Chinese Spelling Check Paper-reading

王曦

Problem Formulation

- Given a Chinese sentence $X = \{x_1, x_2, ..., x_n\}$ of length n
- detect spelling errors on character level and output its correct corresponding sentence Y= $\{y_1, y_2, ..., y_n\}$
- detect misspelled characters and generate candidates via a language model, and then use a phonetic model or rules to filter wrong candidates

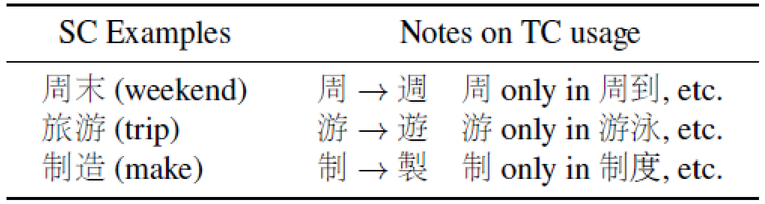
FASPell: A Fast, Adaptable, Simple, Powerful Chinese Spell Checker Based On DAE-Decoder Paradigm

Yuzhong Hong, Xianguo Yu, Neng He, Nan Liu, Junhui Liu
Intelligent Platform Division, iQIYI, Inc.
{hongyuzhong, yuxianguo, heneng, liunan, liujunhui}@qiyi.com

Introduction

- The spell checking methods for languages like English can hardly be directly used for the Chinese language because there are no delimiters between words, whose lack of morphological variations makes the syntactic and semantic interpretations of any Chinese character highly dependent on its context.

- Almost all previous Chinese spell checking models deploy a common paradigm where a fixed set of similar characters of each Chinese character(called confusion set) is used as candidates, and a filter selects the best candidates as substitutions for a given sentence. This naive design is subjected to two major bottlenecks, whose negative impact has been unsuccessfully mitigated:
  - overfitting to under-resourced Chinese spell checking data.
  - inflexibility and insufficiency of confusion set in utilizing character similarity.



SC Examples	Notes on TC usage	
周末 (weekend)	周 → 週	周 only in 周到, etc.
旅游 (trip)	游 → 遊	游 only in 游泳, etc.
制造 (make)	制 → 製	制 only in 制度, etc.

Motivation and contributions

The motivation of this paper is to circumvent the two bottlenecks by changing the paradigm for Chinese spell checking.

As a major contribution and as exemplified by our proposed Chinese spell checking model in Figure 1, the most general form of the new paradigm consists of a denoising autoencoder (DAE) and a decoder. To prove that it is indeed a novel contribution, we compare it with two similar paradigms and show their differences as follows:

- Similar to the old paradigm used in previous Chinese spell checking models, a model under the DAE-decoder paradigm also produces candidates (by DAE) and then filters the candidates (by the decoder). However, candidates are produced on the fly based on contexts.
- In the encoder-decoder paradigm, the encoder extracts semantic information, and the decoder generates texts that embody the information. In contrast, in the DAE-decoder paradigm, the DAE provides candidates to reconstruct texts from the corrupted ones based on contextual feature, and the decoder selects the best candidates by incorporating other features.

Model Architecture

Masked language model (MLM) guesses the tokens that are masked in a tokenized sentence. For simplicity purposes, FASPell adopts the architecture of MLM as in BERT.
However, just using a pre-trained MLM raises the issue that the errors introduced by random masks may be very different from the actual errors in spell checking data. Therefore, we propose the following method to fine-tune the MLM on spell checking training sets:

- For texts that have no errors, we follow the original training process as in BERT
- For texts that have errors, we create two types of training examples by:
    1. given a sentence, we mask the erroneous tokens with themselves and set their target labels as their corresponding correct characters.
    2. to prevent overfitting, we also mask tokens that are not erroneous with themselves and set their target labels as themselves, too.

Model Architecture

Character similarity

Erroneous characters in Chinese texts by humans are usually either visually or phonologically similar to corresponding correct characters, or both. It is also true that erroneous characters produced by OCR possess visual similarity.

We base our similarity computation on two open databases: *Kanji Database Project* and *Unihan Database* because they provide shape and pronunciation representations for all CJK Unified Ideographs in all CJK languages.

Model Architecture

Visual similarity

The Kanji Database Project uses the Unicode standard – Ideographic Description Sequence(IDS) to represent the shape of a character.

In our model, we only adopt the string-form IDS. We define the visual similarity between two characters as one minus normalized Levenshtein edit distance between their IDS representations.

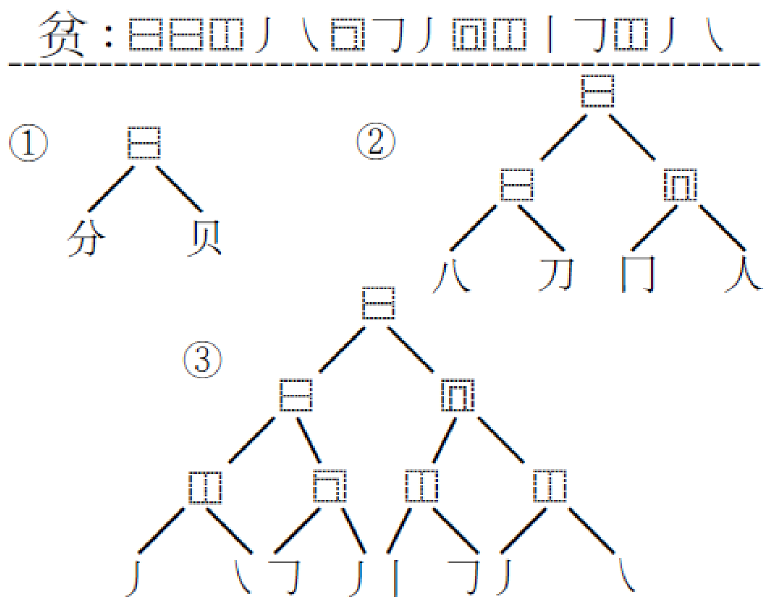
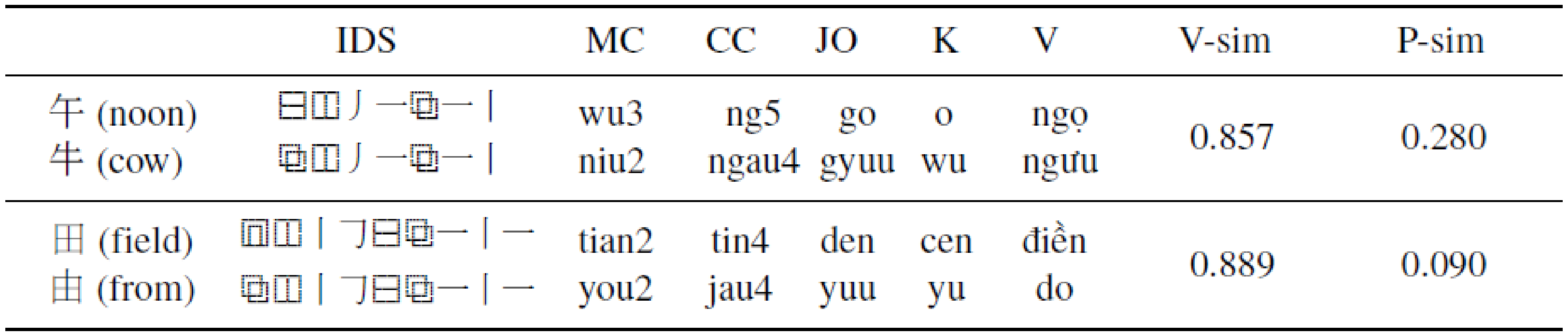


Figure 2: The IDS of a character can be given in different granularity levels as shown in the tree forms in ①-③ for the simplified character 贫 (meaning *poor*). In FASPell, we only use stroke-level IDS in the form of a string, like the one above the dashed ruling line. Unlike using only actual strokes (Wang et al., 2018), the Unicode standard Ideographic Description Characters (e.g., the non-leaf nodes in the trees) describe the layout of a character. They help us to model the subtle nuances in different characters that are composed of identical strokes (see examples in Table 2). Therefore, IDS gives us a more precise shape representation of a character.

Model Architecture

Phonological similarity
Different Chinese characters sharing identical pronunciation is very common, which is the case for any CJK language. Thus, if we were to use character pronunciations in only one CJK language, the phonological similarity of character pairs would be limited to a few discrete values. However, a more continuous phonological similarity is preferred because it can make the curve used for filtering candidates smoother.
Therefore, we utilize character pronunciations of all CJK languages, which are provided by the Unihan Database. To compute the phonological similarity of two characters, we first calculate one minus normalized Levenshtein edit distance between their pronunciation representations in all CJK languages (if applicable). Then, we take the mean of the results. Hence, the similarity should range from 0 to 1.



	IDS	MC	CC	JO	K	V	V-sim	P-sim
午 (noon)	⿱⿰丿一⿻一丨	wu3	ng5	go	o	ngọ	0.857	0.280
牛 (cow)	⿻⿰丿一⿻一丨	niu2	ngau4	gyuu	wu	ngưu		
田 (field)	⿴⿰丨𠃌⿱⿻一丨一	tian2	tin4	den	cen	điền	0.889	0.090
由 (from)	⿻⿰丨𠃌⿱⿻一丨一	you2	jau4	yuu	yu	do		

Model Architecture

Confidence-Similarity Decoder

Candidate filters in many previous models are based on setting various thresholds and weights for multiple features of candidate characters. Instead of this naive approach, we propose a method that is empirically effective under the principle of getting the highest possible precision with minimal harm to recall. Since the decoder utilizes contextual confidence and character similarity, we refer to it as the confidence-similarity decoder (CSD).

Model Architecture

Confidence-Similarity Decoder

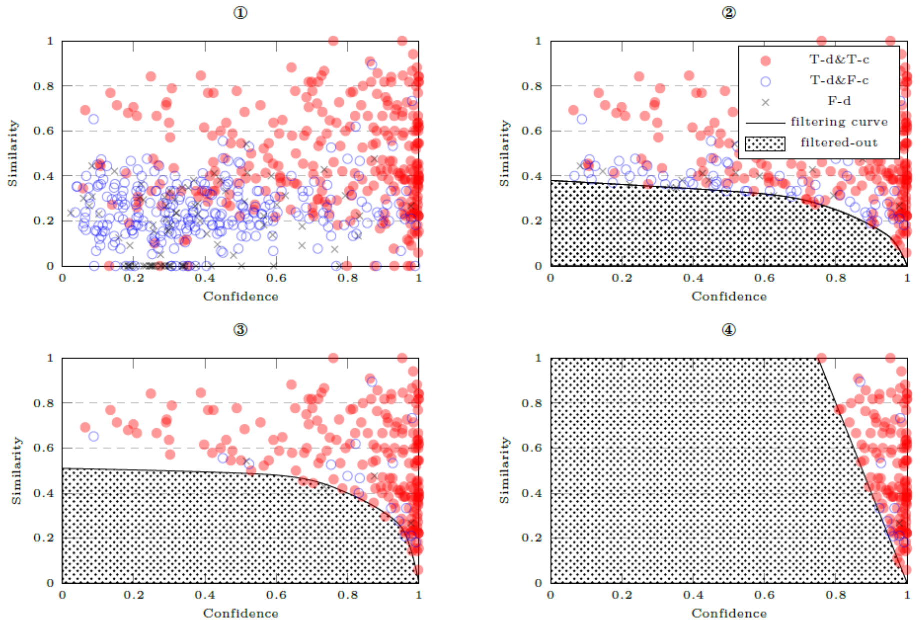
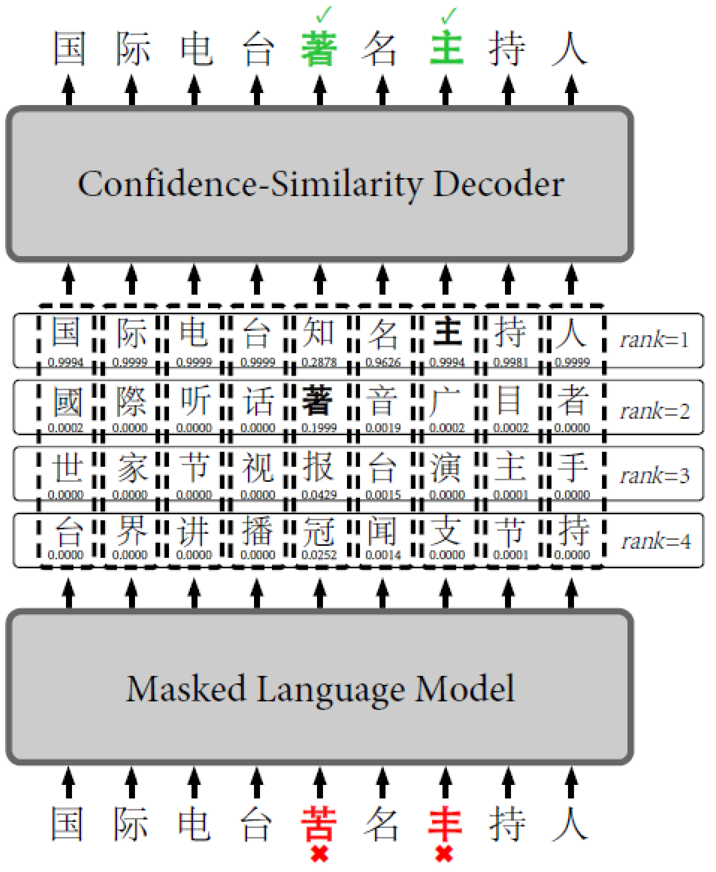


Figure 3: All four plots show the same confidence-similarity graph of candidates categorized by being true-detection-and-true-correction (T-d&T-c), true-detection-and-false-correction (T-d&F-c) and false-detection (F-d). But, each plot shows a different way of filtering candidates: in plot ①, no candidates are filtered; in plot ②, the filtering optimizes detection performance; in plot ③, as adopted in FASPell, the filtering optimizes correction performance; in plot ④, as adopted by previous models, candidates are filtered out by setting a threshold for weighted confidence and similarity ($0.8 \times confidence + 0.2 \times similarity < 0.8$ as an example in the plot). Note that the four plots use the actual first-rank candidates (using visual similarity) for our **OCR data** (Trn_{ocr}) except that we randomly sampled only 30% of the candidates to make the plots more viewable on paper.



Experiments

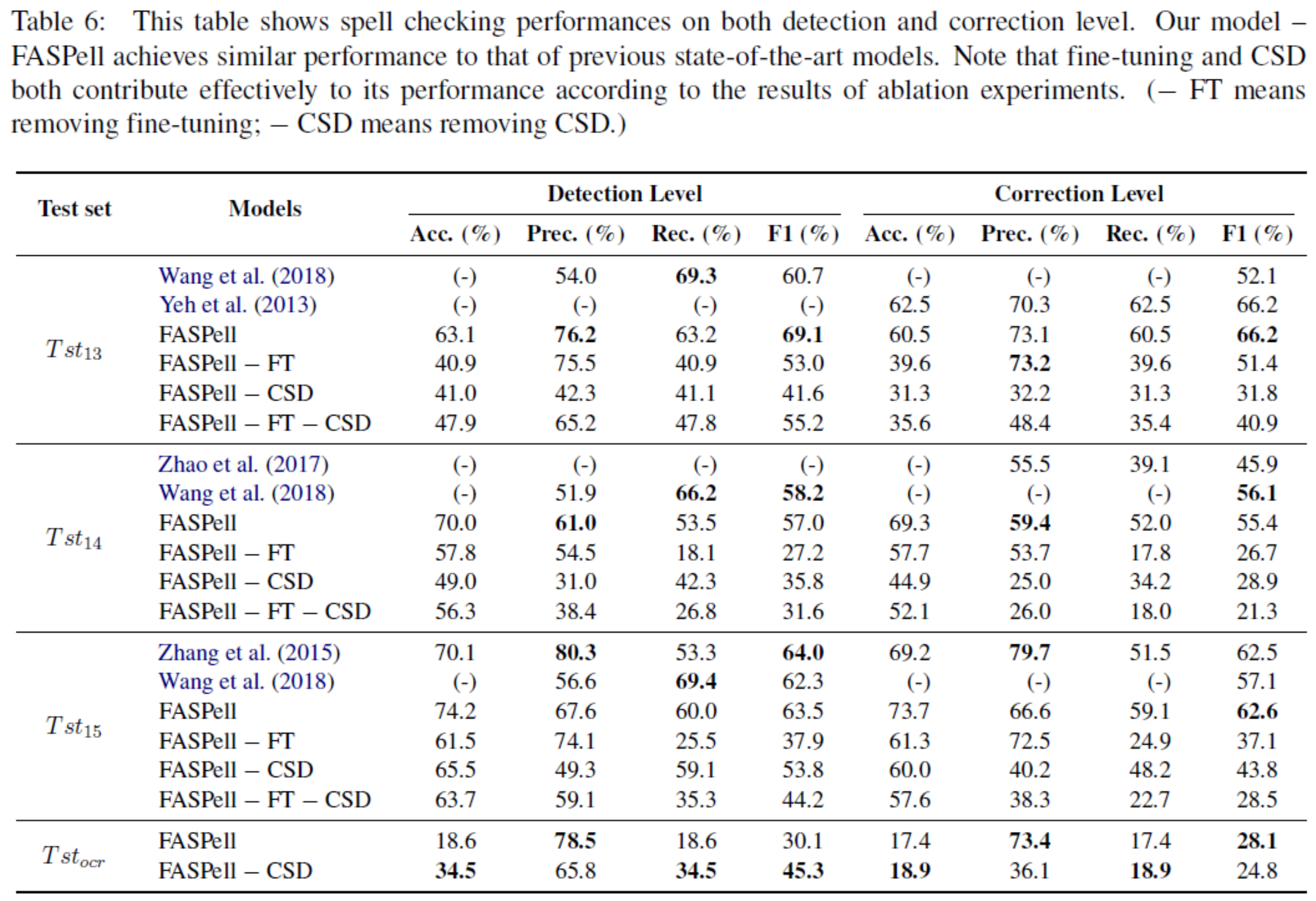


Table 6: This table shows spell checking performances on both detection and correction level. Our model – FASPell achieves similar performance to that of previous state-of-the-art models. Note that fine-tuning and CSD both contribute effectively to its performance according to the results of ablation experiments. (– FT means removing fine-tuning; – CSD means removing CSD.)

Test set	Models	Detection Level				Correction Level			
		Acc. (%)	**Prec. (%)**	**Rec. (%)**	**F1 (%)**	**Acc. (%)**	**Prec. (%)**	**Rec. (%)**	**F1 (%)**
Tst_{13}	Wang et al. (2018)	(-)	54.0	**69.3**	60.7	(-)	(-)	(-)	52.1
	Yeh et al. (2013)	(-)	(-)	(-)	(-)	62.5	70.3	62.5	66.2
	FASPell	63.1	**76.2**	63.2	**69.1**	60.5	73.1	60.5	**66.2**
	FASPell – FT	40.9	75.5	40.9	53.0	39.6	**73.2**	39.6	51.4
	FASPell – CSD	41.0	42.3	41.1	41.6	31.3	32.2	31.3	31.8
	FASPell – FT – CSD	47.9	65.2	47.8	55.2	35.6	48.4	35.4	40.9
Tst_{14}	Zhao et al. (2017)	(-)	(-)	(-)	(-)	(-)	55.5	39.1	45.9
	Wang et al. (2018)	(-)	51.9	**66.2**	**58.2**	(-)	(-)	(-)	**56.1**
	FASPell	70.0	**61.0**	53.5	57.0	69.3	**59.4**	52.0	55.4
	FASPell – FT	57.8	54.5	18.1	27.2	57.7	53.7	17.8	26.7
	FASPell – CSD	49.0	31.0	42.3	35.8	44.9	25.0	34.2	28.9
	FASPell – FT – CSD	56.3	38.4	26.8	31.6	52.1	26.0	18.0	21.3
Tst_{15}	Zhang et al. (2015)	70.1	**80.3**	53.3	**64.0**	69.2	**79.7**	51.5	62.5
	Wang et al. (2018)	(-)	56.6	**69.4**	62.3	(-)	(-)	(-)	57.1
	FASPell	74.2	67.6	60.0	63.5	73.7	66.6	59.1	**62.6**
	FASPell – FT	61.5	74.1	25.5	37.9	61.3	72.5	24.9	37.1
	FASPell – CSD	65.5	49.3	59.1	53.8	60.0	40.2	48.2	43.8
	FASPell – FT – CSD	63.7	59.1	35.3	44.2	57.6	38.3	22.7	28.5
Tst_{ocr}	FASPell	18.6	**78.5**	18.6	30.1	17.4	**73.4**	17.4	**28.1**
	FASPell – CSD	**34.5**	65.8	**34.5**	**45.3**	**18.9**	36.1	**18.9**	24.8

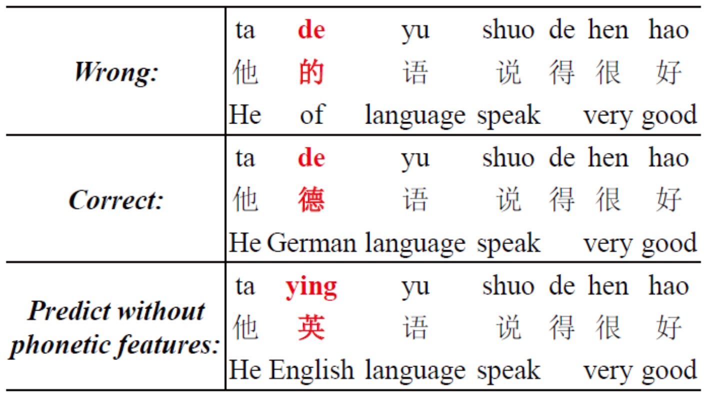
Correcting Chinese Spelling Errors with Phonetic Pre-training

Ruiqing Zhang, Chao Pang, Chuanqiang Zhang, Shuohuan Wang,
Zhongjun He,[*] Yu Sun, Hua Wu and Haifeng Wang
Baidu Inc. No. 10, Shangdi 10th Street, Beijing, 100085, China
{zhangruiqing01, pangchao04, zhangchuanqiang, wangshuohuan}@baidu.com
{hezhongjun, sunyu02, wu_hua, wanghaifeng}@baidu.com

Introduction

- Spelling errors are common in practice and the errors will be enlarged in the downstream tasks. Therefore, Spelling correction is important to many NLP applications such as search optimization, machine Translation, part-of-speech tagging, etc.

- Unlike alphabetic languages, Chinese characters cannot be typed without the help of input systems, such as Chinese Pinyin or automatic speech recognition (ASR). Thus typos of similarly pronounced characters occur quite often in Chinese text. According to Liu et al. (2010), 83% of Chinese spelling errors on the Internet results from phonologically similar characters.



Wrong:	ta	**de**	yu	shuo	de	hen	hao
	他	**的**	语	说	得	很	好
	He	of	language	speak		very	good
Correct:	ta	**de**	yu	shuo	de	hen	hao
	他	**德**	语	说	得	很	好
	He	German	language	speak		very	good
Predict without phonetic features:	ta	**ying**	yu	shuo	de	hen	hao
	他	**英**	语	说	得	很	好
	He	English	language	speak		very	good

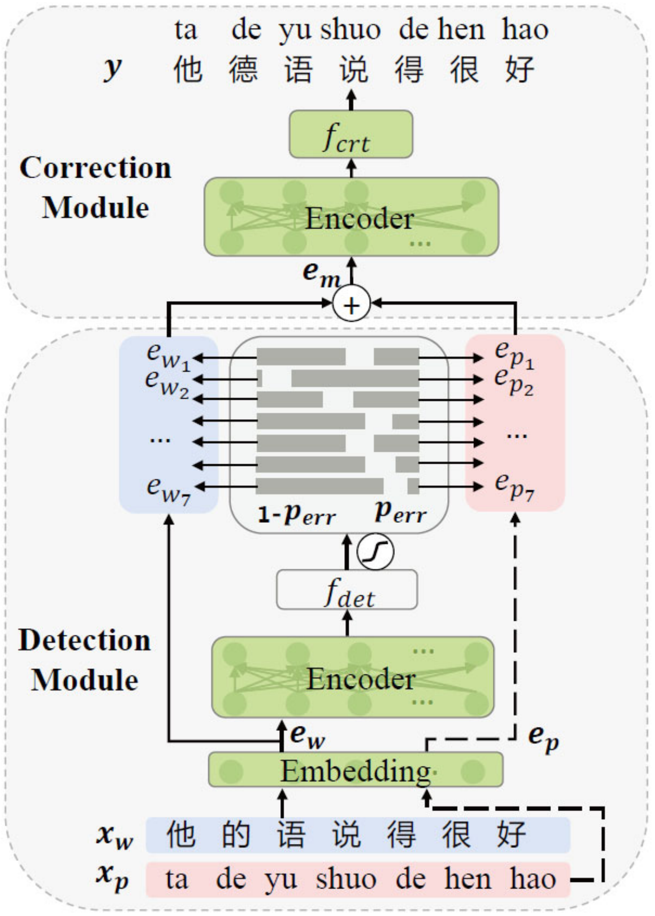
Related Work

- Some work proposed to optimize the error correction performance directly as a sequence-labeling task with:
  - conditional random fields (CRF) (Wu et al., 2018)
  - recurrent neural networks (RNN)(Zheng et al., 2016; Yang et al., 2017)
- Wang et al.(2019) used a sequence-to-sequence framework with copy mechanism to copy the correction results directly from a prepared confusion set for the erroneous words.
- Cheng et al. (2020) built a graph convolution network (GCN) on top of BERT (Devlin et al., 2019) and the graph was constructed from a confusion set.
- Zhang et al. (2020) proposed a soft-masked BERT model that first predicts the probability of spelling error for each word, and then uses the probabilities to perform a soft-masked word embedding for correction.

Methods

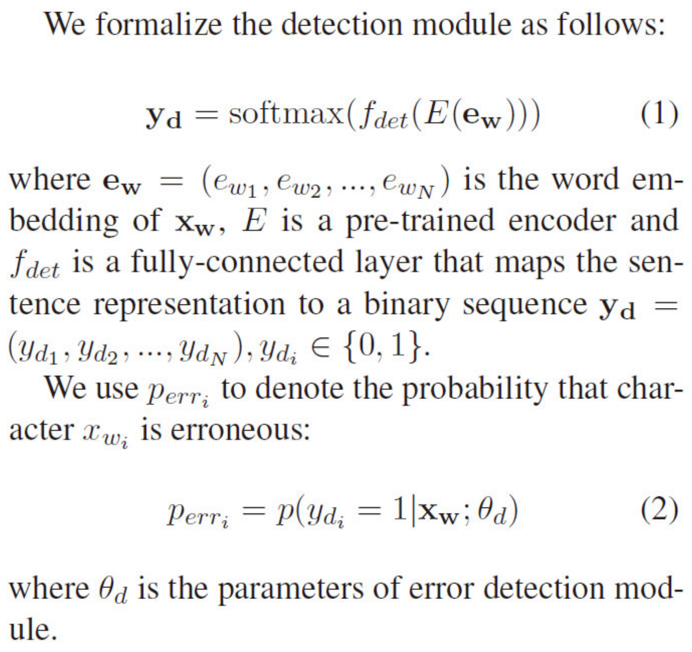
- We propose an end-to-end CSC model which consists of two components, detection and correction.
- The detection module takes x_w as input and predicts the probability of spelling error for each character.
- The correction model takes the combination of the embedding of x_w and its corresponding pinyin sequence x_p as input and predicts the correct sequence y. We propose a method to fuse x_w and x_p embeddings using the probability of spelling error as weights.
- Following the pre-train and fine-tune framework, we first pre-train a masked language model, MLM-phonetics, by learning to predict characters from similarly pronounced characters and pinyin. Then in fine-tuning, we jointly optimize the detection and correction modules.

Model Architecture



Detection Module

the goal of the detection module is to check whether a character is correct or not. For this labelling problem, we use class 1 and 0 to label misspelled characters and correct characters, respectively.



We formalize the detection module as follows:

$$\mathbf{y_d} = \mathrm{softmax}(f_{det}(E(\mathbf{e_w}))) \tag{1}$$

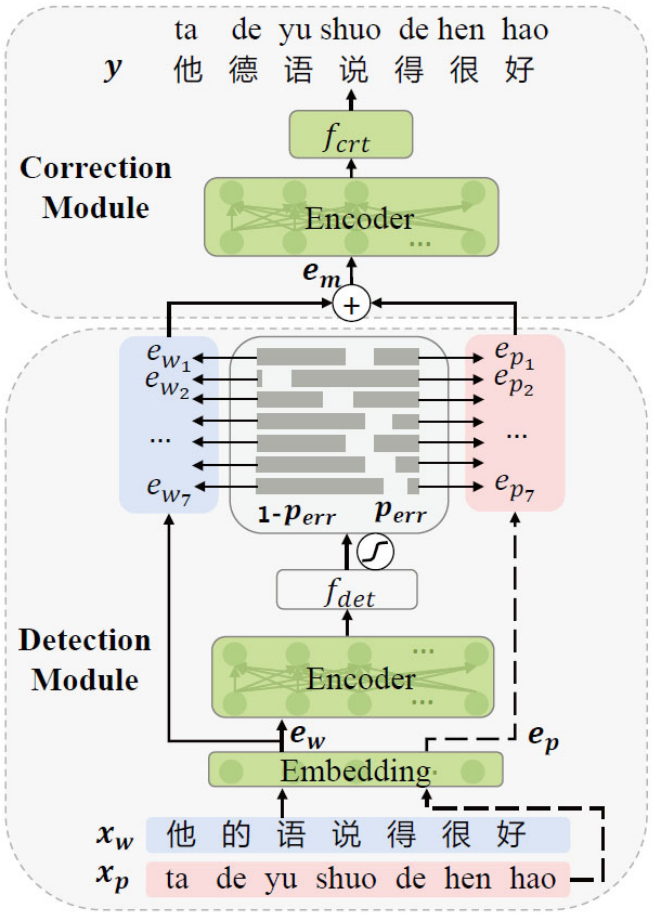
where $\mathbf{e_w} = (e_{w_1}, e_{w_2}, ..., e_{w_N})$ is the word embedding of $\mathbf{x_w}$, E is a pre-trained encoder and f_{det} is a fully-connected layer that maps the sentence representation to a binary sequence $\mathbf{y_d} = (y_{d_1}, y_{d_2}, ..., y_{d_N}), y_{d_i} \in \{0, 1\}$.

We use p_{err_i} to denote the probability that character x_{w_i} is erroneous:

$$p_{err_i} = p(y_{d_i} = 1 | \mathbf{x_w}; \theta_d) \tag{2}$$

where θ_d is the parameters of error detection module.

Model Architecture



Correction Module

The goal of the correction module is to generate correct characters based on the output of the detection module.

We not only use the word embeddings for input, but also use the pinyin embeddings to integrate the phonetic information. Concretely, we first generate the pinyin sequence $\mathbf{x_p}$ using the PyPinyin[3] tool, get the pinyin embedding $\mathbf{e_p}$ from the embedding layer, and fuse it with the word embedding $\mathbf{e_w}$ by linear combination:

$$\mathbf{e_m} = (1 - \mathbf{p_{err}}) \cdot \mathbf{e_w} + \mathbf{p_{err}} \cdot \mathbf{e_p} \qquad (3)$$

Finally, the correction results $\mathbf{y}$ is predicted through a fully-conntected layer f_{crt}:

$$\mathbf{y} = softmax(f_{crt}(E(\mathbf{e_m}))) \qquad (4)$$

Model Architecture

Jointly Fine-tuning

There are two objectives for our model: to train the detection parameters f_det and to adjust the detection and correction modules to achieve an optimal balance. We jointly optimize the detection loss L_d and the correction loss L_c that:

$$\mathcal{L}_d = -\sum_i \log p(\hat{y}_{d_i} | \mathbf{x_w}; \theta_d) \quad (5)$$

$$\mathcal{L}_c = -\sum_i p(y_{d_i} | \mathbf{x_w}; \theta_d) \cdot \log p(\hat{y}_i | \mathbf{e_m}; \theta_c) \quad (6)$$

In particular, the correction loss is the negative log likelihood weighted by the probability of the detection result, $p(y_{d_i} | \mathbf{x_w}; \theta_d) \in (0.5, 1]$. This is to distinguish between the responsibilities of the two tasks.

The adaptive weighting objective enables us to jointly train our model with the sum of the two loss functions:

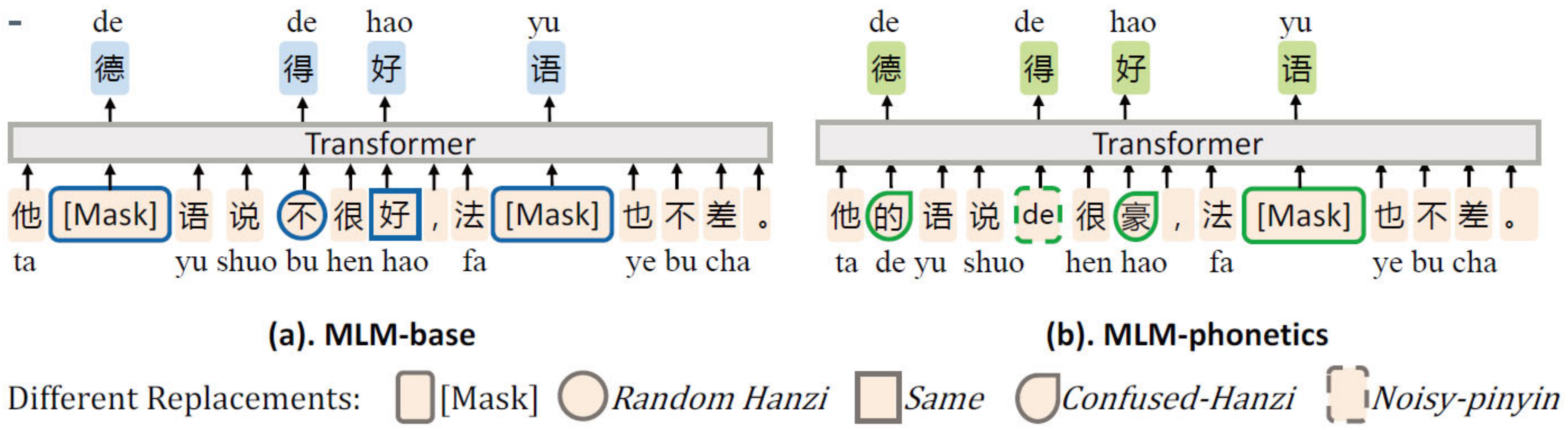
$$\mathcal{L} = \mathcal{L}_d + \mathcal{L}_c \quad (7)$$

Model Architecture

Pre-training MLM-phonetics
In this section, we introduce our pre-trained language model, MLM-phonetics, that 1) integrates phonetic features and 2) solves the problems of using standard masked language model in our CSC architecture.

The input sentences in CSC are with errors, which are different from the pre-training samples. Some work thus far side-stepped the input divergence by avoiding to input error sentences directly to pre-trained models. For example, Zhang et al. (2020) use a bidirectional-GRU for error detection before a BERT-based correction network.

In order to take advantage of the pre-training technique, we modify the pre-training task. In the pre-training of a standard mask language model(MLM-base), the model is trained by predicting 15% randomly selected characters which are replaced with the [MASK] token, random character, and themselves at the sampling rate of 80%, 10%, and 10%, respectively.

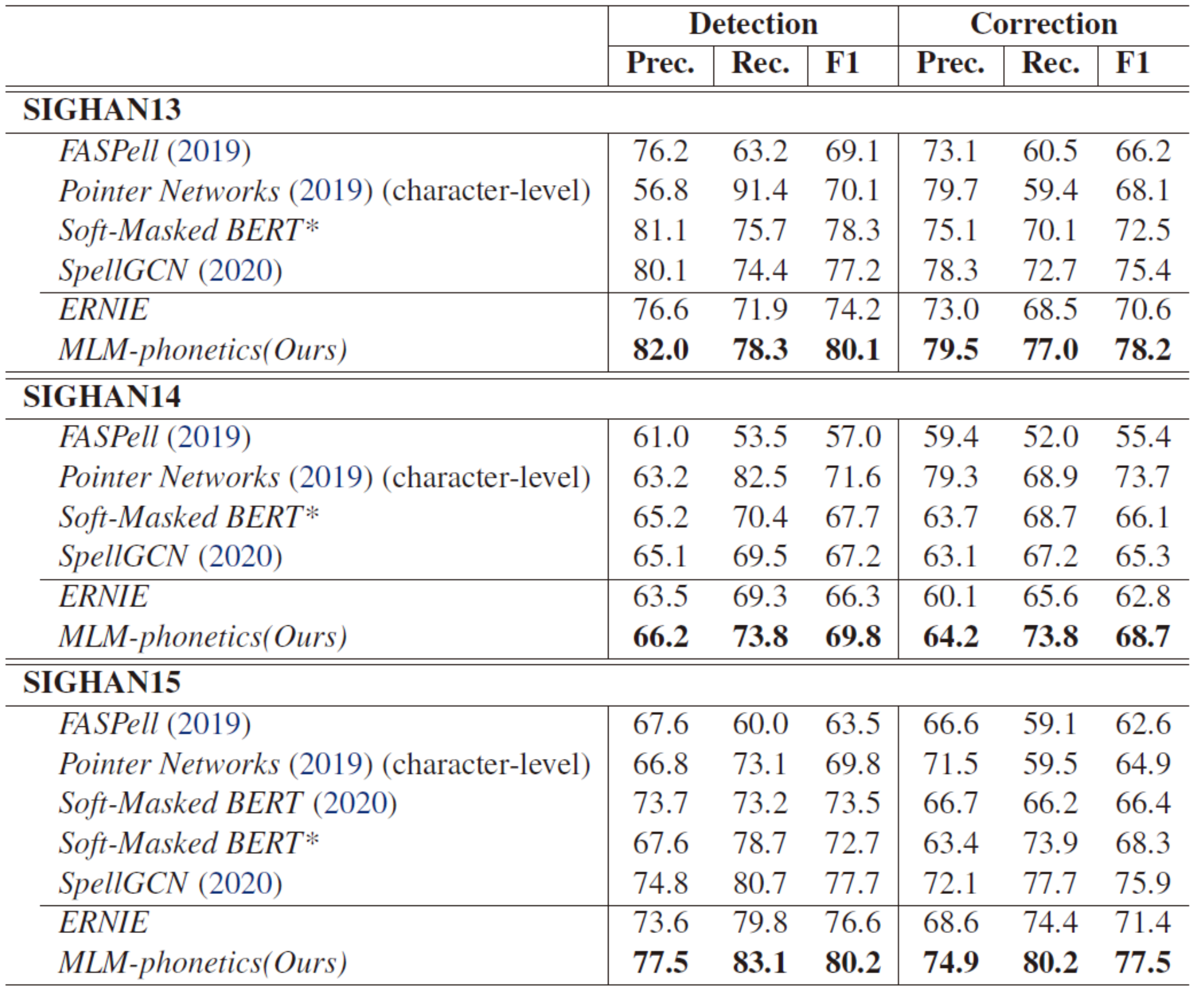


de
德
de
得
hao
好
yu
语
Transformer
他 [Mask] 语 说 不 很 好 , 法 [Mask] 也 不 差 。
ta yu shuo bu hen hao fa ye bu cha
(a). MLM-base
de
德
de
得
hao
好
yu
语
Transformer
他 的 语 说 de 很 豪 , 法 [Mask] 也 不 差 。
ta de yu shuo hen hao fa ye bu cha
(b). MLM-phonetics
Different Replacements:
[Mask]
Random Hanzi
Same
Confused-Hanzi
Noisy-pinyin

To avoid input divergence and integrate phonetic features, we propose two pre-training replacements: confused-Hanzi and a noisy-pinyin.

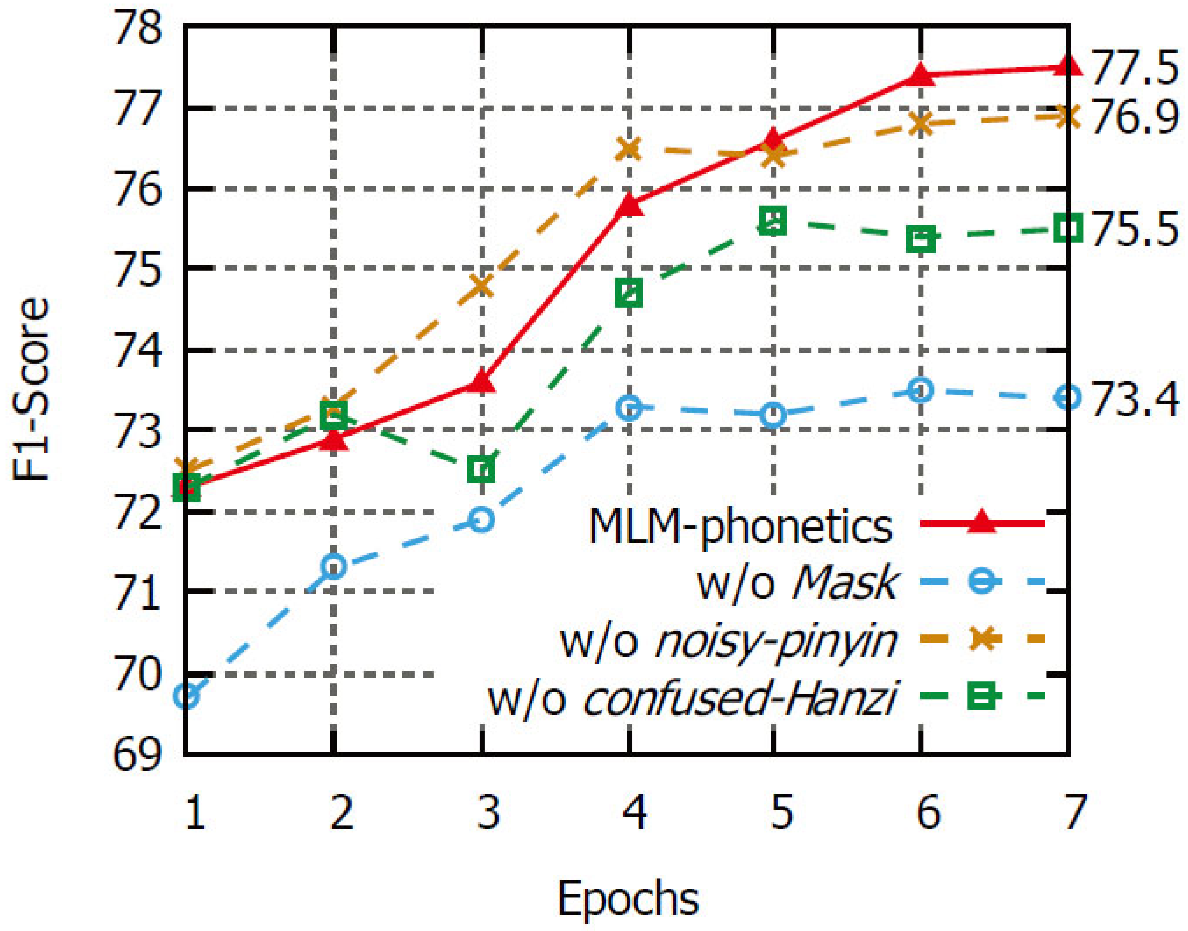
In the pre-training of MLM-phonetics, our data generator randomly chooses 20% of token positions in the training samples. If the ith token is chosen, we empirically replace it with (1) the [MASK] token 40% of the time, (2) the noisy-pinyin of this token 30% of the time, and (3) a confused-Hanzi from its confusion set 30% of the time. Then MLM-phonetics is trained to predict the original sentence from the sentence with replacements.

Experiments



	Detection			Correction		
	Prec.	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**
SIGHAN13						
FASPell (2019)	76.2	63.2	69.1	73.1	60.5	66.2
Pointer Networks (2019) (character-level)	56.8	91.4	70.1	79.7	59.4	68.1
*Soft-Masked BERT**	81.1	75.7	78.3	75.1	70.1	72.5
SpellGCN (2020)	80.1	74.4	77.2	78.3	72.7	75.4
ERNIE	76.6	71.9	74.2	73.0	68.5	70.6
MLM-phonetics(Ours)	**82.0**	**78.3**	**80.1**	**79.5**	**77.0**	**78.2**
SIGHAN14						
FASPell (2019)	61.0	53.5	57.0	59.4	52.0	55.4
Pointer Networks (2019) (character-level)	63.2	82.5	71.6	79.3	68.9	73.7
*Soft-Masked BERT**	65.2	70.4	67.7	63.7	68.7	66.1
SpellGCN (2020)	65.1	69.5	67.2	63.1	67.2	65.3
ERNIE	63.5	69.3	66.3	60.1	65.6	62.8
MLM-phonetics(Ours)	**66.2**	**73.8**	**69.8**	**64.2**	**73.8**	**68.7**
SIGHAN15						
FASPell (2019)	67.6	60.0	63.5	66.6	59.1	62.6
Pointer Networks (2019) (character-level)	66.8	73.1	69.8	71.5	59.5	64.9
Soft-Masked BERT (2020)	73.7	73.2	73.5	66.7	66.2	66.4
*Soft-Masked BERT**	67.6	78.7	72.7	63.4	73.9	68.3
SpellGCN (2020)	74.8	80.7	77.7	72.1	77.7	75.9
ERNIE	73.6	79.8	76.6	68.6	74.4	71.4
MLM-phonetics(Ours)	**77.5**	**83.1**	**80.2**	**74.9**	**80.2**	**77.5**

Experiments



Error Analysis

We summarize the two classes on the SIGHAN15 testset and the proportion of the Detection Error and Correction Error is 83.6% and 16.4%, respectively. This reveals that most of the false predictions are Detection Errors.

We decompose the 83.6% detection errors into the two types and find that false negative errors and false positive errors account for 41.1% and 42.5% respectively. The proportions of the two error types are almost equal.

A possible reason is that some homonyms are indistinguishable, such as “的”, “地”, and “得”. All of the three characters have the pronunciation of “de” and it makes sense to use any of these candidates in many sentences, phonetically or semantically.

PHMOSpell: Phonological and Morphological Knowledge Guided Chinese Spelling Check

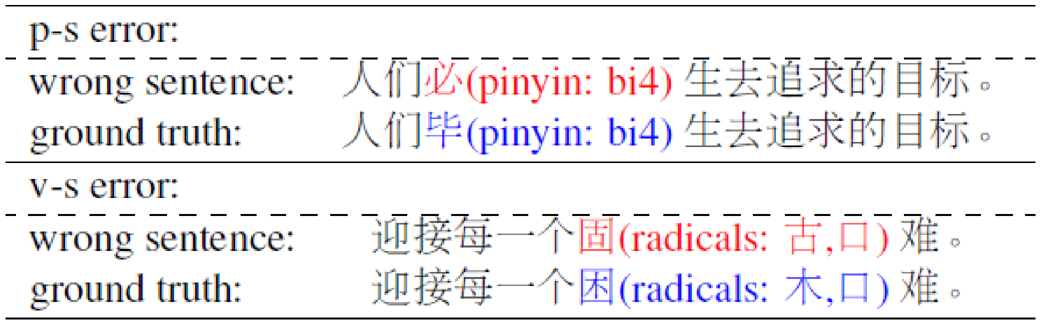
Li Huang[1,2], Junjie Li[2], Weiwei Jiang[2], Zhiyu Zhang[2], Minchuan Chen[2], Shaojun Wang[2] and Jing Xiao[2]

[1]Fudan University
[2]Ping An Technology
lhuang9703@gmail.com, junjielee815@gmail.com

Introduction

- Chinese Spelling Check (CSC) aims to automatically detect and correct spelling errors in Chinese sentences. These errors typically consist of human writing errors and machine recognition errors by automatic speech recognition (ASR) or optical character recognition (OCR) systems (Yu et al., 2014).

- Since a great number of Chinese characters are similar either in phonology or morphology, they are easily misused with each other. According to (Liu et al., 2011), 76% of Chinese spelling errors belong to phonological similarity error and 46% belong to visual similarity error.

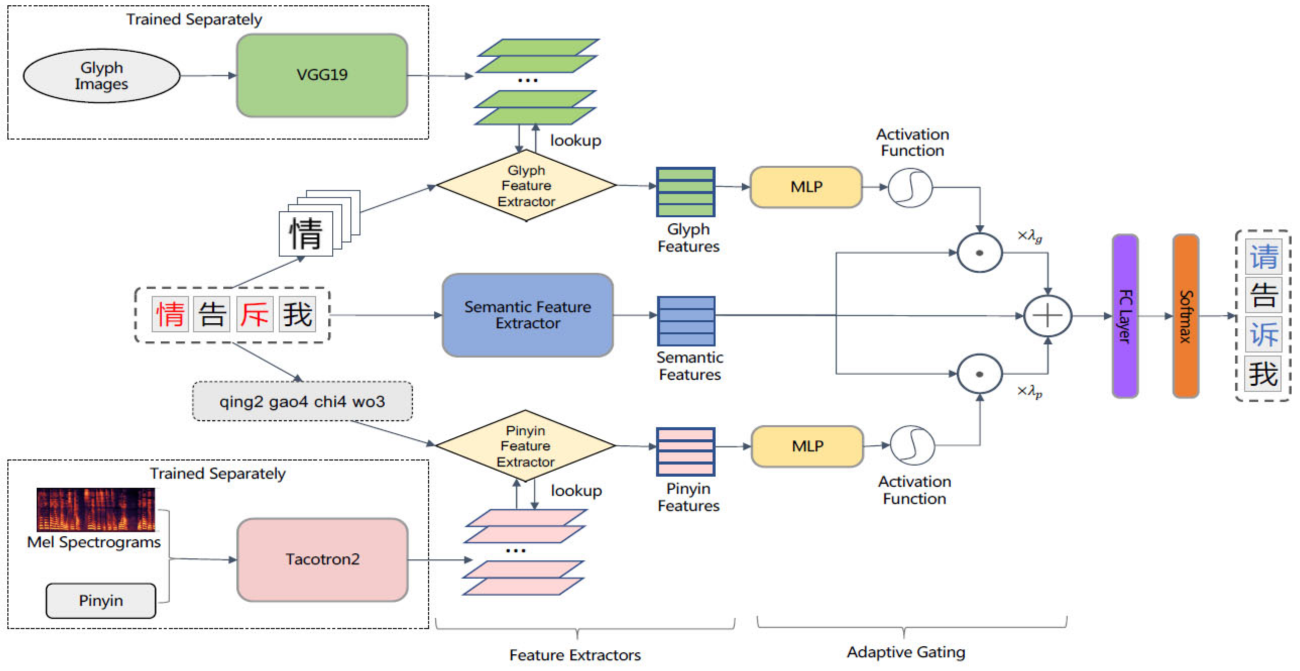


p-s error:	
wrong sentence:	人们必(pinyin: bi4) 生去追求的目标。
ground truth:	人们毕(pinyin: bi4) 生去追求的目标。
v-s error:	
wrong sentence:	迎接每一个固(radicals: 古,口) 难。
ground truth:	迎接每一个困(radicals: 木,口) 难。

Related Work

- Rule based methods
  - Mangu and Brill (1997) proposed a rule based approach for automatically acquiring linguistic knowledge from a small set of easily understood rules.
  - Jiang et al. (2012) arranged a new grammar system of rules to solve both Chinese grammar errors and spelling errors.
  - Xiong et al. (2015)’ s HANSpeller was based on an extended HMM, ranker based models and a rule based model.
- Statistical based methods
  - Noisy Channel Model is the most widely used model. Statistical based methods usually narrowed the candidates choice by utilizing a predefined confusion set, which contains a set of similar character pairs.
- Deep learning based methods

Model Architecture



Our model consists of three feature extractor modules and an adaptive gating module used to fuse kinds of features.

Model Architecture

Pinyin Feature Extractor
In this paper, we leverage Tacotron2, a recurrent sequence-to-sequence mel spectrograms prediction network, to help modeling the phonological representations since its location-sensitive attention can create effective time alignment between the character sequence and the acoustic sequence.

When training a Chinese TTS system with Tacotron2, characters are first converted to pinyin sequence as phoneme form. Then the sequence is represented by the encoder using an embedding layer and the hidden representations are consumed by the decoder to predict a corresponding mel spectrogram one frame at a time.

Motivated by this, we train Tacotron2 separately using public Chinese female voice datasets with teacher forcing. During training, we utilize pinyin transcription and mel spectrograms as input to help modeling pinyin representations. Then we extract pinyin embedding layer of the encoder as our pinyin feature extractor to generate the phonological representations for CSC.

$$\mathbf{F}_p = \{\mathbf{f}_1^p, \mathbf{f}_2^p, ..., \mathbf{f}_n^p\}$$

Model Architecture

Glyph Feature Extractor

To take advantage of powerful pre-trained models and avoid training from scratch, VGG19 (Simonyan and Zisserman, 2014) pretrained on ImageNet is adopted as the backbone of the glyph feature extractor.

Following (Meng et al., 2019), we further finetune it with the objective of recovering the identifiers from glyph images to solve the problem of domain adaptation. After that, we drop the last classification layer and use the outputs of VGG19’ s last max pooling layer as glyph features.

For a given sentence X, our glyph feature extractor is able to first retrieve images for its characters and then generate glyph features:

$$\mathbf{F}^g = \{\mathbf{f}_1^g, \mathbf{f}_2^g, ..., \mathbf{f}_n^g\}$$

Model Architecture

Semantic Feature Extractor

Beyond the phonological and the morphological information, we adopt empirically dominant pretrained language model to capture semantic information from context. Following (Hong et al., 2019; Cheng et al., 2020; Zhang et al., 2020), BERT is employed as the backbone of our semantic feature extractor. Given an input sentence X, the extractor outputs hidden states:

$$\mathbf{F}^s = \{\mathbf{f}_1^s, \mathbf{f}_2^s, ..., \mathbf{f}_n^s\}$$

Model Architecture

Adaptive Gating
Most previous methods for CSC simply used addition or concatenation to fuse different features. However, these fusion strategies ignore the relationship between the features. To tackle this issue, we propose an innovative adaptive gating mechanism served like a gate to finely control the fusion of features. It is defined as follows:

$$AG(\mathbf{F}^p, \mathbf{F}^s) = \sigma(\mathbf{F}^p \mathbf{W}^p + \mathbf{b}^p) \cdot \mathbf{F}^s$$

$$AG(\mathbf{F}^g, \mathbf{F}^s) = \sigma(\mathbf{F}^g \mathbf{W}^g + \mathbf{b}^g) \cdot \mathbf{F}^s$$

The enriched feature F^e is calculated as follows:

$$\mathbf{F}^e = \lambda_p \cdot AG(\mathbf{F}^p, \mathbf{F}^s) + \lambda_g \cdot AG(\mathbf{F}^g, \mathbf{F}^s)$$

Finally, we add residual connection to F^e and F^s by linear combination:

$$\mathbf{F}^{es} = \mathbf{F}^e + \mathbf{F}^s$$

Model Architecture

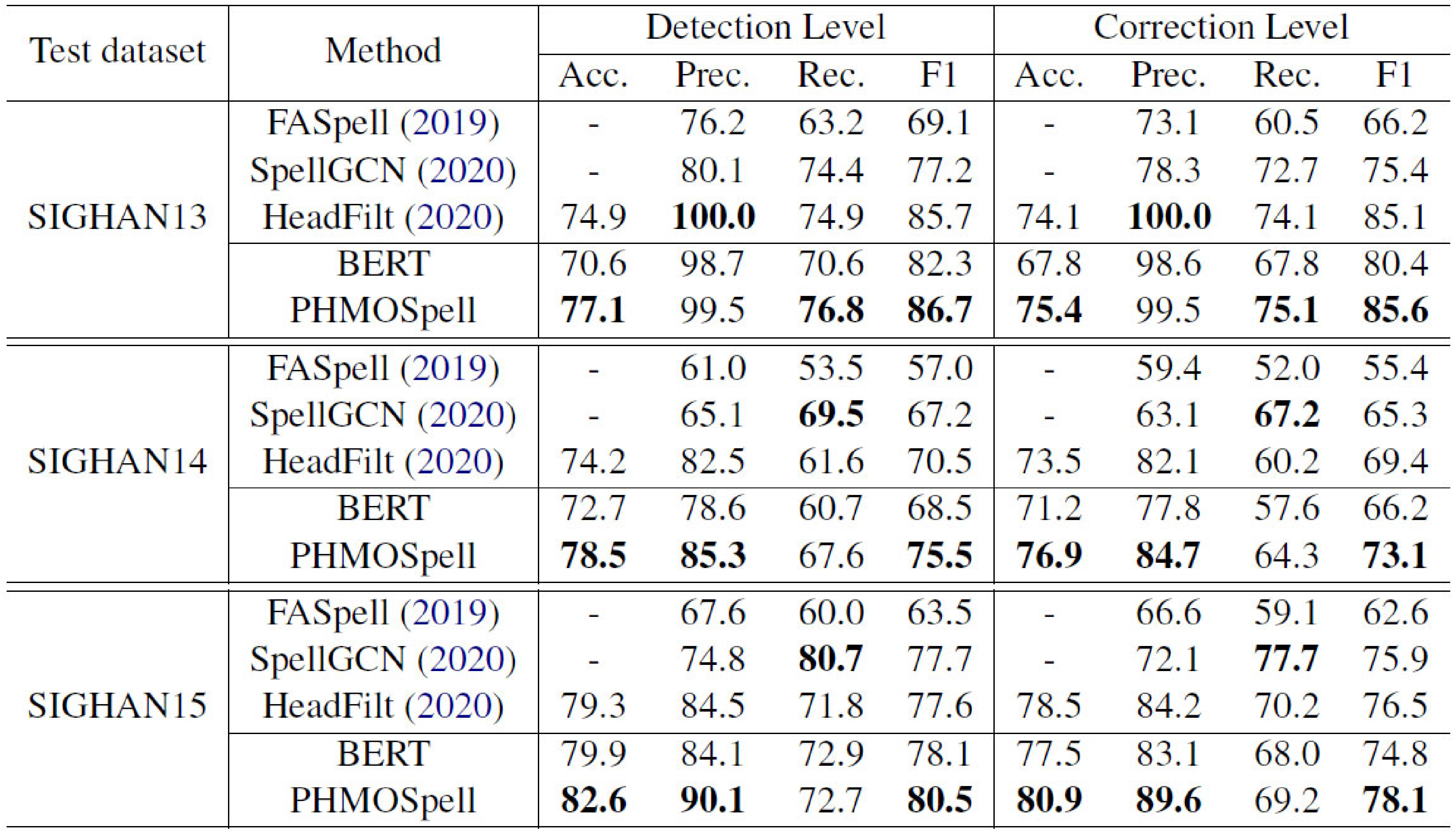
Training
During the training process, the representation F^{es} is fed into a fully-connected layer for the final classification, which is defined as follows:

$$P(Y_p|X) = softmax(\mathbf{F}^{es}\mathbf{W}^{fc} + \mathbf{b}^{fc})$$

The goal of training the model is to match the predicted sequence Y_p and the ground truth sequence Y_g. Overall, the learning process is driven by minimizing negative log-likelihood of the characters:

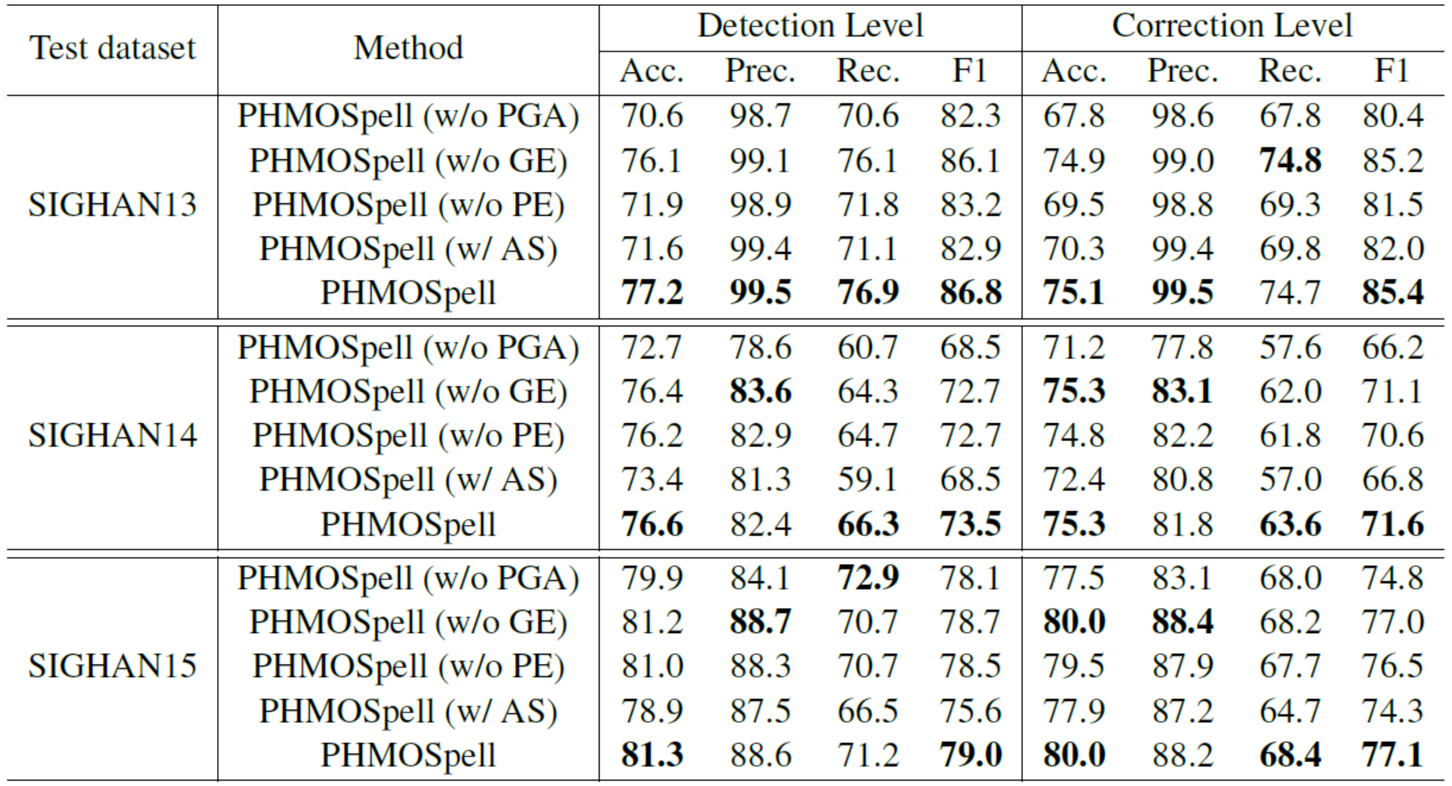
$$\mathcal{L} = -\sum_{i=1}^{n} logP(\hat{y_i} = y_i|X)$$

Experiments



Test dataset	Method	Detection Level				Correction Level			
		Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
SIGHAN13	FASpell (2019)	-	76.2	63.2	69.1	-	73.1	60.5	66.2
	SpellGCN (2020)	-	80.1	74.4	77.2	-	78.3	72.7	75.4
	HeadFilt (2020)	74.9	**100.0**	74.9	85.7	74.1	**100.0**	74.1	85.1
	BERT	70.6	98.7	70.6	82.3	67.8	98.6	67.8	80.4
	PHMOSpell	**77.1**	99.5	**76.8**	**86.7**	**75.4**	99.5	**75.1**	**85.6**
SIGHAN14	FASpell (2019)	-	61.0	53.5	57.0	-	59.4	52.0	55.4
	SpellGCN (2020)	-	65.1	**69.5**	67.2	-	63.1	**67.2**	65.3
	HeadFilt (2020)	74.2	82.5	61.6	70.5	73.5	82.1	60.2	69.4
	BERT	72.7	78.6	60.7	68.5	71.2	77.8	57.6	66.2
	PHMOSpell	**78.5**	**85.3**	67.6	**75.5**	**76.9**	**84.7**	64.3	**73.1**
SIGHAN15	FASpell (2019)	-	67.6	60.0	63.5	-	66.6	59.1	62.6
	SpellGCN (2020)	-	74.8	**80.7**	77.7	-	72.1	**77.7**	75.9
	HeadFilt (2020)	79.3	84.5	71.8	77.6	78.5	84.2	70.2	76.5
	BERT	79.9	84.1	72.9	78.1	77.5	83.1	68.0	74.8
	PHMOSpell	**82.6**	**90.1**	72.7	**80.5**	**80.9**	**89.6**	69.2	**78.1**

Experiments

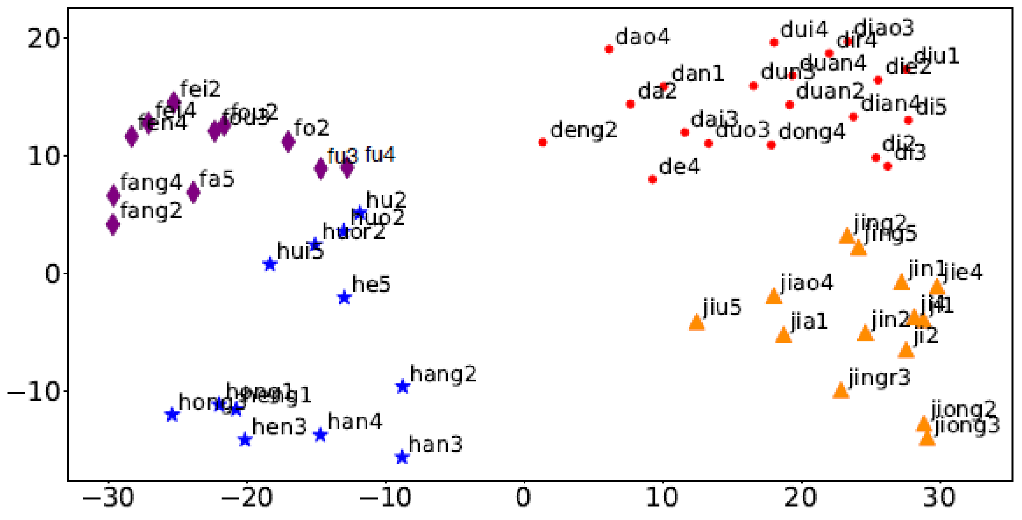


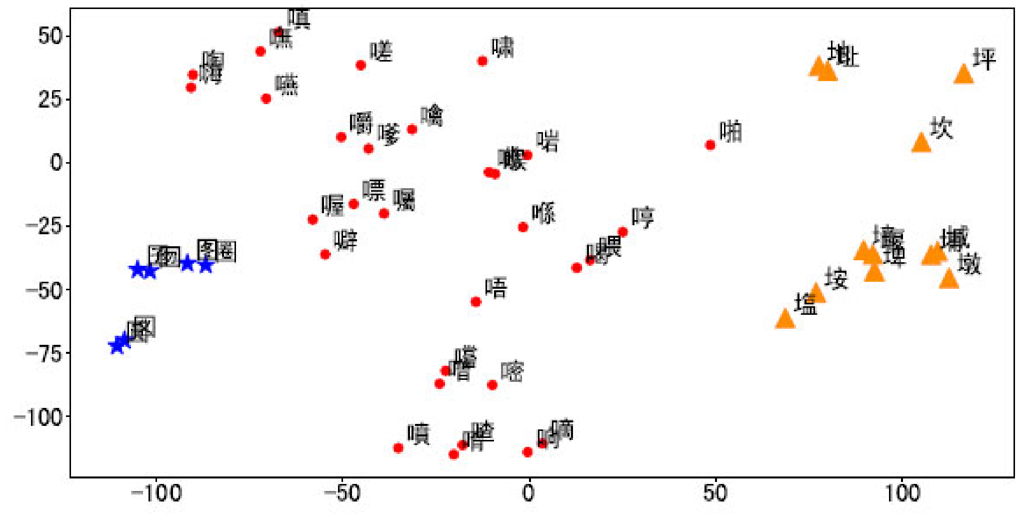
Test dataset	Method	Detection Level				Correction Level			
		Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
SIGHAN13	PHMOSpell (w/o PGA)	70.6	98.7	70.6	82.3	67.8	98.6	67.8	80.4
	PHMOSpell (w/o GE)	76.1	99.1	76.1	86.1	74.9	99.0	**74.8**	85.2
	PHMOSpell (w/o PE)	71.9	98.9	71.8	83.2	69.5	98.8	69.3	81.5
	PHMOSpell (w/ AS)	71.6	99.4	71.1	82.9	70.3	99.4	69.8	82.0
	PHMOSpell	**77.2**	**99.5**	**76.9**	**86.8**	**75.1**	**99.5**	74.7	**85.4**
SIGHAN14	PHMOSpell (w/o PGA)	72.7	78.6	60.7	68.5	71.2	77.8	57.6	66.2
	PHMOSpell (w/o GE)	76.4	**83.6**	64.3	72.7	**75.3**	**83.1**	62.0	71.1
	PHMOSpell (w/o PE)	76.2	82.9	64.7	72.7	74.8	82.2	61.8	70.6
	PHMOSpell (w/ AS)	73.4	81.3	59.1	68.5	72.4	80.8	57.0	66.8
	PHMOSpell	**76.6**	82.4	**66.3**	**73.5**	**75.3**	81.8	**63.6**	**71.6**
SIGHAN15	PHMOSpell (w/o PGA)	79.9	84.1	**72.9**	78.1	77.5	83.1	68.0	74.8
	PHMOSpell (w/o GE)	81.2	**88.7**	70.7	78.7	**80.0**	**88.4**	68.2	77.0
	PHMOSpell (w/o PE)	81.0	88.3	70.7	78.5	79.5	87.9	67.7	76.5
	PHMOSpell (w/ AS)	78.9	87.5	66.5	75.6	77.9	87.2	64.7	74.3
	PHMOSpell	**81.3**	88.6	71.2	**79.0**	**80.0**	88.2	**68.4**	**77.1**

Table 4: Ablation results on three datasets. PHMOSpell (w/ AS) denotes replacing adaptive gating module with aggregate strategy for feature fusion. PHMOSpell (w/o PE) denotes model without pinyin feature extractor. PHMOSpell (w/o GE) denotes model without glyph feature extractor. PHMOSpell (w/o PGA) denotes model without pinyin, glyph feature extractor and adaptive gating, which is a vanilla BERT implementation.

Features Visualization

To understand the effectiveness of our features more intuitively, we reduce features from high-dimensional space to low-dimensional space and visualize some of them using t-SNE.





Discussion

Examples:

“...不惜娱(pinyin: yu2)弄大臣...”

vanilla BERT corrects “娱弄”as“玩弄”

our model: “愚弄”

“...那别人的欢(radicals: 又,欠) 说是没办法改变你的...”

vanilla BERT: “小说”

out model: “劝说”

Wrong cases:

“...他们有时候，有一点捞到...”

“...天将降大任于斯人也，必先苦其心智，劳其筋骨...”