Hierarchical Character Tagger for Short Text Spelling Error Correction

Mengyi Gao [*]
eBay Inc.
menggao@ebay.com

Canran Xu [*]
eBay Inc.
canxu@ebay.com

Peng Shi [*]
eBay Inc.
pshi@ebay.com

Abstract

State-of-the-art approaches to spelling error correction problem include Transformer-based Seq2Seq models, which require large training sets and suffer from slow inference time; and sequence labeling models based on Transformer encoders like BERT, which involve token-level label space and therefore a large pre-defined vocabulary dictionary. In this paper we present a Hierarchical Character Tagger model, or HCTagger, for short text spelling error correction. We use a pre-trained language model at the character level as a text encoder, and then predict character-level edits to transform the original text into its error-free form with a much smaller label space. For decoding, we propose a hierarchical multi-task approach to alleviate the issue of long-tail label distribution without introducing extra model parameters. Experiments on two public misspelling correction datasets demonstrate that HCTagger is an accurate and much faster approach than many existing models.

1 Introduction

A spelling corrector is an important and universal tool for a wide range of text-related applications, such as search engines, machine translation, optical character recognition, medical records, text processors and essay scoring. Although spelling error correction is a long-studied problem, it remains a challenging task because words can be misspelled in a variety of forms, including in-word errors, cross-word errors, non-word errors and real-word errors, depending on the subtle contextual information.

In this paper, we focus on solving the spelling correction problem in user-generated short text, such as queries in search engines or tweets on social media, which has three unique properties compared to long essays. First, search queries or tweets are often short and lack context. Second, most short text contains pure spelling errors and almost no grammatical errors. Third, instant spell checkers used in search engines or social medias have strict latency requirements.

In general, popular approaches to spelling correction make use of parallel corpora in which the source sentence contains spelling errors and the target sentence is error-free. Recently, the Transformer-based sequence-to-sequence (Seq2Seq) model (Vaswani et al., 2017) has gradually proven to be effective on spelling correction problems. Unlike neural machine translation, spelling errors tend to occur locally for a few characters while the rest of the text is correct. To cope with this situation, Zhao et al. propose a scheme to incorporate a copy mechanism in Seq2Seq. The success of this type of Seq2Seq model depends on large scale annotated datasets, which are often generated by constructing text noise from clean text in previous studies. Moreover, this approach suffers from slower inference time and lack of interpretability.

Another class of approaches is based on sequence labeling. Instead of generating the output sequence in an autoregressive fashion, PIE (Awasthi et al., 2019) and GECToR (Omelianchuk et al., 2020) predicts token-level edit operations in one of {*Keep*, *Delete*, *Replace*, *Append*} by leveraging pre-trained Transformer encoders, such as BERT (Devlin et al., 2019). Such models can generate the outputs for all tokens in parallel, and therefore significantly reduces the latency of sequential decoding as in Seq2Seq models while achieving comparable accuracy. However, the approaches in both papers predict edit operations at token level. It can be expected that the *Replace* and *Append* operations are associated with a huge pre-defined vocabulary dictionary. For real-life usages it is infeasible to enumerate all correctly spelled words in the label space.

To address the aforementioned shortcomings,

[*] Equal Contribution

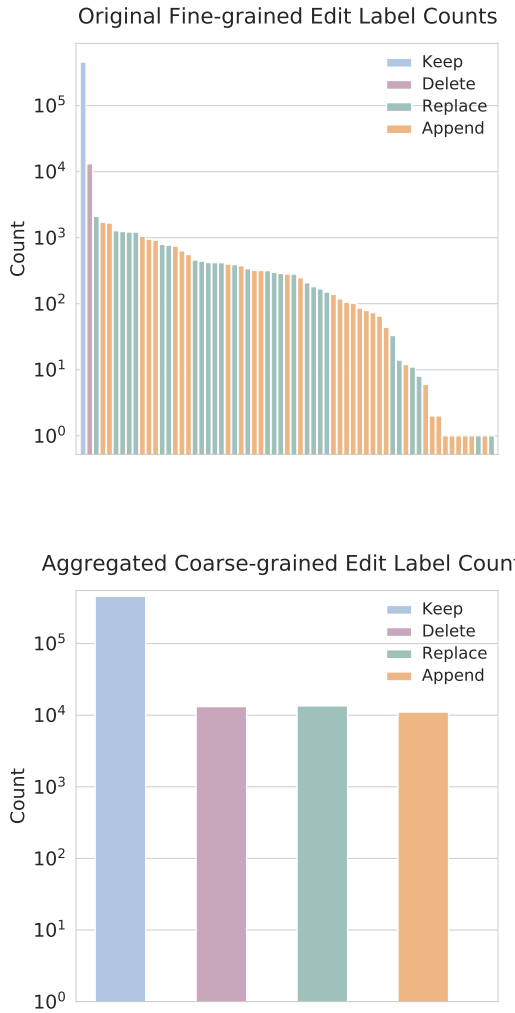

Figure 1: Original and aggregated edit label counts. The upper plot shows original fine-grained edit label counts, which are heavily skewed. The lower plot of aggregated coarse-grained edit label counts has much less skewness.

considering the unique properties of short text misspelling correction, in this paper, we propose a new model called Hierarchical Character Tagger, or HCTagger for short, which uses a pre-trained language model at the character level as a text encoder, and then predicts character-level edits. It is motivated by the straightforward observation that spelling errors usually occur at character level. For the misspelling-correction pair of *shies* → *shoes*, its character-level edit labels would be [*s: Keep*, *h: Keep*, *i: Replace with* ***o***, *e: Keep*, *s: Keep*], which is represented in a much smaller label space compared to [*shies: Replace with* ***shoes***] at token level. While most spelling errors occur within 1-edit distance for each token, for broader coverage we also include character sequence edit operations like *Replace with* ***oa***.

Furthermore, the distribution of edit labels is long-tailed. As shown in Figure 1, *Keep* and *Delete* are more frequent than labels of *Replace/Append with certain character(s)*. If these labels are treated as equivalent, the overall accuracy of the model will be constrained by the unevenness of the label distribution. Therefore, for decoding, we aggregate original fine-grained edit labels into four coarse-grained labels [*Keep*, *Delete*, *Append*, *Replace*]. We propose a hierarchical multi-task approach to learn both the fine-grained and coarse-grained edit labels at the same time without introducing any extra model parameters.

Through extensive experiments on two public datasets, we demonstrate that our proposed HCTagger effectively improves the performance and latency of short text spelling correction.

2 Approach

We describe our model HCTagger in this section.

2.1 Problem Formulation

Without lack of universality of language types, for an input text sequence with spelling error, $S = [c_1, \ldots, c_n]$, our goal is to get the correct spelling of the corresponding text, denoted as $T = [d_1, \ldots, d_m]$, where c_i and d_i are character level input and output. Note that the sequence lengths n and m are not necessarily equal.

To map the source sequence S to target T, a corresponding edit operation sequence $O = [o_1, \ldots, o_n]$ is applied. Note that O has the same sequence length as S. Each edit operation, o_i, falls into one of the following four categories:

Keep The current character remains unchanged. This means that the current character is not misspelled.

Delete The current character is deleted.

Append Append a sequence of characters of length greater than or equal to one after the current character. Each distinct appended sequence is treated as an independent tag type.

Replace Replaces the current character with a number of characters of length greater than or equal to 1. Similar to *Append*, each distinct appended sequence is treated as an independent tag type.

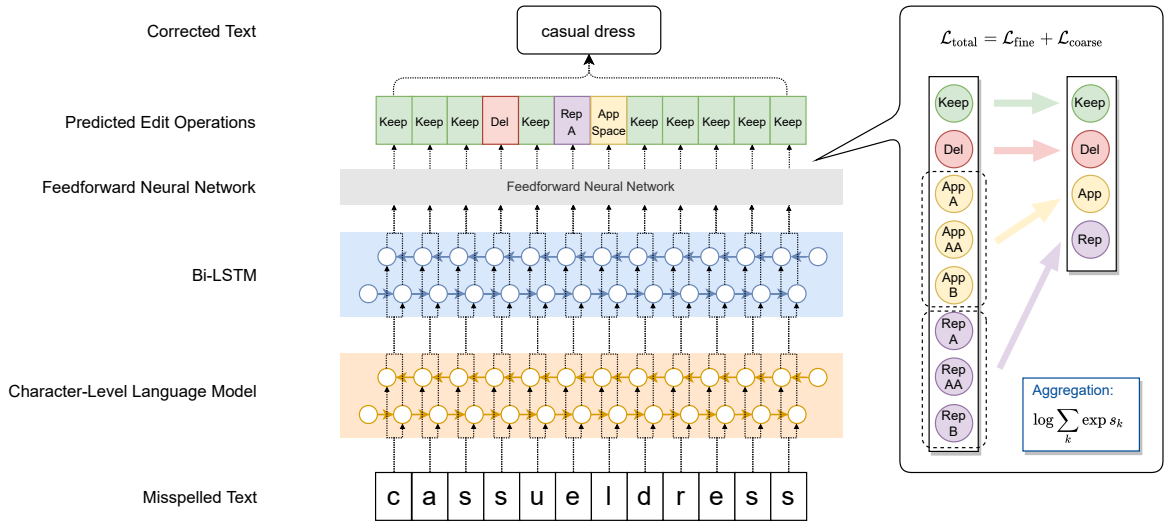

Figure 2: Overall architecture of the model. The text encoder is a character-level language model, followed by a bi-directional LSTM. The edit operations predicted by the feedforward neural network are used to formulate the corrected text. During training, the output of the feedforward neural network is used to construct the hierarchical loss function with two explicit terms.

Note that there could be more than one possible edit operation sequences to transform from source S to target T. We use Python function *SequenceMatcher* in module *difflib* to do obtain the unique edit operation label sequence O. The idea of *SequenceMatcher* is to find the longest contiguous matching subsequence. This does not necessarily yield minimal edit sequences, but does tend to yield matches that "look right" to humans. For more information, refer to the doc [1].

Thus, we eventually transform spelling correction into a sequential labeling problem, i.e., for a given input $S = [c_1, \ldots, c_n]$, predict the edit operation o_i for each character c_i. As a concrete example, to map a misspelled input text *cassueldress* to its correction *casual dress*, 3 edit operations are required, namely (1) deleting the 4th character *s*, (2) replacing the 6th character *e* with *a*, and (3) appending a *space* after the 7th character *l*, while keeping other characters unchanged.

2.2 Model

Our proposed model, HCTagger, consists of two components. First, we encode the text by pre-training a character-level language model. Second, the representation obtained by the language model is encoded by a bi-directional LSTM, which is then fed to a decoder. This decoder is hierarchical: it decodes simultaneously four coarse-grained labels [*Keep*, *Delete*, *Append*, *Replace*] and all the fine-grained tags (such as *Append with* ***a*** or *Replace with* ***eo***), which could be of potentially up to thousands types. The architecture of the model is shown in Figure 2.

Character-level Language Model The character-level language model we use is the pretrained Flair (Akbik et al., 2018), which has been widely shown to be effective for word-level sequence labeling tasks. Specifically, Flair consists of a character-level embedding layer and a (possibly bi-directional) LSTM layer. The model predicts the next character by the preceding or succeeding character inputs. The authors argue that it can capture semantic differences in morphological similarities, as well as contextual information for polysemous words. Moreover, character-level models better handle rare and misspelled words as well as model subword structures such as prefixes and endings.

Though pre-training a character-level language model, in the original paper Flair focuses on word-level sequence labeling task (e.g., NER). Specifically, to obtain word-level embeddings from character-level language model, Flair uses the output hidden state after the last character in each word as the representation of the whole word. However, in our scenario, we use the embedding of current character to predict its own edit operation, regardless of which word it belongs to, even if it is a space or punctuation.

In addition, when using Flair as the text encoder,

[1] https://docs.python.org/3/library/difflib.html

# Iteration	Short Text	# Token-level Errors	# Character-level Errors
Original	fashien industrie	2	3
1	fashion industry	0	0

Table 1: An illustrative example of iterative inference.

we found that fine-tuning Flair's language model parameters along with the sequence labeling task generally perform better than without fine-tuning. Therefore, fine-tuning Flair is our default setting whenever possible.

Hierarchical Multi-Task In a training set of finite size, the original fine-grained edit labels (a certain character being appended or replaced with some characters) form a long-tail distribution, as shown in Figure 1. This makes some relatively rare spelling errors more difficult to be corrected.

For decoding, we feed the hidden states of the bidirectional LSTM into a layer of feedforward neural network whose output dimension is the size of label types. For character c_i, the probability of original fine-grained label type k is $P(k|c_i)$. To alleviate the issue of long-tail distribution for k, we propose to aggregate the probabilities for four coarse-grained edit labels, denoted as $P(v|c_i)$, with $v \in \{\textit{Keep}, \textit{Delete}, \textit{Replace}, \textit{Append}\}$, which are presumably more balanced than the fine-grained labels. To achieve this, we use the rule of sum of probability: as all possible fine-grained *Append* (*Replace*) operations are mutually exclusive, the sum of their probabilities should equal the coarse-grained probability of *Append* (*Replace*). Formally,

$$P(A(R)|c_i) = \sum_{k \in A(R)_{\subset}} P(k|c_i), \qquad (1)$$

where $A_{\subset}$ and $R_{\subset}$ are the subsets consisting of fine-grained *Append* and *Replace* operations, correspondingly.

Denote the logits for original fine-grained tag type k as f_k, and logits for aggregated coarse-grained tag type v as l_v. Then the probability of label type k is $P(k|c_i) = \text{softmax}(f_k)$. Similarly we have $P(v|c_i) = \text{softmax}(l_v)$. Therefore, Equation (1) can be derived as:

$$\frac{\exp\left(l_{A(R)}\right)}{\sum_{m \in \{K,D,A,R\}} \exp\left(l_m\right)} = \sum_{k \in A(R)_{\subset}} \frac{\exp\left(f_k\right)}{\sum_j \exp\left(f_j\right)}, \qquad (2)$$

where K, D, A and R are the short forms for *Keep*, *Delete*, *Append* and *Replace*, accordingly.

As a result, we obtain the coarse-grained logits l_v by solving Equation (2):

$$l_v = \begin{cases} f_k & k = \textit{Keep} \\ f_k & k = \textit{Delete} \\ \log \sum_{k \in A_{\subset}} \exp\left(f_k\right) & k \in A_{\subset} \\ \log \sum_{k \in R_{\subset}} \exp\left(f_k\right) & k \in R_{\subset} \end{cases} \qquad (3)$$

Finally, HCTagger is trained by using the following multi-task loss associated with predicting edit o_i at each character c_i:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{fine}} + \mathcal{L}_{\text{coarse}} \\ &= -\sum_i \log P(f_k^{(i)}|c_i) - \sum_i \log P(l_v^{(i)}|c_i). \end{aligned}$$

Notice that, in contrast to traditional multi-task learning, with the relation between l_v and f_k in Equation (3), the coarse-grained loss function we introduce as an auxiliary task does not contain extra model parameters. The advantage of this design is that both fine-grained and coarse-grained loss functions can reach the optimum at the same time without additional efforts to tune the parameters to balance the two terms.

Inference Some previous studies (Awasthi et al., 2019; Omelianchuk et al., 2020) on Grammar Error Correction (GEC) have shown that a well-established approach for inference is iterative: use the modified result obtained by the model in the current round as input for next round's prediction, and repeat the process several times. These studies find that due to the dependency among grammatical errors (tense, pronoun, subject-verb, preposition, plurals), the performance of model predictions can be steadily improved by multiple iterations. However, iterative inference is confronted with a trade-off between speed and accuracy.

As spelling errors impose a strong notion of locality and have weaker dependency on each other than grammatical errors, the iterative correction process is less necessary. For example, in the example shown in Table 1, the 2 token-level typos, *fashien* and *industrie*, are independent of each other. To this end, the two errors can be corrected simultaneously through a single run in our model.

Dataset	# Train	# Dev	# Test	% Error Rate	# Label Types
Twitter	31,172	4,000	4,000	100	66
Webis	44,772	5,000	5,000	17	112

Table 2: Basic statistics of the datasets.

Indeed, from our experiments, we noticed that having more than one round of inference iteration only marginally improves the accuracy in our task. Therefore, we report the results for HCTagger with only *one* inference iteration in all experiments.

3 Experiments

In this section, we describe the experiments performed on two public datasets for HCTagger. Meanwhile, we compare it with several state-of-the-art baselines.

3.1 Datasets

We conduct experiments on the following two short text datasets:

Twitter Dataset is proposed in Aramaki (2010), which includes 39,172 samples in their spell-error form and error-free form. We adopt the same train / dev / test split as Ribeiro et al. (2018) and Awasthi et al. (2019).

Webis Dataset is introduced in Hagen et al. (2017), which consists of 54,772 queries from AOL search logs. In contrast to the Twitter Dataset, the error rate of this dataset is only $\sim$17%. Since the original dataset does not provide the train / dev / test split, we randomly sample 5000 queries as the dev and test sets, respectively, and use the remaining data as the training set.

The basic statistics of these two datasets and the corresponding number of label types (calculated from training data) are listed in the Table 2.

3.2 Baselines and Implementation Details

The following baseline models are used for the comparison experiments:

Aspell (Atkinson, 2018) works at word level. It uses a combination of metaphone phonetic algorithm, Ispell's near miss strategy and a weighted edit distance metric to score candidate words.

Seq2Seq-LSTM is the standard LSTM-based Seq2Seq architecture.

Seq2Seq-Transformer (Vaswani et al., 2017) is the self-attention based Seq2Seq model.

Local Sequence Transduction (Ribeiro et al., 2018) treats spelling correction as a character-level local sequence transduction task by first predicting insertion slots, followed by a sequence labeling task for output tokens or a special operation *Delete*.

BERT-PIE (Awasthi et al., 2019) or Parallel Iterative Edit model, is a sequence labeling method which uses BERT as its text encoder.

BERT-Neuspell (Jayanthi et al., 2020) is provided by the Neuspell toolkit. It regards spelling correction as a token-level sequence labeling task, where the output for each token is its error-free form. We finetune the BERT model on the Webis dataset.

All the models are implemented in PyTorch (Paszke et al., 2019), and trained with a single Tesla V100 GPU. For HCTagger, we use the English Flair embeddings pretrained on the 1-billion word corpus (Chelba et al., 2014), which are publicly available[2]. We tune the number of LSTM hidden states $\in \{512, 1024\}$, training batch size $\in \{8, 16, 32\}$, learning rate $\in \{1e^{-2}, 1e^{-3}\}$, and optimizer type $\in$\{Adam (Kingma and Ba, 2015), LAMB (You et al., 2020)\}. In addition, both the encoder and decoder of Transformer has two self-attention layers.

3.3 Results

For the Twitter dataset, to align with previous publications, we report the accuracy in the test set to compare the performances among all models. As shown in Table 3, HCTagger improves accuracy over all the models except Transformer. In particular, it is important to note that although the pretrained language model (Flair) we use is lightweight compared to BERT, our model still outperforms BERT-PIE.

Table 3 also reports the performance on the Webis dataset. Our HCTagger exceeds all other models. Transformer model doesn't perform well on

[2] https://github.com/flairNLP/flair

Model	Twitter Dataset Accuracy	Webis Dataset Accuracy
Aspell	30.1 †	65.8
Seq2Seq (LSTM)	52.2 *	83.5
Seq2Seq (Transformer)	**67.6** *	83.7
Ribeiro et al. (2018)	64.6 †	-
BERT-PIE (Awasthi et al., 2019)	67.0 *	-
BERT-Neuspell (Jayanthi et al., 2020)	-	84.0
HCTagger	67.2	**86.8**

Table 3: Performance on Twitter and Webis dataset. Results with † are from Ribeiro et al. (2018); results with ∗ are from Awasthi et al. (2019).

Model	Words per Second
Seq2Seq (Transformer)	36.62
BERT-PIE (Awasthi et al., 2019)	80.43
HCTagger	**251.20**

Table 4: Inference speed on Twitter dataset.

Model	Query per Second
Seq2Seq (LSTM)	83.33
Seq2Seq (Transformer)	40.00
BERT-Neuspell (Jayanthi et al., 2020)	62.50
HCTagger	**250.00**

Table 5: Inference speed on Webis dataset.

this dataset, probably because the number of misspelled queries is small (17%) and is not enough to train Transformer well. In contrast, our model makes more effective use of small training set.

Meanwhile, we also compare the inference speed of the most accurate models, as shown in Table 4 and Table 5. Indeed, the inference speed of HCTagger is much faster than Seq2Seq (LSTM, Transformer), BERT-PIE (Awasthi et al., 2019), and BERT-Neuspell (Jayanthi et al., 2020).

3.4 Ablation Study

To understand the importance of each part of the model, we conduct an ablation study on the Twitter dataset, and report the accuracy in Table 6.

We first take away the pre-trained language model. At this point, the character-level embedding is randomly initialized and the rest of the model is left unchanged. The accuracy decreases by 1.9%.

Subsequently, we preserve the language model but remove the coarse-grained loss term of the Hierarchical Multi-Task. In this case, the accuracy decreases by 0.7%.

	Accuracy
Full Model	67.2
- w/o Pretrained LM	65.3
- w/o Hierarchical Multi-Task	66.5

Table 6: Ablation study on Twitter dataset.

4 Related Works

Hasan et al. use character-based statistical machine translation to correct user queries in the e-commerce domain. They extract training data from query refinement logs, and evaluate the results on an internal dataset.

Grammar Error Correction (GEC) is an extensively researched NLP task. This task contains grammar errors, including spelling, punctuation, grammatical, and word choice errors. PIE (Awasthi et al., 2019) and GECToR (Omelianchuk et al., 2020) are the state-of-the-art models that predict token-level edit operations {*Keep*, *Delete*, *Replace*, *Insert*} by leveraging pre-trained Transformer encoders like BERT. However, their models are not specifically designed for correcting spelling errors, which most often occur at character level. They rely on a small (∼1k) pre-defined token-level *Replace* and *Insert* dictionary. Including all correctly-spelled tokens in the dictionary will make the label space too large.

Transformer based Seq2Seq models (Kiyono et al., 2019; Zhao et al., 2019) prove to be successful on grammar error corrections, but heavily depends on synthetically generated error datasets. Character based Seq2Seq models (Xie et al., 2016) are also explored. Such model architectures involve a separate autoregressive decoder and attention module, which makes the inference time much slower. In particular for spelling error correction task, where the misspelling and correction only differ by one or more characters, Seq2Seq models seem too heavy.

Neuspell (Jayanthi et al., 2020) is a spelling correction toolkit, which implements a wide range of models like SC-Elmo-LSTM and BERT. They regard spelling correction as a token-level sequence labeling task, where the output for each token is its error-free form. For each word in the input text sequence, models are trained to output a probability distribution over a finite vocabulary. Besides the aforementioned excessive label space problem at token-level, another shortcoming of this toolkit is that it assumes the misspelled and correction sentences have exactly the same number of tokens. Therefore, cross-word errors such as *power point* $\rightarrow$ *powerpoint* or *babydoll* $\rightarrow$ *baby doll* cannot be handled properly.

Ribeiro et al. treat spelling correction as a character-level local sequence transduction task by first predicting insertion slots in the input using learned insertion patterns, and then using a sequence labeling task to output tokens or a special token *Delete*. They maintain a dictionary to keep track of the insertion context. For example, letter *a* is inserted frequently after letter *s*. While our pretrained language model layer implicitly encodes such insertion context without the need of keeping a dictionary.

5 Conclusions

We presented the Hierarchical Character Tagger to correct user-generated short text misspellings. HCTagger predicts character-level edits, which has smaller label space than token-level edits. Pretrained character-level language model embedding that we use is lightweight and much faster than BERT-like text encoders in many other state-of-the-art models, while achieving similar or even higher accuracy for short text spelling error correction task. Moreover, our novel Hierarchical Multi-Task decoding approach can be extended to any scenario that contains a hierarchical long-tail distributed label space.

Acknowledgements

We would like to thank Zhe Wu, Julie Netzloff, Xiaoyuan Wu, Hua Yang, Vivian Tian and Scott Gaffney for their support.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 1638–1649. Association for Computational Linguistics.

Eiji Aramaki. 2010. Typo corpus.

Kevin Atkinson. 2018. Gnu aspell.

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 4259–4269. Association for Computational Linguistics.

Ciprian Chelba, Tomás Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 2635–2639. ISCA.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Matthias Hagen, Martin Potthast, Marcel Gohsen, Anja Rathgeber, and Benno Stein. 2017. A large-scale query spelling correction corpus. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 1261–1264. ACM.

Sasa 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, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 451–460. The Association for Computational Linguistics.

Sai Muralidhar Jayanthi, Danish Pruthi, and Graham Neubig. 2020. Neuspell: A neural spelling correction toolkit. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 158–164. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Shun Kiyono, Jun Suzuki, Masato Mita, Tomoya Mizumoto, and Kentaro Inui. 2019. An empirical study of incorporating pseudo data into grammatical error correction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 1236–1242. Association for Computational Linguistics.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem N. Chernodub, and Oleksandr Skurzhanskyi. 2020. Gector - grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications, BEA@ACL 2020, Online, July 10, 2020*, pages 163–170. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

Joana Ribeiro, Shashi Narayan, Shay B. Cohen, and Xavier Carreras. 2018. Local string transduction as sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 1360–1371. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng. 2016. Neural language correction with character-based attention. *CoRR*, abs/1603.09727.

Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. 2019. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 156–165. Association for Computational Linguistics.