) retrieves k-nearest-neighbors of a point in time complexity that is logarithmic of the total number of points (Kibriya and Frank, 2007). There are other methods, such as Locally-Sensitive Hashing (LSH) and KD-Tree, which can also be used to perform fast k-NN search. We use Ball Tree because in our experiments, Ball Tree outperforms both KD-Tree and LSH in terms of speed of computation.

We treat a word as a bag of characters. For each character, an n-dimensional vector representation is learned using all the words from a dictionary of correctly spelled words. Each word is then represented as an n-dimensional vector formed by summing up the vectors of the characters in that word. Then in a similar manner, a vector is obtained for the misspelled word by summing up the vectors of the characters in the misspelled word. We start with a few notations:

- n : dimension of neural embedding
- m : window size for word2vec training
- W : set of all dictionary words
- w : input misspelled word
- k : size of the reduced search space
- C : set of all the language characters present in W
- $C2Vmap$: a map of all characters in C to their n-dimensional vector representations

- $V2Wmap$: a map of vectors to the list of words represented by the vectors[3]
- BT : a Ball Tree of all the vectors

Our method is divided into two procedures. The first procedure is a preprocessing step, which needs to be done only once, and the second procedure is the search space reduction step.

3.1 Procedure 1: preprocessing

1. Prepare sequences for word2vec training: each word w' in W is split into a sequence such that each symbol of such sequence contains the longest possible contiguous vowels[4] or consonants but not both. E.g. "affiliates" generates the sequence "a ff i l ia t e s"

2. Train skip-gram word2vec model with sequences generated in the previous step with hidden layer size as n and window size as m. Training yields neural embeddings for symbols present in training sequences. For each character c in C, store the neural embeddings in $C2Vmap$ for future retrieval.

3. For each word w' in W, compute the n-dimensional vector representation of w' by summing up neural embeddings (using $C2Vmap$) of the characters in w'.

4. Fill $V2Wmap$ with key as vector computed in the previous step and value as list of words represented by that vector. Also construct BT for the word vectors computed in the previous step.

The peculiar way of sequence generation in step 1 of Procedure 1 is chosen for both empirical and intuitive reasons. Experimentally, we tried multiple ways of sequence generation, such as simply breaking a word into all it's characters, making symbols that are longest possible contiguous consonants but each vowel is a separate symbol, making symbols that are longest possible contiguous vowels but each consonant is a separate symbol, and the one given in the step 1 of Procedure 1. We found that the sequence generation given in step 1 of Procedure 1 gives the best success rates. An intuitive reasoning is that if each symbol of a sequence contains the longest possible contiguous vowels or consonants but not both, then it retains phonetic information of a word. Phonetic information is vital for correcting spelling mistakes.

3.2 Procedure 2: search space reduction

1. Compute v_w, the n-dimensional vector representation of misspelled word w, by summing up the vector representations of the characters in w (using $C2Vmap$).

2. Find $kNearNeighb$: k nearest-neighbors of v_w using BT.

3. Using $V2Wmap$ fetch the reduced search space of words corresponding to each vector in $kNearestNeighb$

Once the reduced search space of words is obtained as in step 3 of procedure 2, then any spell correction algorithm can be used to find the correct spelling of misspelled word w. This also means that our search space reduction method is completely decoupled from the final spell correction algorithm.

4 Experiments and Evaluation

In this section, we describe our experiments and their effectiveness in search space reduction of spell correction. As discussed in Section 2, recent work of de Amorim and Zampieri (2013) is closest to our work in terms of the goal of the study, so we make comparisons with their work wherever possible.

4.1 Data

We chose a dictionary W containing 109,582 words[5], which is almost twice the size of dictionary used by de Amorim and Zampieri (2013). For testing, we use the same Birkbeck spelling error corpus as used by de Amorim and Zampieri (2013). However, de Amorim and Zampieri (2013) remove those test cases from the Birkbeck corpus for which the correctly spelled word is not present in their dictionary. We on the other hand include such words in our dictionary and enhance the size of our dictionary. This leads to the final size of 109,897 words in the enhanced dictionary. It is also worth mentioning that Birkbeck corpus is a very challenging test data set, with some spelling mistakes as wide as 10 edit distances apart.

[3] multiple words may have same vector representation, e.g. anagrams

[4] we include character y in the vowel set

[5] http://www-01.sil.org/linguistics/wordlists/english/

4.2 Evaluation Measure

We use success rate as a measure of accuracy. De Amorim and Zampieri (2013) used a relaxed definition of success rate (see Section 2), which we call relaxed success rate. We have a stricter definition of success rate, where success is defined as occurrence of the correct spelling of a misspelled word in the reduced search space. Reduction ratio for our method is $1.1k/|W|$. The 1.1 factor is present because average number of words per vector in $V2Wmap$ is 1.1. Thus, on average, we need to do $1.1k$ distance computations post search space reduction. It is worth noting that k is in fact flexible, and thus it is vital that $k << |W|$ to achieve a significant improvement in time complexity of spell correction.

4.3 Experimental Setup

We implemented the procedures given in Section 3 partly in Java and partly in Python. For word2vec training Deep Learning library DL4J[6] was used, and Scikit-learn (Pedregosa et al., 2011) library was used for Ball Tree[7] to facilitate fast k-NN search. All the experiments were conducted on an Ubuntu 16.04 machine with Intel® Core™ 2 Duo CPU P8800 @ 2.66GHz with 8 GB of RAM.

4.4 Results

In Section 3.1, we already discussed how the sequence generation given in step 1 of Procedure 1 gave the best success rates as compared to other sequence generation methods. Similarly, window size $m = 4$ in word2vec training gave best success rates. Also for k-NN using BT, we experimented with various metrics and found Euclidean metric to be giving best success rates. For reporting, we vary k and n because they directly influence the reduction ratio and time complexity of search space reduction. Table 1 shows success rates for various values of k and n.

n \ k	**1000**	**2000**	**5000**
25	76.26	81.13	87.00
50	77.96	82.39	87.95
100	76.82	82.52	88.20

Table 1: Success rates (%) for various k and n

[6]http://deeplearning4j.org/

[7]http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html

For $k = 5000$ and $n = 100$, we achieve a success rate of 88.20% for a strict (and natural) definition of success rate (defined in Section 4.2) while de Amorim and Zampieri (2013) report a success rate of 88.42% for a relaxed definition of success rate (defined in Section 2). Further, $k = 5000$ boils down to reduction ratio of 0.050, which is an improvement over reduction ratio of 0.057 reported by de Amorim and Zampieri (2013). It is also important to note that even at low dimensions of neural embeddings such as $n = 25$, the success rates are only slightly lower than those at $n = 100$. This means that other fast k-NN retrieval methods such as KD-Trees (Kibriya and Frank, 2007) may also be used because they are quite efficient at such low dimensions. Also, smaller dimensions further speed up computations because of speeding up of vector similarity computations. This is a useful trade-off, where small decrease in accuracy can be traded off for more increase in computation speed. We see such flexibility of choosing k and n as an advantage of our method.

In practice, a large number of spelling mistakes occur within few edit distances of their correct spelling. Thus we also present extremely high success rates of our method for $k = 5000$ and $n = 100$ for the subset of Birkbeck corpus having Damerau-Levenshtein distances between misspelled word and correct spelling within 2, 3, and 4. These results are shown in Table 2

Damerau-Levenshtein distance	**Success Rate**
≤ 2	99.59
≤ 3	97.87
≤ 4	94.72

Table 2: Success rates (%) for test data with mistakes within various Damerau-Levenshtein distances (for $k = 5000$ and $n = 100$)

For $k = 5000$ with $n = 100$, the search space reduction followed by success rate evaluation took on average 52 ms per test case on the modest system configurations given in Section 4.3. This shows that our method has real-time response times. For larger dictionaries, the effect would be more profound as the time complexity of our method is logarithmic in the size of dictionary.

5 Conclusions and Future Work

In this paper, we proposed a novel, unsupervised, distance-measure agnostic method of search space

reduction for spell correction. Our method outperforms one of the recent methods, both in terms of the extent of search space reduction and success rates. For common spelling mistakes, which are usually within a few edit distances, our method has extremely high success rates, for example, we achieved success rate of 99.6% and 97.9% for spelling mistakes within edit distance 2 and 3 respectively.

As we noticed, sequence generation for word2vec training does influence success rates, so we are currently exploring further ways of sequence generation. We would also like to introduce mild supervision element by generating more data for word2vec training by mutating dictionary words using confusion sets (Pedler and Mitton, 2010). We would also like to explore the effectiveness of our approach on languages other than English.

Acknowledgments

We are grateful to the anonymous reviewers for providing reviews that led to improvement to the paper. We also thank Aman Madaan and Priyanka Patel for making useful comments, which were incorporated in the final draft of the paper.

References

Gregory V. Bard. 2007. Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric. In *Proceedings of the fifth Australasian Symposium on ACSW Frontiers*, volume 68, pages 117–124, Ballarat, Australia, January. Australian Computer Society, Inc.

Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, March.

Renato Cordeiro de Amorim and Marcos Zampieri. 2013. Effective spell checking methods using clustering algorithms. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 172–178, Hissar, Bulgaria, September. INCOMA Ltd. Shoumen, BULGARIA.

Mans Hulden. 2009. Fast approximate string matching with finite automata. *Procesamiento del Lenguaje Natural*, 43:57–64.

Ashraf M. Kibriya and Eibe Frank. 2007. An empirical comparison of exact nearest neighbour algorithms. In *Proceedings of the 11th European Conference on Principles of Data Mining and Knowledge Discovery in Databases*, pages 140–151, Warsaw, Poland, September. Springer-Verlag.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, Soviet Union, February.

Ting Liu, Andrew W. Moore, and Alexander Gray. 2006. New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research*, 7:1135–1158, June.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, Lake Tahoe, Nevada, USA, December. Curran Associates, Inc.

Peter Norvig. 2007. How to write a spelling corrector. `http://norvig.com/spell-correct.html`. [Online; accessed 12-November-2016].

Peter Norvig. 2009. Natural language corpus data. In *Beautiful Data*, chapter 14, pages 219–242. O'Reilly Media, Sebastopol, California, USA.

Stephen M. Omohundro. 1989. *Five Balltree Construction Algorithms*. International Computer Science Institute, Berkeley, California, USA.

Jennifer Pedler and Roger Mitton. 2010. A large list of confusion sets for spellchecking assessed against a corpus of real-word errors. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation*, pages 755–762, Valletta, Malta, May. European Language Resources Association.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, October.