Automatic Spelling Correction for Resource-Scarce Languages using Deep Learning

Pravallika Etoori
LTRC, KCIS
IIIT Hyderabad
e.pravallika@research.iiit.ac.in

Manoj Chinnakotla
Microsoft
Bellevue, USA
manojc@microsoft.com

Radhika Mamidi
LTRC, KCIS
IIIT Hyderabad
radhika.mamidi@iiit.ac.in

Abstract

Spelling correction is a well-known task in Natural Language Processing (NLP). Automatic spelling correction is important for many NLP applications like web search engines, text summarization, sentiment analysis etc. Most approaches use parallel data of noisy and correct word mappings from different sources as training data for automatic spelling correction. Indic languages are resource-scarce and do not have such parallel data due to low volume of queries and non-existence of such prior implementations. In this paper, we show how to build an automatic spelling corrector for resource-scarce languages. We propose a sequence-to-sequence deep learning model which trains end-to-end. We perform experiments on synthetic datasets created for Indic languages, Hindi and Telugu, by incorporating the spelling mistakes committed at character level. A comparative evaluation shows that our model is competitive with the existing spell checking and correction techniques for Indic languages.

1 Introduction

Spelling correction is important for many of the potential NLP applications such as text summarization, sentiment analysis, machine translation (Belinkov and Bisk, 2017). Automatic spelling correction is crucial in search engines as spelling mistakes are very common in user-generated text. Many websites have a feature of automatically giving correct suggestions to the misspelled user queries in the form of *Did you mean?* suggestions or automatic corrections. Providing suggestions makes it convenient for users to accept a proposed correction without retyping or correcting the query manually. This task is approached by collecting similar intent queries from user logs (Hasan et al., 2015; Wilbur et al., 2006; Ahmad and Kondrak, 2005). The training data is automatically extracted from event logs where users re-issue their search queries with potentially corrected spelling within the same session. Example query pairs are *(house lone, house loan), (ello world, hello world), (mobilephone, mobile phone)*. Thus, large amounts of data is collected and models are trained using techniques like Machine Learning, Statistical Machine Translation etc.

The task of spelling correction is challenging for resource-scarce languages. In this paper, we consider Indic languages, Hindi and Telugu, because of their resource scarcity. Due to lesser query share, we do not find the same level of parallel alteration data from logs. We also do not have many language resources such as Parts of Speech (POS) Taggers, Parsers etc. to linguistically analyze and understand these queries. Due to lack of relevant data, we create synthetic dataset using highly probable spelling errors and real world errors in Hindi and Telugu given by language experts. Similarly, synthetic dataset can be created for any resource-scarce language incorporating the real world errors. Deep Learning techniques have shown enormous success in sequence to sequence mapping tasks (Sutskever et al., 2014). Most of the existing spell-checkers for Indic languages are implemented using rule-based techniques (Kumar et al., 2018). In this paper, we approach the spelling correction problem for Indic languages with Deep learning. This model can be employed for any resource-scarce language. We propose a character based Sequence-to-sequence text Correction Model for Indic Languages (SCMIL) which trains end-to-end.

Our main contributions in this paper are summarized as follows:

- We propose a character based recurrent

Proceedings of ACL 2018, Student Research Workshop, pages 146–152
Melbourne, Australia, July 15 - 20, 2018.

sequence-to-sequence architecture with a Long Short Term Memory (LSTM) encoder and a LSTM decoder for spelling correction of Indic languages.

- We create synthetic datasets[1] of noisy and correct word mappings for Hindi and Telugu by collecting highly probable spelling errors and inducing noise in clean corpus.

- We evaluate the performance of SCMIL by comparing with various approaches such as Statistical Machine Translation (SMT), rule-based methods, and various deep learning models, for this task.

2 Related Work

Significant work has been done in the field of Spell checking for Indian languages. There are spell-checkers available for Indian languages like Hindi, Marathi, Bengali, Telugu, Tamil, Oriya, Malayalam, Punjabi.

Dixit et al. (2005) designed a rule-based spell-checker for Marathi, a major Indian Language. This is the first initiative for morphology-based spell checking for Marathi. The spell-checker is based on the rules of morphology and the rules of orthography.

A spell-checker is designed for Telugu (Rao, 2011), an agglutinating Indian language which has a very complex morphology. This spell-checker is based on Morphological Analysis and Sandhi splitting rules. It consists of two parts: a set of routines for scanning the text (Morphological Analyzer and sandhi splitting rules) and identifying valid words, and an algorithm for comparing the unrecognized words and word parts against a known list of variantly spelled words and word parts.

Another Hindi Spell-checker (Sharma and Jain, 2013) uses a dictionary with word, frequency pairs as language model. Error detection is done by dictionary look-up. Error correction is performed using Damerau-Levenshtein edit distance and n-gram technique. These candidates are ranked by sorting in increasing order of edit distance. Words at same edit distance are sorted in order of their frequencies.

HINSPELL(Singh et al., 2015) is a spell-checker designed for Hindi which is implemented using a hybrid approach. Error is detected by dictionary look-up. Error correction is done by using Minimum Edit Distance technique where the closest words in the dictionary to the error word are obtained. These obtained words are given priority using a weightage algorithm and Statistical Machine Translation (SMT).

Ambili et al. (2016) designed a Malayalam spell-checker that detects the error by a dictionary look-up approach and error correction is done through N-gram based technique. If a word is not present in the dictionary, it is identified as an error and N-gram based technique corrects error by finding similarity between words and computing a similarity coefficient.

Recently, Ghosh and Kristensson (2017) proposed a Deep Learning model for text correction and completion in keyboard decoding for English. This is a first attempt at text correction using Deep Neural Networks which gave promising results.

Sakaguchi et al. (2017) approached the problem of Spell Correction using semi-character Recurrent Neural Networks on English data.

Studies of spell checking techniques for Indian Languages (Kumar et al., 2018; Gupta and Mathur, 2012) show that the existing spell-checkers have two major steps: Error detection and error correction. Error detection is done by dictionary look-up. Error correction consists of two steps: the generation of candidate corrections and the ranking of candidate corrections. The most studied spelling correction algorithms are: edit distance, similarity keys, rule-based techniques, n-gram-based techniques, probabilistic techniques, neural networks, and noisy channel model. All of these methods can be thought of as calculating a distance between the misspelled word and each word in the dictionary or index. The shorter the distance the higher the dictionary word is ranked.

While there have been a few attempts to design spell-checkers for English and few other languages using Machine Learning, to the best of our knowledge, no such prior work has been attempted for Indian languages.

3 Model Description

We address the spelling correction problem for Indic languages by having a separate corrector network as an encoder and an implicit language model as a decoder in a sequence-to-sequence at-

[1] https://github.com/PravallikaRao/SpellChecker

tention model that trains end-to-end.

3.1 Sequence-to-sequence Model

Sequence-to-sequence (seq2seq) models (Sutskever et al., 2014; Cho et al., 2014) have enjoyed great success in a variety of tasks such as machine translation, speech recognition, image captioning, and text summarization. A basic sequence-to-sequence model consists of two neural networks: an encoder that processes the input and a decoder that generates the output. This model has shown great potential in input-output sequence mapping tasks like machine translation. An input side encoder captures the representations in the data, while the decoder gets the representation from the encoder along with the input and outputs a corresponding mapping to the target language. Intuitively, this architectural set-up seems to naturally fit the regime of mapping noisy input to de-noised output, where the corrected prediction can be treated as a different language and the task can be treated as Machine Translation.

3.2 System Architecture

The Recurrent Neural Network (RNN) (Rumelhart et al., 1986; Werbos, 1990) is a natural generalization of feed-forward neural networks to sequences. Given a sequence of inputs (x_1, ..., x_T), a standard RNN computes a sequence of outputs (y_1, ..., y_T) by iterating the following equation:

$$h_t = sigm(W^{hx}x_t + W^{hh}h_{t-1}) \quad (1)$$

$$y_t = W^{yh}h_t \quad (2)$$

The RNN can easily map sequences to sequences whenever the alignment between the inputs the outputs is known ahead of time. In fact, recurrent neural networks, long short-term memory networks (Hochreiter and Schmidhuber, 1997), and gated recurrent neural networks (Chung et al., 2014) have become standard approaches in sequence modelling and transduction problems such as language modelling and machine translation.

RNNs struggle to cope with long-term dependency in the data due to vanishing gradient problem (Hochreiter, 1998). This problem is solved using Long Short Term Memory (LSTM) recurrent neural networks.

SCMIL has the similar underlying architecture of sequence-to-sequence models. The encoder and decoder in SCMIL operate at character level.

Encoder: In SCMIL, the encoder is a character based LSTM. With LSTM as encoder, the input sequence is modeled as a list of vectors, where each vector represents the meaning of all characters in the sequence read so far.

Decoder: The decoder in SCMIL is a character level LSTM recurrent network with attention. The output from the encoder is the final hidden state of the character based LSTM encoder. This becomes the input to the LSTM decoder.

By letting the decoder have an attention mechanism (Bahdanau et al., 2014), the encoder is relieved from the burden of having to encode all information in the source sequence into a fixed-length vector. With attention, the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly. The attention mechanism computes a fixed-size vector that encodes the whole input sequence based on the sequence of all the outputs generated by the encoder as opposed to the plain encoder-decoder model which looks only at the last state generated by the encoder for all the slices of the decoder.

Thus, SCMIL is a sequence-to-sequence attention model with Bidirectional RNN encoder and attention decoder which is trained end-to-end having a character based representation on both encoder and decoder sides.

3.3 Training details

All the code is written in Python 2.7 using tf-seq2seq (Britz et al., 2017), a general-purpose encoder-decoder framework for Tensorflow (Abadi et al., 2016) deep learning library version 1.0.1. Both the encoder and the decoder are jointly trained end-to-end on the synthetic datasets we created. SCMIL has a learning rate of 0.001, batch size of 100, sequence length of 50 (characters) and number of training steps 10,000. The size of the encoder LSTM cell is 256 with one layer. The size of the decoder LSTM cell is 256 with two layers. We use Adam optimization (Kingma and Ba, 2014) for training SCMIL. The character embedding dimension are fixed to 256 and the dropout rate to 0.8.

4 Experiments and Results

We performed experiments with SCMIL and other models using synthetic datasets which we created for the Indic languages: Hindi and Telugu. Hindi is the most prominent Indian language and the third most spoken language in the world. Telugu is the most widely spoken Dravidian language in the world and third most spoken native language in India.

4.1 Dataset details

Due to lack of data with error patterns in Indic languages, we have built a synthetic dataset that SCMIL is trained on. Initially, we create data lists for Hindi and Telugu. For this, we have extracted a corpus of most frequent Hindi words[2] and most frequent Telugu words[3]. We have also extracted Hindi movie names and Telugu movie names of the movies released between the years 1930 and 2018 from Wikipedia which constitute phrases in the data lists. Thus, the Hindi and Telugu data lists consist of words and phrases consisting maximum of five words.

For each data instance in the data list, multiple noisy words are generated by introducing error. The type of errors include insertion, deletion, substitution of one character, and word fusing. Spaces between words are randomly dropped in phrases to simulate the word fusing problem. The list of errors for Hindi and Telugu is created by collecting the highly committed spelling errors users make in each of these languages. We created this error list from linguistic resources and with help from language experts. The language experts analyzed Hindi and Telugu usage and listed the most probable errors. These errors are based on observations on real data and lexicon of Hindi and Telugu. Thus, the synthetic datasets are made as close as possible to real world user-generated data.

Table 2 shows the example of generation of noisy words corresponding to a correct word considering a Hindi word. Thus, the pairs of noisy word and original word constitute the parallel data for training. Table 1 gives the details about size of the synthetic datasets for Hindi and Telugu.

4.2 Baseline Methods

We perform experiments on various models. The datasets are divided into train, dev, test partitions randomly in the ratio 80:10:10 respectively. In all our results, the models learn over the train partition, get tuned over the dev partition, and are evaluated over the test partition.

We train a SMT model using Moses (Koehn et al., 2007) on Hindi and Telugu synthetic datasets. This is our main baseline model. The standard set of features is used, including a phrase model, length penalty, jump penalty and a language model. This SMT model is trained at the character-level. Hence, the system learns mappings between character-level phrases. Moses framework allows us to easily conduct experiments with several settings and compare with SCMIL.

Other baselines are character based sequence-to-sequence attention models: CNN-GRU and GRU-GRU. All the models compared in this set of experiments look at batch sizes of 100 inputs and a maximum sequence size of 50 characters with a learning rate of 0.001 and run for 10000 steps. Throughout, the size of GRU cell is 256 with one layer. The CNN consists of 5 filters with sizes varying in the range of [2,3,4]. These multiple filters of particular widths produce the feature map, which is then concatenated and flattened for further processing.

4.3 Results and Analysis

Table 3 shows the accuracies reported by SCMIL and SMT methods. To check if Moses performs better on larger training data, we increased the size of Hindi synthetic dataset to 20567 and performed SMT. The accuracy value was 62% which is almost equivalent to the accuracy on original synthetic dataset. SCMIL outperforms Moses, an SMT technique by a huge margin. In Table 3, we find that SCMIL performs better than other sequence-to-sequence models with different convolutional and recurrent encoder-decoder combinations. These accuracies are on test set over the entire sequence. Thus, the results show that SCMIL performs better than all other baseline models. Our results support the conclusion by Britz et al. (2017) that LSTMs outperform GRUs.

The rule-based spell-checker for Hindi, HINSPELL (Singh et al., 2015) reported an accuracy of 77.9% on a data of 870 misspelled words randomly collected from books, newspapers and people etc. Te data used in Singh et al. (2015) and the HINSPELL system are not available. Hence, we

[2]https://ltrc.iiit.ac.in/download.php

[3]https://en.wiktionary.org/Frequency_lists/Telugu

Language	High Frequency Words	Movie names	Size of parallel corpus
Hindi	15000	3021	108587
Telugu	10000	3689	92716

Table 1: Details of the synthetic datasets for Hindi and Telugu.

Correct word	Noisy words
प्रेम कहानी (prem kahaanii)	प्रेमा कहानी (prema kahaanii) प्रेम कहनी (prem kahanii) प्रेमकहानी (premkahaanii) प्रेम कहानि (prem kahaani) प्रैम कहानी (praim kahaanii)

Table 2: Example of noisy words generation for a Hindi word with corresponding transliterations.

Model	Hindi(%)	Telugu(%)
Moses (SMT)	62.8	64.7
CNN-GRU	74.4	79.7
GRU-GRU	77.6	84.3
SCMIL	85.4	89.3

Table 3: Accuracy of spelling correction on Hindi and Telugu synthetic datasets given by Moses, character-based deep learning models(CNN-GRU and GRU-GRU), and SCMIL.

implemented HINSPELL using Shabdanjali dictionary[4] consisting of 32952 Hindi word. This system when tested on our Hindi synthetic dataset, gave an accuracy is 72.3%. This accuracy being lower than the original HINSPELL accuracy can be accounted to larger size of testing data and inclusion of out-of-vocabulary words. Thus, SCMIL outperforms HINSPELL by reporting an accuracy of 85.4%.

In Table 4, we have shown predictions given by SCMIL on few Hindi inputs. The results show that errors are contextually learned by the model. It can also be seen that the model has learned the word fusing problem. Also, the error detection step is not required separately as SCMIL trains end-to-end and learns the presence of noise in the input based on context. The advantage of SCMIL over rule-based techniques is that it can handle out of vocabulary words as it is trained at character level. For example, the last entry in the Table 4, is the English word *bomb* spelled in Hindi. The model has learned it and corrected when misspelled in Hindi as *bombh*. Dictionary based techniques fail in such cases. The model fails in few cases when it corrects one character of the misspelled word instead of other like the example in fourth row of 4.

The slightly higher accuracy of SCMIL on Telugu data than on Hindi data might be due to the fact that the highly probable spelling errors used in data creation are slightly less in number for Telugu when compared to Hindi. This can be handled for any language with more errors by increasing size of dataset which includes enough data instances capturing each kind of error.

[4]https://ltrc.iiit.ac.in/download.php

5 Future Work

This paper is the initial approach to automatic spelling correction for Indic languages using Deep Learning and we have obtained results that are competitive with the existing techniques. SCMIL can be improved and extended in many ways.

SCMIL presently deals with spelling corrections at word level. It can be further extended to automatically make not only spelling corrections but also grammar corrections at phrase level/sentence level.

The synthetic dataset can be improved by collecting noisy words from different platforms like social media, blogs etc. and introducing these real world errors into clean corpus. This will improve the performance of the model on user-generated data. Further, for proper evaluation of the model, The model should be tested on real world user-generated parallel data.

One more potential improvement would be to change the training data from words and phrases to sentences. This will help in achieving context based spelling correction.

The spelling correction model can be extended to a text correction and completion model. Changing the decoder from character-level to word-level will add the functionality of auto completion. This will improve the scope of the model in various applications.

Input	Prediction	Correct Output
राजसि (raajasi)	राजसी (raajasii)	राजसी (raajasii)
दोआँखें (dhoaankhei)	दो आँखें (dho aankhei)	दो आँखें (dho aankhei)
अंस (ans)	अंश (ansh)	अंश (ansh)
दशावतर (dhashaavatar)	दशवतर (dhashavatar)	दशावतार (dhashaavataar)
बॉम्भ (baambh)	बॉम्ब (baamb)	बॉम्ब (baamb)
बॉम्भ (baambh)	बॉम्ब (baamb)	बॉम्ब (baamb)

Table 4: Qualitative evaluation of predictions by SCMIL on few Hindi inputs along with expected outputs and corresponding transliterations.

6 Summary and Conclusion

In this paper, we proposed SCMIL for automatic spelling correction in Indic languages which employs a recurrent sequence-to-sequence attention model to learn the spelling corrections from noisy and correct word pairs. We created a parallel corpus of noisy and correct spellings for training by introducing spelling errors in correct words. We validated SCMIL on these synthetic datasets created for Hindi and Telugu. We implemented spelling correction using Moses, a SMT system as a baseline model. We evaluated our system against existing techniques for Indic languages and showed favorable results. Finally, we discussed possible extensions to improve the scope of SCMIL and perform better evaluation.

SCMIL can be used in applications like search engines as we have shown that it automatically corrects the input text in Indic languages. Most of the deep learning models train on billions of data instances. On the contrary, SCMIL trains on a dataset of less than a million parallel instances and gives competitive results. This shows that our approach can be used for automatic spelling correction of any resource-scarce language.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Farooq Ahmad and Grzegorz Kondrak. 2005. Learning a spelling error model from search query logs. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 955–962. Association for Computational Linguistics.

Ambili, Panchami KS, and Neethu Subash. 2016. Automatic error detection and correction in malayalam. *International Journal of Science Technology and Engineering(IJSTE)*, 3(2).

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Yonatan Belinkov and Yonatan Bisk. 2017. Synthetic and natural noise both break neural machine translation. *arXiv preprint arXiv:1711.02173*.

D. Britz, A. Goldie, T. Luong, and Q. Le. 2017. Massive Exploration of Neural Machine Translation Architectures. *ArXiv e-prints*.

Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Veena Dixit, Satish Dethe, and Rushikesh K Joshi. 2005. Design and implementation of a morphology-based spellchecker for marathi, an indian language. *ARCHIVES OF CONTROL SCIENCE*, 15(3):301.

Shaona Ghosh and Per Ola Kristensson. 2017. Neural networks for text correction and completion in keyboard decoding. *arXiv preprint arXiv:1709.06429*.

Neha Gupta and Pratistha Mathur. 2012. Spell checking techniques in nlp: a survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(12).

Saša Hasan, Carmen Heger, and Saab Mansour. 2015. Spelling correction of user search queries through statistical machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 451–460.

Sepp Hochreiter. 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116.

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

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

Rakesh Kumar, Minu Bala, and Kumar Sourabh. 2018. A study of spell checking techniques for indian languages. *JK Research Journal in Mathematics and Computer Sciences*, 1(1).

Uma Maheshwar Rao. 2011. Telugu spell-checker. *International Telugu Internet Conference Proceedings*.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature*, 323(6088):533.

Keisuke Sakaguchi, Kevin Duh, Matt Post, and Benjamin Van Durme. 2017. Robsut wrod reocginiton via semi-character recurrent neural network. In *AAAI*, pages 3281–3287.

Amit Sharma and Pulkit Jain. 2013. Hindi spell checker. *Indian Institute of Technology Kanpur*.

Harsharndeep Singh et al. 2015. Design and implementation of hinspell-hindi spell checker using hybrid approach. *International Journal of Scientific Research and Management*, 3(2).

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

W John Wilbur, Won Kim, and Natalie Xie. 2006. Spelling correction in the pubmed search engine. *Information retrieval*, 9(5):543–564.