# NeuSpell: A Neural Spelling Correction Toolkit

**Sai Muralidhar Jayanthi, Danish Pruthi, Graham Neubig**
Language Technologies Institute
Carnegie Mellon University
{sjayanth, ddanish, gneubig}@cs.cmu.edu

## Abstract

We introduce NeuSpell, an open-source toolkit for spelling correction in English. Our toolkit comprises ten different models, and benchmarks them on naturally occurring misspellings from multiple sources. We find that many systems do not adequately leverage the context around the misspelt token. To remedy this, (i) we train neural models using spelling errors in context, synthetically constructed by reverse engineering isolated misspellings; and (ii) use contextual representations. By training on our synthetic examples, correction rates improve by 9% (absolute) compared to the case when models are trained on randomly sampled character perturbations. Using richer contextual representations boosts the correction rate by another 3%. Our toolkit enables practitioners to use our proposed and existing spelling correction systems, both via a unified command line, as well as a web interface. Among many potential applications, we demonstrate the utility of our spell-checkers in combating adversarial misspellings. The toolkit can be accessed at neuspell.github.io.[1]

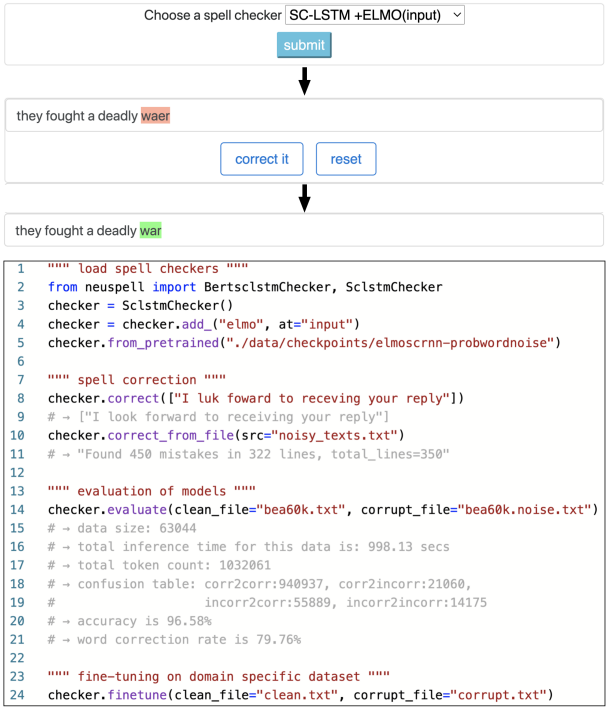


Figure 1: Our toolkit's web and command line interface for spelling correction.

## 1 Introduction

Spelling mistakes constitute the largest share of errors in written text (Wilbur et al., 2006; Flor and Futagi, 2012). Therefore, spell checkers are ubiquitous, forming an integral part of many applications including search engines, productivity and collaboration tools, messaging platforms, etc. However, many well performing spelling correction systems are developed by corporations, trained on massive proprietary user data. In contrast, many freely available off-the-shelf correctors such as Enchant (Thomas, 2010), GNU Aspell (Atkinson, 2019), and JamSpell (Ozinov, 2019), do not effectively use the context of the misspelled word. For instance, they fail to disambiguate thaught to *taught* or *thought* based on the context: "Who thaught you calculus?" versus "I never thaught I would be awarded the fellowship."

In this paper, we describe our spelling correction toolkit, which comprises of several neural models that accurately capture context around the misspellings. To train our neural spell correctors, we first curate synthetic training data for spelling correction *in context*, using several text noising strategies. These strategies use a lookup table for word-level noising, and a context-based character-level confusion dictionary for character-level noising. To populate this lookup table and confusion matrix, we harvest isolated misspelling-correction pairs from various publicly available sources.

[1]Code and pretrained models are available at: https://github.com/neuspell/neuspell

Further, we investigate effective ways to incorporate contextual information: we experiment with contextual representations from pretrained models such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2018) and compare their efficacies with existing neural architectural choices (§ 5.1).

Lastly, several recent studies have shown that many state-of-the-art neural models developed for a variety of Natural Language Processing (NLP) tasks easily break in the presence of natural or synthetic spelling errors (Belinkov and Bisk, 2017; Ebrahimi et al., 2017; Pruthi et al., 2019). We determine the usefulness of our toolkit as a countermeasure against character-level adversarial attacks (§ 5.2). We find that our models are better defenses to adversarial attacks than previously proposed spell checkers. We believe that our toolkit would encourage practitioners to incorporate spelling correction systems in other NLP applications.

| Model | Correction Rates | Time per sentence (milliseconds) |
|---|---|---|
| ASPELL (Atkinson, 2019) | 48.7 | 7.3* |
| JAMSPELL (Ozinov, 2019) | 68.9 | **2.6*** |
| CHAR-CNN-LSTM (Kim et al., 2015) | 75.8 | 4.2 |
| SC-LSTM (Sakaguchi et al., 2016) | 76.7 | **2.8** |
| CHAR-LSTM-LSTM (Li et al., 2018) | 77.3 | 6.4 |
| BERT (Devlin et al., 2018) | 79.1 | 7.1 |
| SC-LSTM | | |
| +ELMO (input) | **79.8** | 15.8 |
| +ELMO (output) | 78.5 | 16.3 |
| +BERT (input) | 77.0 | 6.7 |
| +BERT (output) | 76.0 | 7.2 |

Table 1: Performance of different correctors in the NeuSpell toolkit on the BEA-60K dataset with real-world spelling mistakes. * indicates evaluation on a CPU (for others we use a GeForce RTX 2080 Ti GPU).

## 2 Models in NeuSpell

Our toolkit offers ten different spelling correction models, which include: (i) two off-the-shelf non-neural models, (ii) four published neural models for spelling correction, (iii) four of our extensions. The details of first six systems are following:

- GNU Aspell (Atkinson, 2019): It uses a combination of metaphone phonetic algorithm,[2] Ispell's near miss strategy,[3] and a weighted edit distance metric to score candidate words.
- JamSpell (Ozinov, 2019): It uses a variant of the SymSpell algorithm,[4] and a 3-gram language model to prune word-level corrections.
- SC-LSTM (Sakaguchi et al., 2016): It corrects misspelt words using semi-character representations, fed through a bi-LSTM network. The semi-character representations are a concatenation of one-hot embeddings for the (i) first, (ii) last, and (iii) bag of internal characters.
- CHAR-LSTM-LSTM (Li et al., 2018): The model builds word representations by passing its individual characters to a bi-LSTM. These representations are further fed to another bi-LSTM trained to predict the correction.
- CHAR-CNN-LSTM (Kim et al., 2015): Similar to the previous model, this model builds word-level representations from individual characters using a convolutional network.
- BERT (Devlin et al., 2018): The model uses a pre-trained transformer network. We average the sub-word representations to obtain the word representations, which are further fed to a classifier to predict its correction.

To better capture the context around a misspelt token, we extend the SC-LSTM model by augmenting it with deep contextual representations from pre-trained ELMo and BERT. Since the best point to integrate such embeddings might vary by task (Peters et al., 2018), we append them either to semi-character embeddings before feeding them to the biLSTM or to the biLSTM's output. Currently, our toolkit provides four such trained models: ELMo/BERT tied at input/output with a semi-character based bi-LSTM model.

**Implementation Details** Neural models in NeuSpell are trained by posing spelling correction as a sequence labeling task, where a correct word is marked as itself and a misspelt token is labeled as its correction. Out-of-vocabulary labels are marked as `UNK`. For each word in the input text sequence, models are trained to output a probability distribution over a finite vocabulary using a softmax layer.

We set the hidden size of the bi-LSTM network in all models to 512 and use {50,100,100,100} sized convolution filters with lengths {2,3,4,5} respectively in CNNs. We use a dropout of 0.4 on the bi-LSTM's outputs and train the models using cross-entropy loss. We use the BertAdam[5] optimizer for models with a BERT component and the

[2] http://aspell.net/metaphone/
[3] https://en.wikipedia.org/wiki/Ispell
[4] https://github.com/wolfgarbe/SymSpell
[5] github.com/cedrickchee/pytorch-pretrained-BERT

Adam (Kingma and Ba, 2014) optimizer for the remainder. These optimizers are used with default parameter settings. We use a batch size of 32 examples, and train with a patience of 3 epochs.

During inference, we first replace UNK predictions with their corresponding input words and then evaluate the results. We evaluate models for accuracy (percentage of correct words among all words) and word correction rate (percentage of misspelt tokens corrected). We use AllenNLP[6] and Huggingface[7] libraries to use ELMo and BERT respectively. All neural models in our toolkit are implemented using the Pytorch library (Paszke et al., 2017), and are compatible to run on both CPU and GPU environments. Performance of different models are presented in Table 1.

## 3 Synthetic Training Datasets

Due to scarcity of available parallel data for spelling correction, we noise sentences to generate misspelt-correct sentence pairs. We use 1.6M sentences from the one billion word benchmark (Chelba et al., 2013) dataset as our clean corpus. Using different noising strategies from existing literature, we noise ∼20% of the tokens in the clean corpus by injecting spelling mistakes in each sentence. Below, we briefly describe these strategies.

**RANDOM:** Following Sakaguchi et al. (2016), this noising strategy involves four character-level operations: permute, delete, insert and replace. We manipulate only the internal characters of a word. The permute operation jumbles a pair of consecutive characters, delete operation randomly deletes one of the characters, insert operation randomly inserts an alphabet and replace operation swaps a character with a randomly selected alphabet. For every word in the clean corpus, we select one of the four operations with 0.1 probability each. We do not modify words of length three or smaller.

**WORD:** Inspired from Belinkov and Bisk (2017), we swap a word with its noised counterpart from a pre-built lookup table. We collect 109K misspelt-correct word pairs for 17K popular English words from a variety of public sources.[8]

For every word in the clean corpus, we replace it by a random misspelling (with a probability of 0.3) sampled from all the misspellings associated with that word in the lookup table. Words not present in the lookup table are left as is.

**PROB:** Recently, Piktus et al. (2019) released a corpus of 20M correct-misspelt word pairs, generated from logs of a search engine.[9] We use this corpus to construct a character-level confusion dictionary where the keys are ⟨character, context⟩ pairs and the values are a list of potential character replacements with their frequencies. This dictionary is subsequently used to sample character-level errors in a given context. We use a context of 3 characters, and backoff to 2, 1, and 0 characters. Notably, due to the large number of unedited characters in the corpus, the most probable replacement will often be the same as the source character.

**PROB+WORD:** For this strategy, we simply concatenate the training data obtained from both WORD and PROB strategies.

## 4 Evaluation Benchmarks

**Natural misspellings in context** Many publicly available spell-checkers correctors evaluate on isolated misspellings (Atkinson, 2019; Mitton, na; Norvig, 2016). Whereas, we evaluate our systems using misspellings in context, by using publicly available datasets for the task of Grammatical Error Correction (GEC). Since the GEC datasets are annotated for various types of grammatical mistakes, we only sample errors of SPELL type.

Among the GEC datasets in BEA-2019 shared task[10], the Write & Improve (W&I) dataset along with the LOCNESS dataset are a collection of texts in English (mainly essays) written by language learners with varying proficiency levels (Bryant et al., 2019; Granger, 1998). The First Certificate in English (FCE) dataset is another collection of essays in English written by non-native learners taking a language assessment exam (Yannakoudakis et al., 2011) and the Lang-8 dataset is a collection of English texts from Lang-8 online language learning website (Mizumoto et al., 2011; Tajiri et al., 2012). We combine data from these four sources to create the BEA-60K test set with nearly 70K spelling mistakes (6.8% of all tokens) in 63044 sentences.

The JHU FLuency-Extended GUG Corpus (JFLEG) dataset (Napoles et al., 2017) is another

[6] allennlp.org/elmo

[7] huggingface.co/transformers/model_doc/bert.html

[8] https://en.wikipedia.org/, dcs.bbk.ac.uk, norvig.com, corpus.mml.cam.ac.uk/efcamdat

[9] https://github.com/facebookresearch/moe

[10] www.cl.cam.ac.uk/research/nl/bea2019st/