SoftCorrect: Error Correction with Soft Detection for Automatic Speech Recognition

Yichong Leng\textsuperscript{1*}, Xu Tan\textsuperscript{2}, Wenjie Liu\textsuperscript{3}, Kaitao Song\textsuperscript{2}, Rui Wang\textsuperscript{2}
Xiang-Yang Li\textsuperscript{1}, Tao Qin\textsuperscript{2}, Ed Lin\textsuperscript{3}, Tie-Yan Liu\textsuperscript{2}
\textsuperscript{1}University of Science and Technology of China
\textsuperscript{2}Microsoft Research Asia
\textsuperscript{3}Microsoft Azure Speech
\textsuperscript{1}lyc123go@mail.ustc.edu.cn, xiangyangli@ustc.edu.cn
\textsuperscript{2}\{xuta,kaitaosong,ruiwa,taoqin,tyliu\}@microsoft.com \textsuperscript{3}\{liwenjie,edlin\}@microsoft.com

Abstract

Error correction in automatic speech recognition (ASR) aims to correct those incorrect words in sentences generated by ASR models. Since recent ASR models usually have low word error rate (WER), to avoid affecting originally correct tokens, error correction models should only modify incorrect words, and therefore detecting incorrect words is important for error correction. Previous works on error correction either implicitly detect error words through target-source attention or CTC (connectionist temporal classification) loss, or explicitly locate specific deletion/substitution/insertion errors. However, implicit error detection does not provide clear signal about which tokens are incorrect and explicit error detection suffers from low detection accuracy. In this paper, we propose SoftCorrect with a soft error detection mechanism to avoid the limitations of both explicit and implicit error detection. Specifically, we first detect whether a token is correct or not through a probability produced by a dedicatedly designed language model, and then design a constrained CTC loss that only duplicates the detected incorrect tokens to let the decoder focus on the correction of error tokens. Compared with implicit error detection with CTC loss, SoftCorrect provides explicit signal about which words are incorrect and thus does not need to duplicate every token but only incorrect tokens; compared with explicit error detection, SoftCorrect does not detect specific deletion/substitution/insertion errors but just leaves it to CTC loss. Experiments on AISHELL-1 and Aidatatang datasets show that SoftCorrect achieves 26.1\% and 9.4\% CER reduction respectively, outperforming previous works by a large margin, while still enjoying fast speed of parallel generation.

Introduction

Correction (Cucu et al. 2013; D’Haro and Banchs 2016; Anantaram et al. 2018; Du et al. 2022) has been widely used in automatic speech recognition (ASR) to refine the output sentences of ASR systems to reduce word error rate (WER). Considering the error rate of the sentences generated by ASR is usually low (e.g., <10\%, which means only a small proportion of tokens are incorrect and need correction), how to accurately detect errors is important for correction (Leng et al. 2021b). Otherwise, correct tokens may be changed by mistake, or error tokens cannot be corrected. Previous works conduct error detection in different ways: 1) Implicit error detection, where the errors are not explicitly detected but embedded in the correction process. For example, Liao et al. (2020); Mani et al. (2020); Wang et al. (2020); Zhu et al. (2021) adopt an encoder-decoder based autoregressive correction model with a target-source (decoder-encoder) attention (Vaswani et al. 2017); Gu and Kong (2021) duplicate the source tokens several times and leverage a CTC (connectionist temporal classification) loss (Graves et al. 2006), where the target-source alignments learnt in decoder-encoder attention or CTC paths play a role of implicit error detection. 2) Explicit error detection, where the specific deletion/substitution/insertion errors are detected out explicitly. For example, Leng et al. (2021b,a); Du et al. (2022); Shen et al. (2022) rely on the predicted duration to determine how many target tokens each source token should be corrected to (e.g., 0 stands for deletion error, 1 stands for no change or substitution error, \geq 2 stands for insertion error).

Implicit error detection enjoys the advantage of the flexibility of model learning but suffers from the limitation that it does not provide clear signal for model training about which tokens are incorrect. In the contrast, explicit error detection enjoys the advantage of clear signal but suffers from the limitation that it requires precise error patterns and thus new error will be introduced once error detection is not accurate. For example, if a substitution error is predicted as an insertion error by explicit error detection, then the model cannot correct this error but introduce new error by inserting a wrong token. A natural question arises: can we design a better error detection mechanism that inherits the advantages of both implicit and explicit error detection and avoids their limitations?

To answer this question, in this paper, we propose a soft error detection mechanism with an error detector (encoder) and an error corrector (decoder). Specifically, we dedicately design a language model as the encoder to determine whether a token is correct or not and design a constrained CTC loss on the decoder to only focus on correcting the de-
tected error tokens (focused error detection):

- Instead of predicting correction operations such as deletion, substitution and insertion (Leng et al. 2021b,a; Du et al. 2022), we only detect whether a token is correct or not. To this end, we can either use a binary classification (Fang et al. 2022) or the probability from language model for error detection. We choose the latter one since the probability from a language model contains more knowledge in language understanding and vocabulary space (Hinton et al. 2015; Gou et al. 2021) than a simple binary classification. However, previous methods for language modeling such as left-to-right language modeling (e.g., GPT (Brown et al. 2020)) and bidirectional language modeling (e.g., BERT (Devlin et al. 2019)) are not suitable in our scenario: 1) left-to-right language models like GPT only leverage unidirectional context information (e.g., left), which cannot provide enough context information for accurate probability estimation; 2) bidirectional language models like BERT can leverage bidirectional context information, but it needs N passes (where N corresponds to the number of tokens) (Salazar et al. 2020) to estimate the probability of all the N tokens in a sentence, which cannot satisfy the fast speed requirement for error detection. In this paper, we train the encoder with a novel language model loss, to output probabilities effectively and efficiently to detect error tokens in source sentence.

- Instead of duplicating all the source tokens multiple times in CTC loss, we only duplicate the incorrect tokens detected (as indicated by the probabilities from the encoder trained with our novel language model loss) and use a constrained CTC loss to let the decoder focus on the correction of these duplicated error tokens, resulting in a focused error correction. Compared with the standard CTC loss that duplicates all the tokens, our constrained CTC loss provides clear signals about which part of tokens should be corrected.

Furthermore, previous works (Weng et al. 2020; Liu et al. 2018) have shown that the multiple candidates generated by ASR beam search can be leveraged to verify the correctness of tokens (Leng et al. 2021a) in each candidate. To further improve correction accuracy, we take multiple candidates from ASR beam search as encoder input. Accordingly, the error detection in the encoder contains two steps, i.e., first selecting a better candidate from multiple candidates (equivalent to detect which candidates are likely to be incorrect) for further correction, and then detecting which tokens are likely to be incorrect in the selected candidate. The contributions of this paper are summarized as follows:

- We propose SoftCorrect with a soft error detection mechanism for ASR error correction to inherit the advantages of both explicit and implicit error detection and avoid their limitations.

- We design a novel language model loss for encoder to enable error detection and a constrained CTC loss for the decoder to focus on the tokens that are detected as errors.

- Experimental results on AISHELL-1 and Aidatatang datasets demonstrate that SoftCorrect achieves 26.1% and 9.4% CER reduction respectively, while still enjoying fast error correction with parallel generation.

**Background**

**Error Correction for ASR**

**Error Correction Models** Error correction is widely used in ASR systems (Shivakumar et al. 2018; Hu et al. 2020) to reduce word error rate. Error correction models usually take the sentences outputted by ASR systems as input and generate corrected sentences, and have evolved from early statistic machine translation models (Cucu et al. 2013; D’Haro and Banchs 2016), to later neural-network based autoregressive models (Tanaka et al. 2018; Liao et al. 2020; Wang et al. 2020), and to recent non-autoregressive models (Leng et al. 2021b,a; Du et al. 2022). Non-autoregressive error correction models generate sentences in parallel with the help of a duration predictor (Gu et al. 2018) to predict the number of tokens that each input token can be corrected to, which achieve much faster inference speed than autoregressive counterparts and approximate correction accuracy, making it suitable for online deployment.

**Multiple Candidates** Recent works (Zhu et al. 2021; Liu et al. 2018; Imamura and Sumita 2017; Weng et al. 2020) show that the multiple candidates generated by ASR beam search can have voting effect (Leng et al. 2021a), which can be beneficial for both autoregressive and non-autoregressive correction models. They first align the multiple candidates to the same length using multi-candidate alignment algorithm based on the token-level similarity and phoneme-level similarity, and take the aligned candidates as encoder input. SoftCorrect also leverages multiple candidates since the difference of beam search results can show the uncertainty of ASR model and give clues about potential error tokens.

**Error Detection by Target-Source Alignments**

Error detection can be achieved via the alignments between the target (correct) sentence and the source (incorrect) sentence, which can be either explicit or implicit.

**Explicit Alignment** By explicitly aligning the source and target sequences together with edit distance (Leng et al. 2021b), we can obtain the number of target tokens (duration) aligned with each source token and train a duration predictor. Thus, we can detect insertion, deletion and substitution error with corresponding duration (e.g., 0 stands for deletion error, 1 stands for no change or substitution error, ≥ 2 stands for insertion error). However, the duration predictor is hard to optimize precisely and thus new error will be introduced once duration prediction is not accurate.

**Implicit Alignment** Errors can be “detected” via implicit alignment between target and source sequences. For example, Transformer (Vaswani et al. 2017) based autoregressive models embed the target-source alignment in decoder-encoder attention (Wang et al. 2020; Zhu et al. 2021), and CTC-based models (Libovický and Helcl 2018; Saharia et al. 2020; Majumdar et al. 2021) leverage a CTC loss (Graves et al. 2006) to align target with duplicated
source implicitly. A desirable property of CTC loss is that it enables parallel error correction, without the need of duration prediction and with more flexibility during correction. Thus, in this paper, we adopt the CTC based solution but enhance it with a soft error detection mechanism.

**SoftCorrect**

**System Overview**

As shown in Figure 1, SoftCorrect consists of an error detector (the encoder) and a focused error corrector (the decoder). We introduce the whole system step by step:

- Motivated by the voting effect in multiple candidates for error detection, we leverage multiple candidates from ASR beam search. We first align these candidates to the same length following Leng et al. (2021a). The aligned candidates are shown in the bottom left of Figure 1. The aligned candidates are converted into token embeddings, concatenated along the position and fed into a linear layer.
- The detector is a standard Transformer Encoder (Vaswani et al. 2017) which takes the output of the previous step as input, and generates a probability for each token in each candidate. Specifically, the output hidden of the encoder is multiplied with a token-embedding matrix to generate a probability distribution over the whole vocabulary. For example, the output probability distribution in the last position in Figure 1 can provide the probability for token $E$ and $E'$ simultaneously. Since ASR usually has low WER, to prevent encoder from learning trivial copy, we propose an anti-copy language model loss to train the encoder to output this probability distribution.
- Based on the probability, we can choose the token with the highest probability in each position from multiple candidates and obtain a better candidate, which usually contains less errors and thus makes the error detection easier. This step is illustrated as the “Candidate Selection” module in Figure 1 and the selected candidate is $AB''CD'E$. Noted that we conduct position-wise selection and the tokens in selected candidate (e.g., $B''$ and $E$) can come from different candidates.
- After candidate selection, we combine the probability of each token in selected candidate with its corresponding probability from the ASR model. The error detection score for each token is the weighted linear combination of encoder probability and ASR output probability reflecting the similarity between token pronunciation and audio. A token is detected as incorrect when the combined probability is lower than a threshold (Huang and Peng 2019).
- The corrector (decoder) takes the generated candidate as input and outputs refined tokens. It is trained with a constrained CTC loss, which learns to only modify the detected “incorrect” tokens while directly copying remaining tokens to output. Therefore, we only duplicate the incorrect tokens detected in previous step. As shown in the bottom right of Figure 1, the detected and duplicated error tokens are $B''$ and $D'$.

In the next subsections, we introduce the details of the anti-copy language model loss in encoder to generate token probability for error detection, and the constrained CTC loss in decoder for focused error correction.

**Anti-Copy Language Modeling for Detection**

We use the encoder to output a probability for each token in the multiple candidates, where this probability can be used in candidate selection and error detection. Since we need to select better candidate as well as detect errors, the probability from binary classification based error detection method (Omelianchuk et al. 2020) is unsuitable for the lack of knowledge in language understanding and vocabulary space (Hinton et al. 2015; Gou et al. 2021). Indeed,
the probability we need is like a kind of language modeling, which determines whether a token is natural/reasonable given the context information. Common language modeling methods are unsuitable for probability estimation since GPT-based methods lack bidirectional context information and BERT-based methods are too slow with N-pass (Salazar et al. 2020) inference. To alleviate these issues, we propose a novel method to not only leverage bidirectional information but also provide fast probability estimation.

A straightforward way is to train the Transformer encoder to predict ground-truth (correct) tokens in each position given multiple aligned candidates as input. In this way, the encoder can learn to output probability to determine whether a token is natural/reasonable given the context information. However, since the ASR systems usually have relatively low WER (e.g., <10%), a large proportion of tokens in aligned input are consistent (i.e., with the same token) and correct. In this way, directly predicting the ground-truth token in each position would result in trivial copy. For example, the first position in Figure 1 might be a trivial copy since the aligned input tokens are consistent (i.e., all A) and the corresponding target token is also A. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.

To alleviate this problem, we propose an anti-copy language model loss to prevent learning copy only, which modifies the standard cross-entropy loss by changing its prediction vocabulary and adding a regularization term. Specifically, we add a special GT token (it does not stand for any token, and GT stands for ground-truth), and the corresponding target token is also GT. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.

To alleviate this problem, we propose an anti-copy language model loss to prevent learning copy only, which modifies the standard cross-entropy loss by changing its prediction vocabulary and adding a regularization term. Specifically, we add a special GT token (it does not stand for any token, and GT stands for ground-truth), and the corresponding target token is also GT. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.

To alleviate this problem, we propose an anti-copy language model loss to prevent learning copy only, which modifies the standard cross-entropy loss by changing its prediction vocabulary and adding a regularization term. Specifically, we add a special GT token (it does not stand for any token, and GT stands for ground-truth), and the corresponding target token is also GT. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.

To alleviate this problem, we propose an anti-copy language model loss to prevent learning copy only, which modifies the standard cross-entropy loss by changing its prediction vocabulary and adding a regularization term. Specifically, we add a special GT token (it does not stand for any token, and GT stands for ground-truth), and the corresponding target token is also GT. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.

To alleviate this problem, we propose an anti-copy language model loss to prevent learning copy only, which modifies the standard cross-entropy loss by changing its prediction vocabulary and adding a regularization term. Specifically, we add a special GT token (it does not stand for any token, and GT stands for ground-truth), and the corresponding target token is also GT. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.

To alleviate this problem, we propose an anti-copy language model loss to prevent learning copy only, which modifies the standard cross-entropy loss by changing its prediction vocabulary and adding a regularization term. Specifically, we add a special GT token (it does not stand for any token, and GT stands for ground-truth), and the corresponding target token is also GT. This trivial copy issue will cause the model outputting an extreme high probability for the input token on consistent position and thus hurt the ability of encoder on detecting errors on consistent position.
where \( x \) is the decoder input (i.e., the selected candidate with expansion on the “incorrect tokens”), just as “decoder input” shown in Figure 3). \( \phi'(y) \) represents all the possible alignment paths to generate \( y \) in our constrained CTC (different from the standard alignments \( \phi(y) \)), and \( z \) represents one possible CTC path, as shown in the red arrow in Figure 3.

The main difference between our loss and standard CTC loss is the all possible alignment paths \( \phi'(y) \). As shown in Figure 3, the possible CTC alignment paths \( \phi'(y) \) lay only on the shaded (green) nodes, in contrast to the standard CTC where the alignment paths \( \phi(y) \) lay on all nodes. As a result, the “correct token” is used as an anchor and cannot be dynamically aligned (the output of anchor token must be the anchor token itself). During the inference, we perform softmax over the possible error tokens, select the best token from each of those positions and then remove duplicates and blank. With the help of the explicit error detection from encoder, we can skip the correction process of the decoder to reduce the system latency if all input tokens are detected as correct token.

Explicit error detector can possibly produce new errors (e.g., some correct tokens are identified as errors) and propagate them to the error corrector. Hence, we need the error corrector to be more robust to the outputs of error detector. Specifically, when training corrector, we randomly select 5% correct tokens and regard them as pseudo error tokens to simulate the mistakes from detector, so that model will not modify these correct tokens during the optimization and can be more robust to the outputs of detector.

**Experimental Setup**

In this section, we introduce the datasets and the ASR model used for correction, and some previous error correction systems for comparison.

**Datasets and ASR Model**

We conduct experiments on two Mandarin ASR datasets, AISHELL-1 (Bu et al. 2017) and Aidatatang.200zh, referred to as Aidatatang for short. AISHELL-1 contains 150/10/5-hour speech data for train/development/test, while Aidatatang contains 140/20/40-hour speech data for train/development/test, respectively.

The ASR model used in our experiments is a state-of-the-art model with Conformer architecture (Gulati et al. 2020), enhanced with SpecAugment (Park et al. 2019) and speed perturbation for data augmentation, and a language model for joint decoding. The hyper-parameters of this ASR model follow the ESPnet codebase (Watanabe et al. 2018)\(^1\).\(^2\)

The training, development, and test data for correction models are obtained by using the ASR model to transcribe the corresponding datasets in AISHELL-1 and Aidatatang. Following the common practice in ASR correction (Leng et al. 2021b,a; Du et al. 2022; Zhu et al. 2021), we use 400M unpaired text data to construct a pseudo pretraining dataset for both SoftCorrect and baseline systems.

**Baseline Systems**

We compare SoftCorrect with the several error correction baselines, including systems using implicit and explicit error detection.

For baselines with implicit error detection, we use: 1) **AR Correct**. A standard autoregressive (AR) encoder-decoder model based on Transformer (Vaswani et al. 2017). 2) **AR N-Best**. Following Zhu et al. (2021), we train an AR Transformer model by taking the aligned multiple candidates from ASR beam search as input and generating correction result.

For baselines with explicit error detection, we use: 1) **FastCorrect**. FastCorrect (Leng et al. 2021b) is a non-autoregressive model for ASR correction, which utilizes token duration to adjust input sentence length to enable parallel decoding. 2) **FastCorrect 2**. Leng et al. (2021a) introduce multiple candidates into non-autoregressive FastCorrect model and achieve state-of-the-art correction accuracy.

Considering SoftCorrect leverages multiple candidates, we also take rescoring method for comparison. The rescoring model is a 12-layer Transformer decoder model and the details of rescoring follow Huang and Peng (2019). Besides, we also combine non-autoregressive correction with rescoring together to construct another two baselines. One is **FC + Rescore** where the outputs of ASR are first corrected by FastCorrect (FC for short) and then rescored, the other is **Rescore + FC** where the ASR outputs are first rescored and then corrected.

**Results**

In this section, we first compare the character error rate reduction (CERR) and latency of SoftCorrect with baselines, and then conduct ablation studies to verify the effectiveness of several designs in SoftCorrect, including anti-copy language model loss and constrained CTC loss. Besides, we conduct some analyses to show the advantages of SoftCorrect on the ability of error detection and error correction over baseline systems.

---

1. [github.com/espnet/espnet/tree/master/egs/aishell](https://github.com/espnet/espnet/tree/master/egs/aishell)
2. [github.com/espnet/espnet/tree/master/egs/aidatatang.200zh](https://github.com/espnet/espnet/tree/master/egs/aidatatang.200zh)
Ablation Studies

We conduct ablation studies to verify the effectiveness of our soft detection with encoder for error detection and decoder for focused correction, as shown in Table 2. We introduce these studies as follows:

- We first remove our anti-copy language model loss and simply use standard cross-entropy loss to train the encoder (Setting ID 2), resulting in an inferior accuracy due to the model may learn to copy the ground-truth token, hurting the error detection ability.
- Since there are some popular methods to model token-level probability such as BERT (Devlin et al. 2019) and GPT (Brown et al. 2020), we apply BERT-style and GPT-style training loss on the encoder to detect errors in the aligned multiple candidates, as shown in ID 3 and 4. Moreover, we also train the encoder to perform detection on each token of each candidate with a binary classification loss (ID 5). The poor results of GPT-style training shows that bi-directional information is necessary for error-detection. The BERT-style training or binary classification achieves lower accuracy than SoftCorrect, showing the effectiveness of our anti-copy language model loss.
- When removing the constraint on the CTC loss (ID 6), the correction accuracy is lower, which demonstrates the advantage of only focusing on correcting the detected error tokens. The results also show that the error detector is reliable because the attempt on modifying tokens that are detected to be right (Setting ID 6) only leads to worse accuracy.

Method Analyses

We compare the error detection and correction ability of SoftCorrect with previous autoregressive and non-autoregressive baselines. We measure the error detection ability using the precision ($P_{\text{det}}$) and recall ($R_{\text{det}}$) of that an error token is detected as error, and measure the error correction ability using the precision ($P_{\text{cor}}$) of that an error
Table 2: Ablation studies on the designs in SoftCorrect, including some variants of the anti-copy language model loss for encoder and the constrained CTC loss for decoder. “CE loss” means using standard cross-entropy loss to predict ground truth, “BERT-style” refers to using BERT model to estimate the probability of each token via N-pass (Salazar et al. 2020). “GPT-style” refers to using left-to-right language model to estimate the probability of each token and “Binary” refers to using binary classification to detect errors.

<table>
<thead>
<tr>
<th>Model</th>
<th>AISHHELL-1</th>
<th>Aidatatang</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CER ↓</td>
<td>CERR ↑</td>
</tr>
<tr>
<td>SoftCorrect</td>
<td>3.57</td>
<td>26.09</td>
</tr>
<tr>
<td>Encoder CE</td>
<td>3.77</td>
<td>21.94</td>
</tr>
<tr>
<td>BERT-style</td>
<td>3.93</td>
<td>18.63</td>
</tr>
<tr>
<td>GPT-style</td>
<td>2.76</td>
<td>1.45</td>
</tr>
<tr>
<td>Binary</td>
<td>3.98</td>
<td>17.60</td>
</tr>
<tr>
<td>Decoder</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>No Correction</td>
<td>4.83</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Comparison of different systems in terms of error detection and correction ability. $P_{\text{det}}$, $R_{\text{det}}$, and $F_{\text{1det}}$ represent the precision, recall, and F1 score of error detection. $P_{\text{cor}}$ represents the precision of correction on error tokens. The character error rate reduction (CERR) is also shown.

<table>
<thead>
<tr>
<th>Model</th>
<th>AISHHELL-1</th>
<th>Aidatatang</th>
<th>CERR</th>
<th>CERR</th>
<th>CERR</th>
<th>CERR</th>
<th>CERR</th>
<th>CERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit error detection baselines</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR Correct</td>
<td>84.56</td>
<td>33.00</td>
<td>54.73</td>
<td>64.73</td>
<td>15.73</td>
<td>73.55</td>
<td>14.32</td>
<td>29.78</td>
</tr>
<tr>
<td>AR N-Best</td>
<td>76.03</td>
<td>45.13</td>
<td>54.96</td>
<td>72.29</td>
<td>18.43</td>
<td>57.98</td>
<td>32.18</td>
<td>56.55</td>
</tr>
<tr>
<td>Explicit error detection baselines</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FastCorrect</td>
<td>83.72</td>
<td>34.54</td>
<td>50.10</td>
<td>59.84</td>
<td>13.87</td>
<td>69.78</td>
<td>9.78</td>
<td>29.78</td>
</tr>
<tr>
<td>FastCorrect 2</td>
<td>80.38</td>
<td>32.54</td>
<td>56.51</td>
<td>70.13</td>
<td>14.91</td>
<td>60.50</td>
<td>32.35</td>
<td>53.48</td>
</tr>
<tr>
<td>SoftCorrect</td>
<td>84.06</td>
<td>49.71</td>
<td>59.94</td>
<td>71.30</td>
<td>26.09</td>
<td>80.52</td>
<td>25.29</td>
<td>49.32</td>
</tr>
</tbody>
</table>

As shown in Table 3, we can observe that: 1) on both datasets, SoftCorrect achieves better $P_{\text{det}}$, $R_{\text{det}}$, and $P_{\text{cor}}$ than non-autoregressive baselines with explicit error detection, which shows SoftCorrect has a stronger ability on error detection and correction; 2) Compared with autoregressive baselines using implicit error detection, SoftCorrect performs better on balancing the precision and recall of error detection (higher $F_{\text{1det}}$), which verifies the necessity of soft error detection. 3) The errors in Aidatatang dataset is hard to detect, which cannot be handled with implicit error detection or duration-based explicit error detection. On this dataset, previous method may mistake a correct token which introduces new error, or miss an incorrect token. In contrast, the precision or recall of the detection in SoftCorrect is higher, demonstrating the advantage of soft error detection. Moreover, with high-accurate error detection, constrained CTC loss makes the error correction more focused and thus easier, resulting in the higher $P_{\text{cor}}$ of SoftCorrect.

Conclusion

In this paper, we design a soft error detection mechanism for ASR error correction, which consists of an encoder for error detection and a decoder for focused error correction. Considering error detection is important for ASR error correction and previous works using either explicit or implicit error detection suffer from some limitations, we propose SoftCorrect with a soft error detection mechanism. Specifically, we design an anti-copy language model loss to enable the encoder to select a better candidate from multiple input candidates and detect errors in the selected candidate, and design a constrained CTC loss to help decoder focus on correcting detected error tokens while keeping undetected tokens unchanged. Experimental results show that SoftCorrect achieves much larger CER reduction compared with previous explicit and implicit error detection methods in ASR error correction, while still enjoying fast inference speed.
References


data augmentation method for automatic speech recognition. In INTERSPEECH.


