# 基于层次化修正框架的文本纠错模型

叶俊民,罗达雄,陈 曙
(华中师范大学计算机学院,湖北武汉 430079)

**摘 要:** 文本中存在的表达冗余、词汇误用和内容缺失等错误会显著影响文本语义的理解,当前解决上述文本错误的纠错模型存在两个主要的问题:当前的文本纠错模型主要基于编码器-解码器框架,解码速度较慢;许多工作将错误检测和修正分离成两个任务,没有形成统一的整体.为此,提出了一种基于层次化修正框架的文本纠错模型.首先,基于预训练模型建模得到文本的多种语义表示;其次,利用文本的语义表示识别出文本中错误的位置;最后,利用层次化修正框架计算精化的修正操作并完成对错误的修正.针对公开文本纠错数据集CONLL-14进行了相关实验,结果表明本文模型比所选取的对比模型有更快的解码速度和更高的召回率.

**关键词:** 文本纠错;预训练模型;层次化修正框架;深度学习
**中图分类号:** TP391 **文献标识码:** A **文章编号:** 0372-2112 (2021)02-0401-07
**电子学报URL:** http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20200448

## A Text Error Correction Model Based on Hierarchical Editing Framework

YE Jun-min, LUO Da-xiong, CHEN Shu
(*School of Computer Science, Central China Normal University, Wuhan, Hubei* 430079, *China*)

**Abstract:** Redundant expressions, misuse of words, and missing content and other text errors can seriously affect the interpretation of text semantics. There exist two major problems with current text error correction models: The Encoder-Decoder based text error correction models have slow decoding speed; Text error detection task and text correction task are handled as two separate tasks. Hence, a text error correction model based on a hierarchical editing framework is proposed in this paper. Firstly, a variety of text semantic representations are obtained through modelling pre-trained model. Secondly, text errors are located by using these text semantic representations. Finally, on the basis of hierarchical editing framework, precise editing operations are worked out to edit the errors. Experiments on the published text error correction dataset show that the proposed model has faster decoding speed and higher recall rate than comparison models.

**Key words:** text error correction; pre-trained model; hierarchical editing framework; deep learning

## 1 引言

在基于文本数据的语义理解和应用任务中,文本的质量显著地影响着对应任务的执行效果.因此,如何发现、定位与修正文本数据(下简称文本)中的错误是一个非常值得研究的问题.

对当前文本纠错研究工作主要有三类:(1)基于统计机器翻译的文本纠错研究,如Rozovskaya A等人将文本纠错任务转化为翻译任务来处理[1],该类纠错模型能够定位和修正简单的文法错误(如搭配错误),但无法有效处理文本的复杂语义错误;(2)基于编码器-解码器框架的文本纠错研究,如基于RNN编码器-解码器框架[2]、基于CNN编码器-解码器框架[3]和基于Transformer框架[4]的三类纠错模型,这些模型利用局部语义信息可发现大部分的文本错误,但共性问题是:解码器生成正确文本的速度较慢;所得出的解在一定程度上存在偏差;(3)基于大型预训练语言模型的文本纠错研究,由于这类模型可并行计算,故加快了模型训练和解码的速度,实验结果和基于编码器-解码器的最佳结果近似[5],但这类方法存在着将文本错误的检测和修正分成不同时间段的两个任务、缺失词不在预先设定的词表中、难以处理语序不当错误等问题.

上述研究表明,文本纠错存在待解决的问题是:(1)解码速度较慢的问题;(2)将文本错误的检测和修正相分离的问题.

## 2 问题定义

设 $\boldsymbol{s}=[s_1,\cdots,s_n]\in R^n$ 是带有错误的源文本,$n$ 为源

收稿日期:2020-05-13;修回日期:2020-08-02;责任编辑:马兰英
基金项目:国家社会科学基金后期资助项目(No. 20FTQB020)

文本的词个数. $\boldsymbol{t}=[t_1,\cdots,t_m]\in R^m$是对 $s$ 进行纠错后的目标文本,$m$ 为目标文本的词个数.

文本纠错的目标定义一个学习函数 $f$,使得 $t=f(s;\boldsymbol{\theta})$, $f$ 中的参数 $\boldsymbol{\theta}=\text{argmin}_\theta(L(f(s),t))$,其中 $L$ 为损失函数. 下面定义相关术语.

**定义 1** 文本差异 $diff(\boldsymbol{s},\boldsymbol{t})$. 设 $\boldsymbol{s}$ 是源文本,$\boldsymbol{t}$ 是目标文本,$diff(\boldsymbol{s},\boldsymbol{t})$是将源文本转换成目标文本所需进行的操作. $diff(\boldsymbol{s},\boldsymbol{t})$采用文献[5]中改进的 Levenshtein distance algorithm 算法得到.

**定义 2** 修正操作集合. edit_vocab 包含了从 1 开始的多个正整数编号,其中每一个编号对应一个具体的修正操作,具体定义如表 1 所示.

**表 1 修正操作集合**

| 操作名 | 对应的操作编号属于的整数区间 | 操作的含义与实例 |
|---|---|---|
| 复制(C)[5] | 1 | 保留源文本当前位置单词(记为$s_i$),如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,1,\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_i,\cdots]$ |
| 删除(D)[5] | 2 | 删除源文本当前位置单词(记为$s_{i+1}$),如:<br>$\boldsymbol{s}=[\cdots,s_i,s_{i+1},s_{i+2},\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,1,2,1\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_i,s_{i+2},\cdots]$ |
| 插入常见词(I)[5] | $[3,3+\sum_{i=1}^{1}op\_num_i]$<br>($op_num_1$为通过统计得到的常见插入词表中的词个数) | 在源文本当前位置单词(记为$s_i$)后一个位置,插入常见插入词表中的词,如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,op,\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_i,the,\cdots]$ |
| 插入后缀(IS) | $[3+\sum_{i=1}^{1}op\_num_i,3+\sum_{i=1}^{2}op\_num_i]$<br>($op_num_2$为常见的后缀个数) | 在源文本当前位置单词(记为$s_i$)词尾,插入常见后缀,如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,op,\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_i+ing,\cdots]$ |
| 插入表外词(IOOV) | $[3+\sum_{i=1}^{2}op\_num_i,3+\sum_{i=1}^{3}op\_num_i]$<br>($op_num_3$为不在常见插入词表中的其他词的个数) | 在源文本当前位置单词(记为$s_i$)后一个位置,插入表外词(记为 document),如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,op,\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_i,\text{document},\cdots]$ |
| 替换常见词(R)[5] | $[3+\sum_{i=1}^{3}op\_num_i,3+\sum_{i=1}^{4}op\_num_i]$<br>($op_num_4$为常见替换词表中的词个数) | 将源文本当前位置的单词(记为$s_i$),替换为常见替换词表中的词,如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,op,\cdots]$,<br>$\boldsymbol{t}=[\cdots,Because,\cdots]$ |
| 词型变换(RT) | $[3+\sum_{i=1}^{4}op\_num_i,3+\sum_{i=1}^{5}op\_num_i]$<br>($op_num_5$为常见词型变换的个数) | 将源文本当前位置的单词(记为$s_i$),进行词型变换,如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,op,\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_i-e+s,\cdots]$ |
| 替换为源文本中的词(RS) | $[3+\sum_{i=1}^{5}op\_num_i,3+\sum_{i=1}^{6}op\_num_i]$<br>($op_num_6$为模型可接受的源文本的最大长度,一般为 64 或 128) | 将源文本当前位置的单词(记为$s_i$),替换成源文本中另一个单词(记为$s_1$),如:<br>$\boldsymbol{s}=[\cdots,s_i,\cdots]$,<br>$edits(\boldsymbol{s})=[\cdots,op,\cdots]$,<br>$\boldsymbol{t}=[\cdots,s_1,\cdots]$ |

**定义 3** 修正操作. 设 $edits(\boldsymbol{s})=[e_1,\cdots,e_n]$是源文本的修正操作,其中,$e_i\in edit\_vavab$. 当输入源文本 $\boldsymbol{s}$,通过计算得出针对输入源文本的修正操作 $edits(\boldsymbol{s})$,并利用 $edits(\boldsymbol{s})$修改源文本 $\boldsymbol{s}$,以得到目标文本 $\boldsymbol{t}$.

基于上述定义,本文要解决的文本纠错问题定义如下.

**定义 4** 文本错误修正.

**输入** 带有错误的源文本 $\boldsymbol{s}=[s_1,\cdots,s_n]$;

**输出** 对 $\boldsymbol{s}$ 进行纠错后的目标文本 $\boldsymbol{t}=[t_1,\cdots,t_m]\in R^m$.

针对定义 4 中的问题,基于深度学习理论,设计一种层次化修正框架的文本纠错模型(图 1).

该模型工作原理如下:(1)输入为带有错误的源文本 $\boldsymbol{s}=[s_1,\cdots,s_n]$;(2)利用 BERT 得到源文本的多种语义表示$\boldsymbol{s}_{\text{sementic}}=[\boldsymbol{H}_1,\cdots,\boldsymbol{H}_n,\boldsymbol{I}_1,\cdots,\boldsymbol{I}_n,\boldsymbol{R}_1,\cdots,\boldsymbol{R}_n]$(参见 3.1 节),包括原语义、插入语义和替换语义表示;(3)得到每个位置的错误概率$[\boldsymbol{is_error}_1,\cdots,\boldsymbol{is_error}_n]$,定位文本中错误位置 $i$(参见 3.2 节);(4)针对 $i$,采用层次化修正框架得到其修正操作(参见 3.2 节);(5)根据(4)得到修正操作列表 $edits(\boldsymbol{s})$,利用该列表修正 $\boldsymbol{s}$ 得到正确文本 $\boldsymbol{t}$.

## 3 文本纠错模型

### 3.1 多种语义建模

针对源文本$\boldsymbol{s}=[w_1,\cdots,w_n]$的每一个词,依据

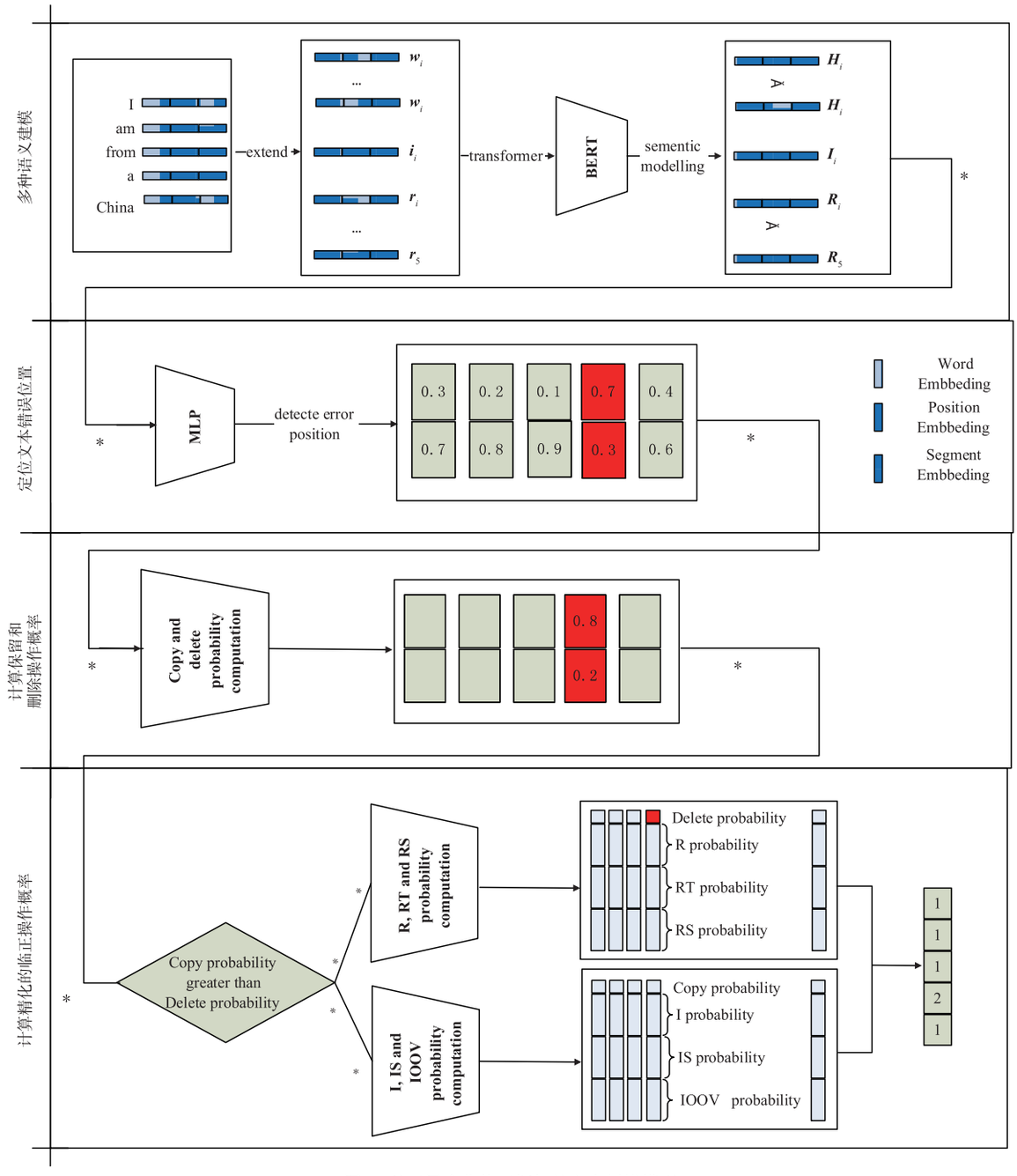



图1　基于层次化修正框架的文本纠错模型

BERT 的输入要求,其原语义表示的初始输入如式(1)所示.

$$\boldsymbol{s}=[\boldsymbol{w}_1,\cdots,\boldsymbol{w}_n],\boldsymbol{w}_i=[\boldsymbol{emb}_{\mathrm{word}[i]};\boldsymbol{emb}_{pos[i]};\boldsymbol{emb}_{seg}] \quad (1)$$

其中,$\boldsymbol{emb}_{\mathrm{word}[i]}$是词嵌入;$\boldsymbol{emb}_{pos[i]}$是位置嵌入;$\boldsymbol{emb}_{seg}$是段嵌入;“;”是向量的拼接运算.

为了计算插入语义和替换语义表示,对输入$\boldsymbol{s}$进行扩展[5],在$\boldsymbol{s}$中增加插入语义表示的初始输入$\boldsymbol{i}_i$和替换语义表示的初始输入$\boldsymbol{r}_i$(式(2)).

$$\begin{cases}\boldsymbol{s}_{\mathrm{extend}}=[\boldsymbol{w}_1,\cdots,\boldsymbol{w}_n,\boldsymbol{i}_1,\cdots,\boldsymbol{i}_n,\boldsymbol{r}_1,\cdots,\boldsymbol{r}_n]\\ \boldsymbol{i}_i=[\boldsymbol{MASK};\dfrac{\boldsymbol{emb}_{pos[i]}+\boldsymbol{emb}_{pos[i+1]}}{2};\boldsymbol{emb}_{seg}]\\ \boldsymbol{r}_i=[\boldsymbol{MASK};\boldsymbol{emb}_{pos[i]};\boldsymbol{emb}_{seg}]\end{cases} \quad (2)$$

其中,$\boldsymbol{MASK}$表示[$mask$]字符串的词嵌入,$\boldsymbol{i}_i$为源文本位置$i$和$i+1$的插入语义表示的初始输入,$\boldsymbol{r}_i$为源文本位置$i$的替换语义表示的初始输入.

将$\boldsymbol{s}_{\mathrm{extend}}$通过 BERT 对上下文语义进行建模(式

(3)). 在多头注意力机制计算权重时,三类语义表示遵循不同的规则,具体为:(1)对原语义表示,每个词针对原语义表示的所有词计算注意力权重;(2)对插入语义表示,每个词针对自身与原语义表示的所有词计算注意力权重;(3)对替换语义表示,每个词针对自身和原语义表示中不与当前词位置对应的词计算注意力权重.

$$\boldsymbol{s}_{\text{sementic}} = \boldsymbol{BERT}(\boldsymbol{s}_{\text{extend}}) = [\boldsymbol{H}_1, \cdots, \boldsymbol{H}_n, \boldsymbol{I}_1, \cdots, \boldsymbol{I}_n, \boldsymbol{R}_1, \cdots, \boldsymbol{R}_n] \tag{3}$$

其中,$\boldsymbol{H}_i$为源文本对应 $i$ 位置的原语义表示;$\boldsymbol{I}_i$为源文本对应 $i$ 和 $i+1$ 位置的插入语义表示;$\boldsymbol{R}_i$表示源文本对应 $i$ 位置的替换语义表示.

### 3.2 错误检测与修正

在建模三种语义表示的基础上,利用两层的全连接神经网络得到每个位置是否错误的概率(式(4)).

$$[\boldsymbol{is\_error}_1, \cdots, \boldsymbol{is\_error}_n] = MLP([\boldsymbol{H}_1, \cdots, \boldsymbol{H}_n]) \tag{4}$$

其中,$\boldsymbol{is\_error}_i \in R^2$,其中,$\boldsymbol{is_error}_i[0]$表示 $i$ 位置存在错误的概率,$\boldsymbol{is_error}_i[1]$表示 $i$ 位置正确的概率.

当 $\boldsymbol{is_error}_i[0]$大于 $\boldsymbol{is_error}_i[1]$时,模型认为 $i$ 位置存在错误,随即完成对文本错误的检测. 在此定义第一个损失函数(式(5)).

$$Loss_1 = \sum_{s \in S} \sum_{i \in [1, len(s)]} \text{Cross\_Entropy}(\boldsymbol{is\_error}_i, \boldsymbol{real\_error}_i) \tag{5}$$

其中,$S$ 为源文本的集合,$len$ 为计算文本词个数的函数,Cross_Entropy 为交叉熵损失函数,$\boldsymbol{real_error}_i$是 $i$ 位置是否错误的标识.

针对计算所得到的存在错误的相应位置,基于文献[5],本模型层次化地计算其需要施加的修正操作:

(1)设 $i$ 位置的词的保留概率为$\text{copy}_i$,而词的删除概率记为$\text{delete}_i$(式(6)).

$$\begin{cases} \text{copy}_i = \boldsymbol{W}_{\text{copy}}^{\text{T}} \boldsymbol{H}_i + \boldsymbol{emb}_{\text{word}[i]}^{\text{T}} \boldsymbol{H}_i \\ \text{delete}_i = \boldsymbol{W}_{\text{delete}}^{\text{T}} \boldsymbol{H}_i \end{cases} \tag{6}$$

其中,$\boldsymbol{W}_{\text{copy}}$和$\boldsymbol{W}_{\text{delete}}$是分别计算复制概率$\text{copy}_i$和删除概率$\text{delete}_i$的权重向量.

(2)依据保留概率$\text{copy}_i$和删除概率$\text{delete}_i$之值,进行如下处理:

($a$)当$\text{delete}_i > \text{copy}_i$时,模型计算替换常见词、词型变换和替换为源文本中的词的概率(式(7)).

$$\begin{cases} R_i(w) = \boldsymbol{W}_{R(w)}^{\text{T}} \boldsymbol{H}_i + \boldsymbol{emb}_{R(w)}^{\text{T}} \boldsymbol{R}_i - \boldsymbol{emb}_{\text{word}[i]}^{\text{T}} \boldsymbol{R}_i \\ RT_i(\text{transformer}) = \boldsymbol{W}_{RT(\text{wtransformer})}^{\text{T}} \boldsymbol{H}_i + \boldsymbol{emb}_{\text{word}[i]}^{\text{T}} \boldsymbol{H}_i \\ RS_i(sp) = (\boldsymbol{emb}_{RS(sp)}^{\text{T}} \boldsymbol{R}_i - \boldsymbol{emb}_{\text{word}[i]}^{\text{T}} \boldsymbol{R}_i - \boldsymbol{MASK}^{\text{T}} \boldsymbol{R}_i) * \sigma \end{cases} \tag{7}$$

其中,$\boldsymbol{W}_{R(w)}$计算替换为常见替换词表中的词 $w$ 的权重,$\boldsymbol{emb}_{R(w)}$是 $w$ 的词嵌入;$\boldsymbol{W}_{RT(\text{transformer})}$计算进行词型变换 transformer 的权重;$\boldsymbol{emb}_{RS(sp)}$是源文本 $sp$ 位置上的词的词嵌入,而 $\sigma$ 为阈值,用于控制概率值的大小.

($b$)当$\text{delete}_i < copy_i$时,模型计算替换插入常见词、插入后缀和插入表外词的概率(式(8)).

$$\begin{cases} I_i(w) = \boldsymbol{W}_{I(w)}^{\text{T}} \boldsymbol{H}_i + \boldsymbol{emb}_{\text{word}[i]}^{\text{T}} \boldsymbol{H}_i + \boldsymbol{emb}_{I(w)}^{\text{T}} \boldsymbol{I}_i \\ IS_i(suffix) = \boldsymbol{W}_{IS(suffix)}^{\text{T}} \boldsymbol{H}_i + \boldsymbol{emb}_{\text{word}[i]}^{\text{T}} \boldsymbol{H}_i \\ IOOV_i(w) = \boldsymbol{emb}_{IOOV(w)}^{\text{T}} \boldsymbol{I}_i \end{cases} \tag{8}$$

其中,$\boldsymbol{W}_{I(w)}$为计算插入常见插入词表中的词 $w$ 的权重,$\boldsymbol{emb}_{I(w)}$是 $w$ 的词嵌入;$\boldsymbol{W}_{IS(suffix)}$为计算插入后缀 $suffix$ 的权重;$\boldsymbol{emb}_{IOOV(w)}$是表外词 $w$ 的词嵌入.

依据式(8)~(10)的计算结果,定义模型的第二个损失函数(式(9)).

$$Loss_2 = \sum_{s \in S} \sum_{i \in [1, len(s)]} \text{Cross\_Entropy}(\boldsymbol{p}_i, \boldsymbol{r\_p}_i) \tag{9}$$

其中,$\boldsymbol{p}_i$是模型输出的 $i$ 位置进行不同修正操作的概率,$\boldsymbol{r_p}_i$是 $i$ 位置真实的修正操作标识.

整个模型的总损失函数如式(10)所示.

$$Loss = \alpha * Loss_1 + (1 - \alpha) * Loss_2 \tag{10}$$

其中,$\alpha$ 为平衡两个损失的参数;$Loss_1$为错误定位任务的损失;$Loss_2$为错误修正任务的损失.

## 4 实验结果及分析

### 4.1 实验数据和测评指标

首先,本文的实验数据包括预训练数据和对比实验的训练和预测数据. 首先,选择 One Billion Word Benchmark[5]数据集构建预训练数据. 其次,对比实验的训练数据为三个公开的文本纠错数据集[5],即 Lang-8、NUCLE 和 FCE. 最后,选择 CoNLL-2014[5]作为对比实验的测试数据.

本文实验中所使用的测评指标有 4 个,分别为错误检测的准确率 $S$、文本纠错的查准率 $P$、查全率 $R$ 和 $M^2F_{0.5}$分数. 这四个测评指标具体的计算方法如式(11)所示.

$$\begin{cases} S = \dfrac{\sum\limits_{s \in \text{Source}} \sum\limits_{i \in [1, len(s)]} I(s_i^r, s_i^p)}{\sum\limits_{s \in \text{Source}} len(s)} \\ P = \dfrac{\sum\limits_{s \in \text{Source}} |\, e(s) \cap g(s) \,|}{\sum\limits_{s \in \text{Source}} |\, e(s) \,|} \\ R = \dfrac{\sum\limits_{s \in \text{Source}} |\, e(s) \cap g(s) \,|}{\sum\limits_{s \in \text{Source}} |\, g(s) \,|} \\ F_{0.5} = \dfrac{(1 + 0.5^2) * P * R}{R + 0.5^2 * P} \end{cases} \tag{11}$$

### 4.2 预处理

#### 4.2.1 构造预训练数据

采用了数据集 One Billion Word Benchmark 中的文

本构造得到预训练数据,在此过程中根据采样的错误类型,基于文献[5],对文本 $s$ 的位置 $i$ 按照如下规则进行构造操作:

**规则 1**　当采样为插入常见词错误时,若当前位置为插入常见词表中的词,则将其删除.

**规则 2**　当采样为替换常见词错误时,若当前位置为替换常见词表中的词,则将其随机替换成另一个词.

**规则 3**　当采样为删除错误时,则在当前位置的后一个位置处插入一个随机词.

**规则 4**　当采样为插入后缀和词型变换错误时,若当前位置处存在其他形式的插入后缀和词型变换,则随机选取一个错误的变换应用到当前位置.

#### 4.2.2　统计常见的插入和替换词

统计训练数据中源文本被修改为目标文本时所插入/替换的词的出现次数,选择出现次数较多的插入/替换词组成常见插入/替换词. 在本文中,常见插入/替换词个数均为 1000.

#### 4.2.3　生成源文本的修正操作

由于训练数据经过处理后为源文本-目标文本的形式,而模型可处理的形式为源文本-修正操作形式,所以需要定义两种形式之间的转换方法.

(1)申明修正操作 $e$;(2)计算源文本和目标文本的文本差异 *diff*;(3)遍历 *diff* 中的每一个元素 *diff*[$i$];(4)当 *diff*[$i$]的操作为复制或删除时,将 $e[i]$赋值为复制或删除操作在 edit_vocab 中对应的编号;(5)当 *diff*[$i$]的操作为插入时,依据以下规则进行赋值.

**规则 1**　$e[i-1]$为复制操作时,当插入的内容为后缀 suffix,将 $e[i]$赋值为 suffix 对应的操作编号;当插入的内容为常见插入词 I_word,将 $e[i]$赋值为 I_word 对应的操作编号;当插入的内容为表外词 O_word,将 $e[i]$赋值为 O_word 对应的操作编号.

**规则 2**　$e[i-1]$为删除操作时,当插入的内容和删除的内容可以通过词型变换 transformer 得到,将 $e[i]$赋值为 transformer 对应的操作编号;当插入的内容为常见替换的词 R_word,将 $e[i]$赋值为 R_word 对应的操作编号;当插入的内容为源文本 index 位置的词,将 $e[i]$赋值为 index 对应的操作编号.

#### 4.2.4　词嵌入聚类

在插入表外词操作中,通常需要对所有词进行聚类操作:(1)提取 BERT 中已训练词嵌入表示;(2)利用 PCA 算法将嵌入的维度从 1024 维转换到 16 维;(3)利用 K-MEANS 算法将 28996 个词嵌入归类到 300 个类别中并得到其聚类中心.

### 4.3　实验设置及超参选择实验

#### 4.3.1　实验设置

利用 TensorFlow 框架实现模型,其重要参数设置如下:(1)常见插入和替换词均为 1000,插入后缀和词型变换的个数为 61,表外词为 28996,对其聚类个数为 300;(2)多种语义建模部分遵循 BERT-LARGE 的设置,错误检测与修正部分的超参 $\sigma$ 通过实验选取;(3)优化方法为(Adam,batch_size)为 64,模型的最大句子长度为 128.

#### 4.3.2　超参数选择及实验结果

针对所提出模型在不同的超参数和结构设置上进行实验,具体如表 2 所示:首先,在文本错误的检测任务上,本文的模型可以达到 91.9% 的准确率,即定位了多数文本错误.

**表 2　超参选择实验**

| | $P$ | $R$ | $F_{0.5}$ |
|---|---|---|---|
| Model1 | 64.01 | 42.52 | 58.13 |
| $\text{Model2}_{0.5}$ | 65.33 | 43.43 | 59.46 |
| $\text{Model2}_{0.6}$ | 64.45 | 43.91 | 58.93 |
| $\text{Model2}_{0.7}$ | **62.94** | **44.34** | **58.07** |
| $\text{Model2}_{0.8}$ | 62.43 | 44.78 | 57.87 |
| $\text{Model2}_{0.9}$ | 56.54 | 46.63 | 54.23 |
| Model3 | 63.66 | 43.42 | 58.23 |

将去除了替换为源文本中的词和插入表外词两个操作后的模型命名为 Model1. 从表 2 可以看出,本文模型所实施的修正操作的正确率为 64.01%,同时找到了测试集中 42.52% 的文本错误.

现将在 Model1 的基础上增加了替换为源文本的词操作的模型命名为 Model2,不同的阈值设置通过下标表示,根据表 2:(1)与 Model1 相比,模型 $\text{Model2}_{0.5}$和模型 $\text{Model2}_{0.6}$的 $p$ 值和 $r$ 值均有提升;(2)$\text{Model2}_{0.7}$和 $\text{Model2}_{0.8}$的 $r$ 值都有接近 2% 的提升,说明模型可以有效发现此类错误. 同时,两个模型的 $p$ 值有所下降,说明新增加的操作中存在错误的修正. 同时,与 $\text{Model2}_{0.5}$和 $\text{Model2}_{0.6}$的相比,二者的 $F_{0.5}$值均较低. 但是,由于 $F_{0.5}$是更偏重于 $p$ 值,当使用偏重于 $r$ 值 $F$ 指标时(如 $F_{1.5}$),$\text{Model2}_{0.7}$($F_{1.5}$值为 48.73)和 $\text{Model2}_{0.8}$($F_{1.5}$值为 48.97)的 $F_{1.5}$值大于 $\text{Model2}_{0.5}$($F_{1.5}$值为 48.65)和 $\text{Model2}_{0.6}$($F_{1.5}$值为 48.67)的 $F_{1.5}$值. 由于增加此结构的目的是发现更多的错误,所以本文认为 $\text{Model2}_{0.7}$和 $\text{Model2}_{0.8}$是更好的模型. 最终,根据 $\text{Model2}_{0.7}$和 $\text{Model2}_{0.8}$的 $F_{0.5}$值,本文选择 0.7 作为最终的阈值;(3)将 $\text{Model2}_{0.7}$的基础上增加了插入表外词操作的模型命名为 Model3,Model3 的 $p$ 值有一定程度上升,$r$ 值则出现了一定程度的下降,说明此结构在纠错方面的作用还不明显. 最终选择 $\text{Model2}_{0.7}$作为最优的模型.

#### 4.3.3　对比实验结果

本文将所提出的模型与多种类别的模型组进行对

比,首先说明作为对比的模型组的情况.

**模型组 1**[6] 即 Transformer + Pre-training + LM + Spellcheck + Ensemble Decoding.

**模型组 2**[4] 即 Transformer + Pre-training + Iterative refinement + Ensemble Decoding.

**模型组 3**[2] 即 SMT + NMT (Bi-GRU) + LM + Ensemble Decoding + Spellcheck.

**模型组 4**[3] 即 CNN + LM + Ensemble Decoding + Spellcheck.

**模型组 5**[5] 即 BERT + Sequence labeling + Iterative refinement + Factorize pre-trained bidirectional LMs.

对比实验结果如表 3 所示.

**表 3 对比实验结果**

| | $P$ | $R$ | $F_{0.5}$ |
|---|---|---|---|
| 模型组 1 | **71.6** | 38.7 | **61.2** |
| 模型组 2 | 66.7 | 43.9 | 60.4 |
| 模型组 3 | 66.8 | 34.5 | 56.3 |
| 模型组 4 | 65.5 | 33.1 | 54.8 |
| 模型组 5 | 66.1 | 43.2 | 59.7 |
| 本文模型 | 62.9 | **44.3** | 58.07 |

根据表 3,相关结果分析如下.

(1)模型组 1、模型组 3 和模型组 4 的 $r$ 值均达不到 40%,说明这些模型组均以忽略了文本中的许多错误为代价而取得了更高的修正准确性.

(2)本文的模型有最高 $r$ 值,说明本文模型能够发现并正确修正更多的错误,在重视 $r$ 值的 $F$ 指标下(如 $F_{1.5}$),本文模型($F_{1.5}$ 值为 48.73)优于模型 5($F_{1.5}$ 值为 48.35). 总体而言,本文模型的效果略差于模型 2($F_{1.5}$ 值为 49.06). 但是,本文模型比模型 2 有更快的解码速度.

(3)现说明本文模型对比基于编码器-解码器框架的模型有更快的解码速度. 在相同的实验环境下对比模型组 2、模型组 5 和本文模型在解码数据集 CONLL-14 中的不同源文本长度数据时所需要的解码时间. 具体的实验环境设置为:操作系统为 Ubuntu16.04,服务器为 NVIDIA Tesla V100-16G,Python 版本为 Python3.6,Tensorflow 版本为 Tensorflow-gpu-1.12. 根据图 2 所示,首先,基于编码器-解码框架的模型随着源文本长度的增长,解码时间趋近于线性增长. 其次,PIE 模型和本文模型解码时间不会随着源文本长度的增长而明显增加. 最后,与 PIE 模型相比,本文模型需要更多解码时间.

#### 4.3.4 相关工作对比

(1)与基于编码器-解码器框架的文本纠错模型[2-4,6]相比,本文模型在实现原理上不同,具体表现在:($a$)基于编码器-解码器框架的纠错模型将文本纠错任务视为生成任务,而本文模型将文本纠错任务视为标注任务;($b$)基于编码器-解码器框架的纠错模型使用了 S2S 的相关技术,本文模型使用了大型预训练语言模型中的相关技术;($c$)基于编码器-解码器的纠错模型需要按时序解码,而本文模型则可并行地进行解码.

(2)与文献[5]相比,主要区别在于:($a$)文献[5]直接计算源文本中每个位置的修正操作概率,而本文模型使用层次化的计算方式,首先定位存在错误的位置,然后再针对错误位置计算需要施加的修正操作;($b$)文献[5]定义了 5 类修正操作概率,而本文模型定义了 8 类修正操作,故可以发现更多类型的文本错误.

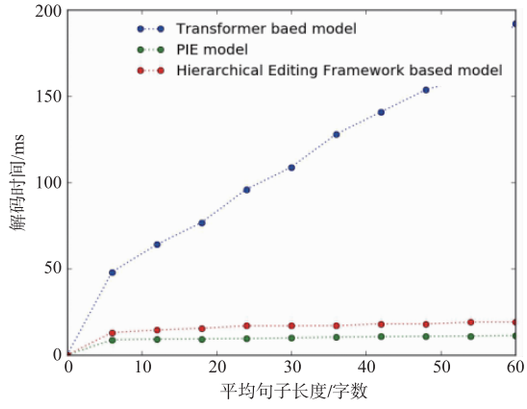



图2 不同源文本长度下的解码时间

## 5 结论

针对当前文本纠错模型解码速度较慢以及将检测与修正相分离的问题,在 PIE 模型基础上,本文提出了一种基于层次化修正框架的文本纠错模型,该模型提供了相比编码 - 解码方法更快的解码速度,同时能够发现更多的文本错误,并在公开数据集上进行了相关实验,结果表明本文模型在保证一定准确率的情况下能够发现并正确修正更多类型的文本错误,并且解码的速度快于基于编码器-解码器框架的模型.

本文的方法还存在以下有待研究的问题:(1)在插入表外词操作中,探索更有效的词表划分方法;(2)在替换成源文本中的词操作中,研究替换为源文本中连续词序列的方法;(3)针对更多的错误类型从整体结构的层面对模型进行改进. 以上问题将在未来的研究中进一步讨论.

#### 参考文献

[1] Rozovskaya A, Roth D. Grammatical error correction: Machine translation and classifiers [A]. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics [C]. Berlin, Germany, 2016. 2205 – 2215.

[2] Grundkiewicz R, Junczys-Dowmunt M. Near human-level

performance in grammatical error correction with hybrid machine translation[A]. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics:Human Language Technologies[C]. New Orleans,USA,2018. 284 – 290.

[3] Chollampatt S, Ng H T. A multilayer convolutional encoder-decoder neural network for grammatical error correction[A]. Proceedings of the Thirty-Second AAAI Conference on Artifificial Intelligence[C]. New Orleans, Louisiana,USA,2018. 5755 – 5762.

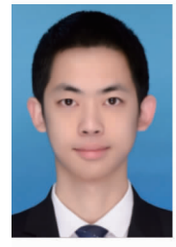
[4] Zhao W, Wang L, Shen K, et al. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data[A]. Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics[C]. Minneapolis, Minnesota,USA,2019. 156 – 165.

[5] Awasthi A, Sarawagi S, Goyal R, et al. Parallel iterative edit models for local sequence transduction[A]. Proceedings of Empirical Methods in Natural Language Processing[C]. Hong Kong,China,2019. 4259 – 4269.

[6] Lichtarge, Jared, et al. Corpora generation for grammatical error correction[A]. Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics[C]. Minneapolis, Minnesota, USA, 2019. 3291 – 3301.

## 作者简介

**叶俊民**　男,1965 年 10 月出生于四川省成都市. 2005 年毕业于哈尔滨工程大学,博士,现为华中师范大学教授,博士生导师,当前研究方向为学习分析、软件工程.
E-mail:jmye@ mail. ccnu. edu. cn



**罗达雄(通信作者)**　男,1995 年 2 月出生于福建省南平市. 2020 年毕业于华中师范大学,现为 VIVO(杭州)公司算法工程师,当前研究方向为自然语言处理和教育数据挖掘.
E-mail:18140663659@163. com

**陈　曙**　男,1981 年 1 月出生于湖北省武汉市. 2009 年毕业于武汉大学,博士,现为华中师范大学计算机学院讲师,当前研究方向为软件工程和学习分析.
E-mail:chenshu@ ccnu. edu. cn