自然语言处理课程期末大作业: 语法纠错

林语盈[1)] 徐熔杞[1)] 黄逸轩[1)]

[1)](南开大学 计算机学院, 天津, 中国)

摘 要 南开大学 2022 年春自然语言处理课程期末作业。本文调研了中英文语法纠错的相关工作，复现了三篇论文，并提出了一些创新点。

关键词 自然语言处理; 语法纠错; 语法检测

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.01.2022.00001

Final Project:GEC

Yuying Lin[1)] Rongqi Xu[1)] Yixuan Huang[1)]

[1)](Department of Computer Science, Nankai University,Tianjin, China)

Abstract This is the final project of natural language processing course of Nankai University in spring 2022. This paper firstly investigates the related work of grammatical error correction, reproduces three papers, and puts forward some innovations

Key words NLP;GEC;GED

1 引言

本文调研了中英文语法纠错的相关工作，复现了三篇论文,并提出了一些创新点。本文的第二节是问题定义,第三节是目前国内外相关研究现状,第四、五、六节分别是复现的三篇论文，并提出了一些创新点。最后, 第六节是总结与展望。项目代码见https://gitee.com/nkunlpcourse-gecgroup/final-project。在附录中展示了运行过程中的截图和小组成员分工。

2 问题定义

语法纠错 (GEC) 是 NLP 领域中的一个重要任务，GEC 任务要求检测一句话中是否有语法错误，并自动将检测出的语法错误进行纠正。这里给出了四种语法错误的例子，如图1所示，包括遗漏单词（M）、冗余单词（R）、单词选择错误（S）和词序错误（W）。

问题的输入输出可简要概括如下:

- 输入：句子;
- 输出：句子是否正确，若有错误，输出错误类型、位置、改正方案。

收稿日期：2022-05-27；修改日期：2022-05-27 林语盈，E-mail: 2012174@mail.nankai.edu.cn. 徐熔杞，E-mail: 185010769@qq.com. 黄逸轩，E-mail: spacetime57@163.com.

Error Type	Original Sentence	Correct Sentence
M	每个城市的超市能看到这些食品。	每个城市的超市**都**能看到这些食品。
R	我和妈妈**是**不像别的母女。	我和妈妈不像别的母女。
S	最重要的是**做**孩子想学的环境。	最重要的是**创造**孩子想学的环境。
W	“静音环境”**是对人体应该**有危害的。	“静音环境”**应该是对人体**有危害的。

图 1 四种语法错误例，遗漏单词（M）、冗余单词（R）、单词选择错误（S）和词序错误（W）

3 目前国内外研究现状

当前主要的模型可分为基于端对端神经网络机器翻译的方法 (包括基于 RNN、CNN、Transformer 的方法)、基于大规模预训练模型的方法、序列标注的方法等。

3.1 基于神经机器翻译的方法

与统计机器学习的方法不同，此类方法无需构建句子的特征，而是直接采用正确句-错误句的平行数据对作为模型的输入，采用编码器-解码器的结构。具体来说，编码器或解码器的选择包括 CNN、RNN 或注意力机制。大规模的预训练模型也被广泛尝试。

基于 RNN 模型 2017 年，Sakaguchi 等人创新性地将强化学习应用于 GEC，以更好地纳入任务特定指标，并克服传统 MLE 导致的暴露偏差。2018 年,[1] 提出了另一项使用基于 RNN 的 NMT 模型的新工作，该模型旨在提高源句子的流利度和可靠性，以纠正语法错误，称为流利度提升学习。这项工作

为使用基于 NMT 的方法解决 GEC 提供了一个新的视角。

基于 CNN 模型与基于 RNN 的模型相比，基于 CNN 的模型的优势在于，CNN 在捕捉本地上下文方面更有效，从而纠正更广泛的语法错误。多层体系结构可以捕获长期的上下文信息。此外，无论输入长度如何，只对输入执行恒定数量的非线性计算，而在 RNN 中，非线性的数量为 O（n），从而减少了远距离单词的影响。

基于 transformer 模型。随着 Transformer[2] 的提出，许多基于 NMT 的 GEC 模型用 Transformer 取代了传统的基于 RNN 的编码器-解码器，并取得了有希望的结果。2019 年，复制机制[3] 被创新地应用于 GEC 任务，允许模型将未更改的源单词直接复制到目标端，并实现了最先进的性能。与 RNN 和 CNN 相比，Transformer 通过注意在句子中建立远程依赖的能力更强，并且它能够实现更高效的并行计算。此外,[4] 提出了浅攻击解码 (SAD)，以提高 transformer 在线即时语法纠错 (GEC) 的计算性能。

3.2 大规模预训练模型

随着数据和算力的不断丰富，大规模预训练模型被用于 GEC 任务。2019 年,[5] 首先将大规模的预训练模型与 transformer 用于 GEC 任务。2020 年,[6] 将预训练的模型与编解码模型结合并运用到汉语语法纠错任务中。[7] 将预先训练好的掩码语言模型（MLM），整合到用于语法错误纠正（GEC）的编码器-解码器（EncDec）模型中。首先使用给定的 GEC 语料库对模型参数进行微调，然后将微调后的模型参数输出作为 GEC 模型中的附加特征。该模型在 BEA-2019 和 CoNLL-2014 中取得了当时最优的效果

3.3 基于序列标注的方法

序列标注模型只是对特定位置的词做特定标注的更改，以[8] 为例，将 GEC 问题转换为局部词语，即 token-level 的特定操作，如 3 所示。这样使得序列标注模型具有更小的解空间，相对于 Seq2seq 而言，效率大幅提升。在[9] 中十分详细地定义了各种类型的操作标签

4 GECToR −Grammatical Error Correction: Tag, Not Rewrite

本文的动机旨在解决 NMT-based 模型的三个问题：即推理速度慢、需要大量的训练数据以及可解释性问题。本文针对以上三点提出了序列标注方法，取代了原本的序列生成方法本文提出的基于序列标注的方法可概括为以下三个方面：

1)token-level transformation/edit

该方法扩大了输出序列的解空间，对于大小为 5000 的词表，包含了 4971 个基本变换（包括保持不变、删除和其他 1167 个依赖于 token 的添加词语操作以及 3802 个替换词语操作）以及 29 个与具体 tokne 独立的 g-transformation，下面具体阐述两类 transformation 方法，即 basic transformation 和 g-transformation

basic transformation：基本变换即最常见的 token-level 的操作，例如保持当前的词语不变、删除当前词语、在当前词语后添加词语以及将当前词语替换为其它词语

g-transformation： g-transformation 是基于特定任务的操作，例如转换词语的大小写、将当前词语与后一个词合并、将当前的词分解为两个词、名词的单复数变换、动词的形式变换等

2)Tagging model architecture

模型结构，本文使用的 GEC 序列标注模型采用基于预训练的 BERT 的 transformer 作为 encoder，其中 transformer 含有两个带有 softmax 的线性层。其中，分词操作依赖于具体地 transformer 的设计。为了在 token-level 传递信息，本文将每个 token 的第一个词经过 encoder 得到的特征传入后续的线性层，两个线性层分别负责错误检测和纠错操作类型的标注

3)Iterative sequence tagging approach

采用多次迭代进行纠错的方式，逐步的改正错误序列。随着每次迭代，纠正错误的次数逐渐减少，并且大多数错误的改正都发生在前两次迭代中。因此，对于迭代次数进行上限的限制可以在保证生成效果的同时提高效率

4.1 方法

预处理

预处理操作分为三个步骤：

- 将训练集/验证集中的序列数据对中的每一个错误的 token 与其相对应的纠正后的 token 建立映射关系
- 对于每个映射，找到其对应的变换类型
- 对于错误序列中的每个 source token，只保留一个变换类型, 这样在每个迭代周期中都只对

token 进行一种变换，通过多次迭代以达到最终对原序列进行语法纠错的目的

训练：训练过程分为三个步骤，即首先在合成的错误句子上的预训练，再在在错误句子上进行微调，最后在错误句子和正确句子上进行微调

4.2 可以做出的改进

- 设置一个不改变原来 token 的概率阈值，记为 confidence bias，增加模型的灵活度
- 在错误检测层，设置一个句子级别的 minimum error probabilityl 阈值，增加模型的稳定性

4.3 复现过程

4.3.1 一、复现方法描述

对该论文的复现可以分为数据预处理、模型训练、模型评价三个步骤。

4.3.2 二、复现的细节以及难点

数据预处理：由于模型的训练输入需要特定的数据格式，即含有 TAG 的，能反映从错误文本到正确文本的改正过程的映射的数据，因此利用官方代码中 utils 工具文件夹中的 preprocess_data.py 文件将含有错误语法的源文件 (source)，以及正确的目标文件 (target) 两者作为输入，得到一个具有上述格式的输出文件，这个文件就可以直接作为训练数据或验证数据输入到网络模型中

另外，由于文中在 stage2 中使用的数据集——FCE、Lang8、Locness 等数据集中一般只含有一个 m2 格式的文件，该文件的每一个模块都代表一个句子，其中第一行以 S 开始，代表 source 文本，后面的若干行以 A 开始，代表对修改方式的标注 (annotation)，代表不同句子的两个模块间以空行分隔。而这样的格式与 GECToR 模型所需要的输入格式不同，因此需要从这个文件中获得 source 文本和 target 文本，并再利用上述的预处理脚本得到满足输入格式的数据。而论文附带的官方代码中没有处理该问题的文件，因此需要自行处理：利用如下指令抽取 src 数据：

grep ^S m2_file | cut −c3− > $output.src$

再编写 gec_corr.py 脚本文件抽取正确的 target 文本，该脚本以添加到 gitee 仓库本文目录下的 utils 文件夹中

训练：训练分为三个阶段，分别是在合成数据上训练、在 FCE、Lang8、Locness 等有错误文本数据集上 Fine-Tune、第三阶段在 $W\&I+locness$ 的正确和错误混合文本上 Fine-Tune

第一阶段在大小为 900 万句子的数据集上训练且 batch_size=256，由于机器内存不足，故将该数据集变为原来的 0.3 倍大小，并将 batch_size 改为 128。复现结果将在下一小节展示

第二阶段论文中在 NUCLE, FCE, Lang8, $W\&I+locness$ 的混合数据集上训练，而由于 NUCLE 数据集没有找到，因此采用了其他数据集的混合作为最终第二阶段的数据

第三阶段在 $W\&I+locness$ 数据集上进行 Fine-Tune，此阶段不加赘述

4.3.3 模型评价：

根据论文中的评价方法，采用了 ERRANT 工具对模型在测试数据集上的准确率、召回率以及 $F_{0.5}$ 进行测试。ERRANT 需要先利用如下命令将 source 文本和模型预测得到的文本作为输入得到 hypothesis.m2 文件，代表模型预测的文本对 source 的修改：

errant_parallel -orig <orig_file> -cor <cor_file1> [<cor_file2> ...] -out <out_m2>

再用同样地命令将 source 文本和正确文本作为输入得到 reference.m2 文件，得到正确文本相对于 source 文本的修改。最终利用 ERRANT 工具的如下指令:

errant_compare -hyp <hyp_m2> -ref <ref_m2>

将两个 m2 文件作为输入得到模型预测结果在测试集上的 precision、recall 和 $F_{0.5}$

4.4 实验

4.4.1 一、实验数据集

1. stage1 中的数据集为合成数据集，大小为 2G，900 万个句子，实验中由于内存不足，减小为原来的 0.3 倍；
2. stage2 中在 FCE, Lang8, $W\&I+locness$ 上 Fine-Tune
3. stage3 中在 $W\&I+locness$ 上 Fine-Tune

4.4.2 二、实验设置

这里采用 xlnet 作为 transformer 模型进行实验，stage2 采用的是论文中 github 链接附带的 xlnet 在 stage1 中的预训练权重，而非上述在 0.3 倍合成数据集上 4 个 epoch 的结果。由于二、三阶段的数据量不算很大，因此参数完全按照论文中所给值进行复现，其中三个阶段相同的设置有：

1. transformer_model:'xlnet'

2. lr:1e-5

3. patience:3

以下是三个阶段各自的一些参数:

stage1: n_epoch=4,batch_size=128,pretrain=None

stage2: n_epoch=9,batch_size=128,pretrain=BEST_MODEL_FROM_STAGE1

stage3: n_epoch=4,batch_size=32,pretrain=BEST_MODEL_FROM_STAGE2

4.4.3 三、实验环境

Ubuntu 服务器 +32G 内存 +RTX3090(24G 显存)+Visual Studio Code、Python==3.7+Pytorch==1.9.1+cuda==11.1+ allennlp==0.8.4

4.4.4 四、复现结果

stage1:

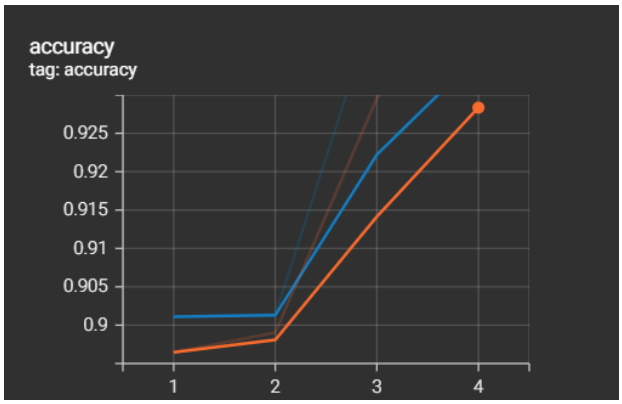

图 2 stage1:accuracy 随 epoch 变化

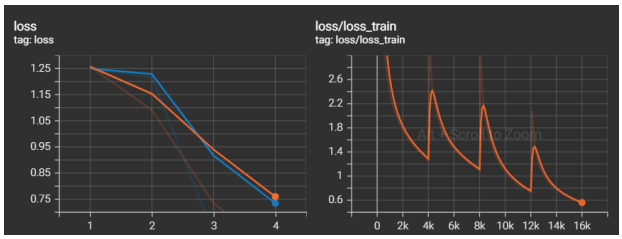

图 3 stage1:loss 随 epoch 变化

上面图2图3是 loss 和 accuracy 随着 epoch 的变化曲线，下图4则展示了每个 epoch 中更为具体和细节的 loss 与 accuracy 的变化:

利用 ERRANT 对上面在合成数据训练 4epoch 后各项指标评价如图5:

stage2

在 stage2 的训练过程中由于设置了 patience=3，即连续三个 epoch 没有提升就停止训练，因此，尽管设置了 n_epoch=9，但由于从第三个 epoch 开始

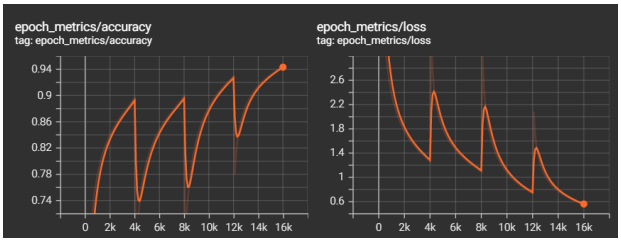

图 4 stage1:loss 和 accuracy 在每个 epoch 内部的变化

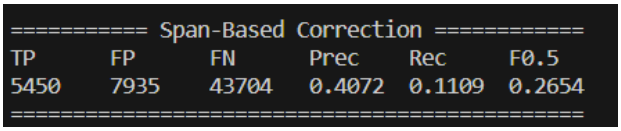

图 5 stage1:ERRANT 的评价结果

连续三个 epoch 没有提升因此在第五个 epoch 处停止，利用 tensorboard 对 loss 和 accuracy 可视化如图6、图7、图8: 可以看到第三个周期后验证集

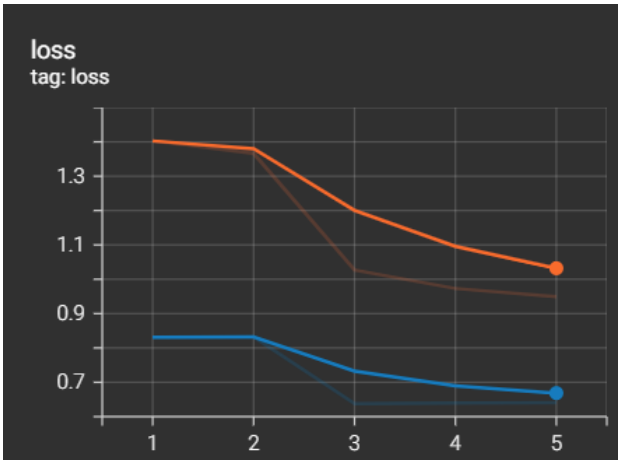

图 6 stage2:loss 随 epoch 变化

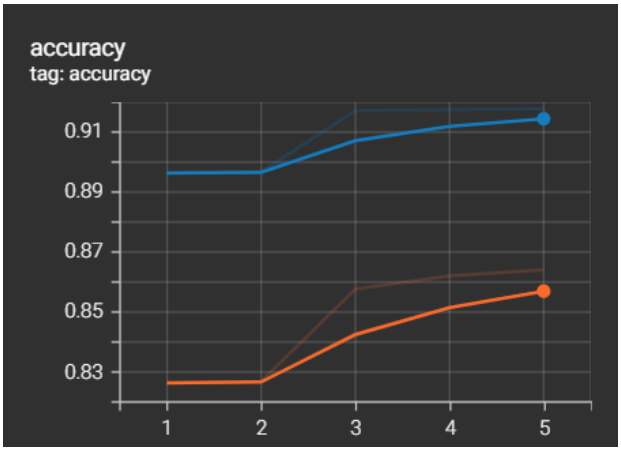

图 7 stage2:accuracy 随 epoch 变化

上 loss 几乎没有变化 (图6浅蓝色折线)，最终利用 ERRANT 评价各项指标如图9:

stage3

stage3 使用第二阶段的权重在 $W\&I+locness$

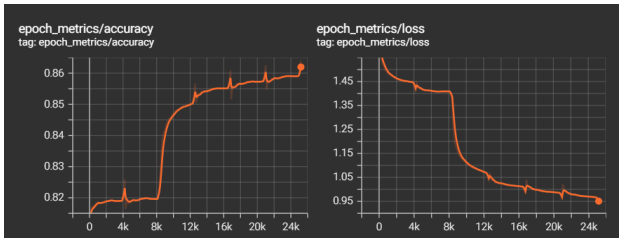

图 8　stage2:loss 和 accuracy 在每个 epoch 内部的变化

```
============ Span-Based Detection ============
TP      FP      FN      Prec      Rec      F0.5
21813   26701   27341   0.4496    0.4438   0.4484
==============================================
```

图 9　stage2 的各项指标

上 Fine-Tune，但发现虽然在训练集上的 loss 不断下降，accuracy 不断上升，但在验证集上却恰好相反，因此在经过 patience=3 后 model dump 停止了训练。这样由于只在 stage3 训练了 1 个 epoch，因此最终的指标与 stage2 几乎一致

4.4.5　五、结果分析

首先是在第一阶段，由于计算资源的限制 (RTX-3090,24G; 论文中使用的 GPU 是 NVIDIA-A100)，不得不将论文中使用的数据集减小为原来的 0.3 倍，batch_size 由 256 变为 128，尽管如此，在训练过程中，通过 nvidia-smi 查看 GPU 使用率高达百分之 90, 显存占用率也超过一半，因此最终的各项指标都低于论文中的结果，论文中在 xlnet 上的 $F_{0.5}$ 为 0.49，而复现结果仅为 0.27

其次是第二阶段，在各数据集上 Fine-tune 后的 $F_{0.5}$ 为 0.44，与论文中的 0.5 的结果有所差距；而第三阶段由于在 devset 上无法收敛，导致提前终止训练，因此，与论文中 0.65 的结果相差较大。分析原因是由于论文中的一些参数细节没有公开，例如除了 stage1 外的 batch_size、patience 还有其他许多参数可能与实验中使用的参数相差较大

5　Tail-to-Tail Non-Autoregressive Sequence Prediction for Chinese Grammatical Error Correction

5.1　模型方法描述

模型 TtT[10] 发表在 ACL 2021 上，是腾讯 AI Lab 在中文语法错误纠正上（CGEC）的又一重要成果。

对于 CGEC 任务，语法错误类型和对应的纠正操作可以总结为替换、插入和删除、局部转换三类，其中替换不会改变句子前后的长度，而插入和删除以及局部转化可能会改变句子的长度，属于变长操作。近年来，使用序列标注模型有效解决了 seq2seq 框架因为错误累计而出现幻象（hallucinate），即产生不相关或者反事实的内容的问题，但却因为需要给每个词进行标记而导致计算效率非常低。于此同时，单纯的标记方法需要进行多轮预测只至稳定，这进一步降低了效率。最近许多研究关注于在 CGEC 任务上精调 BERT[11] 等预训练语言模型，但是受限于模型本身，它们大部分都只能处理固定长度的错误纠正，而无法执行变长操作。

针对以上问题，论文提出了一个新的模型框架 TtT（Tail-to-Tail non-autoregressive sequence prediction）。其总体结构如图10所示。它仍然使用 Bert 作为骨干模型，可以将字符的信息直接从 bottom tail 传到 up tail，并且学习双向上下文的信息。为了能够同时进行替换、删除、插入、局部转换等各种操作，模型在 up tail 端使用了条件随机场（Conditional Random Fields, CRF)[12] 通过对相邻字符的依赖关系建模来进行非自回归的序列预测。另外，针对任务本身存在的类不平衡问题（句子中大部分字符是正确的，只有很少一部分需要修改），模型还使用了焦点损失惩罚策略[13]。

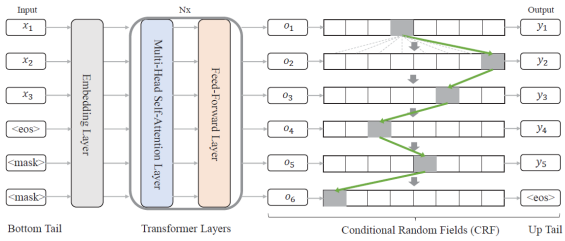

Figure 3: The proposed tail-to-tail non-autoregressive sequence prediction framework (TtT).

图 10　TtT 总体结构

5.1.1　变长输入

为了使得模型能够执行插入删除等变长操作，即输入 $X = (x_1, x_2, \cdots, x_T)$ 的长度 T 和输出 $Y = (y_1, y_2, \cdots, y_{T'})$ 的长度 T' 不一定相等，模型会对输入输出的格式进行一定的处理。假设输入 $X = (x_1, x_2, x_3, <\text{eos}>)$，当 $T = T'$ 时，不用处理；当 $T > T'$，比如 $Y = (y_1, y_2, <\text{eos}>)$，这意味着 X 中的某些字符被删除，那么在训练阶段，会在 Y 的尾部补充 $T - T'$ 个特殊的符号 <pad>，以使得 $T = T'$，即 $Y = (y_1, y_2, <\text{eos}>, <\text{pad}>)$ 当 $T < T'$ 时，比如 $Y = (y_1, y_2, y_3, y_4, y_5, <\text{eos}>)$，这就意味着在原始的输入 X 中插入了额外的信息，那么

模型将会在 X 的尾部插入特殊的符号 <mask>，来暗示这些位置可能可以插入一些新的字符，即 $X=\left(x_1, x_2, x_3, <\text{eos}>, <\text{mask}>, <\text{mask}>\right)$

5.1.2 双向语义建模

在准备好输入样本后，将会输入由预训练中文 BERT 模型初始化的嵌入层和堆叠的 Transformer 层来对语义信息建模。具体来说，首先将词向量和位置向量相加来获取其表征：

$$\mathbf{H}_t^0=\mathbf{E}_{w_t}+\mathbf{E}_{p_t} \tag{1}$$

其中 0 是层数标号，t 是状态标号，$\mathbf{E}_{w_t}$ 和 $\mathbf{E}_{p_t}$ 分别是 tokens 的词向量和位置向量。然后嵌入向量 $\mathbf{H}_t^0$ 会被输入多个 Transformer 层，使用多头自注意力机制来学习双向语义表征。

$$\begin{aligned}\mathbf{H}_t^1&=\mathbf{LN}\left(\mathrm{FFN}\left(\mathbf{H}_t^1\right)+\mathbf{H}_t^1\right)\\ \mathbf{H}_t^1&=\mathbf{LN}\left(\mathrm{SLF-ATT}\left(\mathbf{Q}_t^0,\mathbf{K}^0,\mathbf{V}^0\right)+\mathbf{H}_t^0\right)\\ \mathbf{Q}^0&=\mathbf{H}^0\mathbf{W}^Q\\ \mathbf{K}^0,\mathbf{V}^0&=\mathbf{H}^0\mathbf{W}^K,\mathbf{H}^0\mathbf{W}^V\end{aligned} \tag{2}$$

5.1.3 非自回归序列预测

在获取了句子的表征向量 $\mathbf{H}^L$ 后，理论上来说可以直接添加一个 softmax 层来预测结果：

$$\begin{aligned}\mathbf{s}_t&=\mathbf{h}_t^{\top}\mathbf{W}_s+\mathbf{b}_s\\ P_{\mathrm{dp}}\left(y_t\right)&=\mathrm{softmax}\left(\mathbf{s}_t\right)\end{aligned} \tag{3}$$

生成对于目标词 V 的概率分布 $P_{\mathrm{dp}}\left(y_t\right)$，选取概率最大的那一项作为最终的结果。尽管这种直接预测的方法在固定长度的语法纠错问题上很高效，但是它只能进行相同位置的替换操作，无法进行变长操作。因此论文提出使用 CRF 解决非自回归序列预测的这一问题。

在 CRF 框架下，给定输入序列 X，长度为 T' 的目标序列 Y 的概率为：

$$P_{\mathrm{crf}}(Y\mid X)=\frac{1}{Z(X)}\exp\left(\sum_{t=1}^{T'}s\left(y_t\right)+\sum_{t=2}^{T'}t\left(y_{t-1},y_t\right)\right) \tag{4}$$

其中 $Z(X)$ 是归一化因子，$s\left(y_t\right)$ 表示 y 在位置 t 的得分，值 $t\left(y_{t-1},y_t\right)=\mathbf{M}_{y_{t-1},y_t}$ 表示从 token y_{t-1} 到 y_t 的转移得分。同时，引入了低秩分解和截断维特比算法来加速计算，$\mathbf{M}\in\mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ 是转移矩阵，可以通过两个低维的神经参数矩阵 $\mathbf{E}_1,\mathbf{E}_2\in\mathbb{R}^{|\mathcal{V}|\times d_m}$ 来估计：

$$\mathbf{M}=\mathbf{E}_1\mathbf{E}_2^{\top} \tag{5}$$

5.1.4 焦点损失函数

同时使用直接预测和 CRF 的损失作为损失函数。此外为了解决传统最大似然对于不均匀的分类效果较差的问题，作者提出使用焦点损失函数。

$$\begin{aligned}\mathcal{L}_{\mathrm{dp}}^{\mathrm{f}}&=-\sum_{t=1}^{1}\left(1-P_{\mathrm{dp}}\left(\mathrm{y}_t\mid X\right)\right)^{\gamma}\log P_{\mathrm{dp}}\left(\mathrm{y}_t\mid X\right)\\ \mathcal{L}_{\mathrm{crf}}^{\mathrm{fl}}&=-\left(1-P_{\mathrm{crf}}(Y\mid X)\right)^{\gamma}\log P_{\mathrm{crf}}(Y\mid X)\end{aligned} \tag{6}$$

$$\mathcal{L}^{\mathrm{fl}}=\mathcal{L}_{\mathrm{dp}}^{\mathrm{f}}+\mathcal{L}_{\mathrm{crf}}^{\mathrm{fl}} \tag{7}$$

其中，γ 是一个用来控制惩罚权值的超参数。

5.2 复现的难点及细节

根据原论文的开源代码[1]完成复现。首先运行 train.py 进行训练，将训练过程的参数 ckpt 进行保存，再通过 test.py 生成语法纠正的句子对文件。最后，为和原论文保持一致，使用 eval.py 对生成语法纠正的句子对文件计算 F1 分数、准确率、召回率等。

关于模型参数的维度和数据处理。首先，读取数据，如论文所述，模型读入句子中的每个字符。一是构建 train_batch_text_list，大小为 (batch_size, 此 batch 内的最长句子长度)，为 list 类型。第一维是 batch 中每个句子，第二维是该句子的每个中文字符，为待改正的错句。二是构建 train_tag_matrix，大小为 (batch_size, 此 batch 内的最长句子长度)，为 numpy.ndarray 类型，第一维是 batch 中每个句子，第二维是该句子的每个中文字符在 vocab.txt 中对应的 index，为标准答案的正确句。三是构建 train_mask_matrix 大小为 (batch_size, 此 batch 内的最长句子长度)，为 numpy.ndarray 类型，第一维是 batch 中每个句子，第二维是 0 或 1，1 表示该字符有意义，0 表示该字符无意义。将三者输入 bert 进行 encoding，输出维度为 [seq_len, batch_size, embedding_size]。

接下来，将其通过一个线性层映射到词表，词表共有 16541 个词，即模型的分类数。原始错句经过 bert 的 encoder 表示，通过原文 2.4 公式 (3) 增加线性层，输出维度为 [seq_len, batch_size, num_class]。按 num_class 维做 softmax，每个数在 (0,1) 之间，并放大了差距，维度不变。

然后，将每个句子的 16541 维分类结果输入 CRF 层，分别使用 $loss_{dp}$ 和 $loss_{crf}$ 二者结合作为损失函数，输出 [seq_len, batch_size, num_class] 维

[1] https://github.com/lipiji/TtT

的分类结果，并取 top-k 的值映射到词表，作为生成的正确句输出。

由于数据集较大，将 batch_size 设置为 50 时，一个 epoch 大约要运行 1 个小时左右，故仅运行了较少的 epoch。

5.3　改进方向与创新点

数据预处理　在论文中，句子按字级别进行纠错，但这样使之缺少了词级别的特征，纠错结果常常出现单字错误，如图所示。在数据处理时，尝试首先进行分词，并以词为粒度进行增加、替换、删除。

loss function　在论文中的 DP loss function 直接将 BERT 输出的该句子特定对应分类的概率差作为 loss，并未考虑其他类别的预测概率，这样可能造成每个分类的概率都较大，所以，针对这个问题，改动 loss function，将全部的概率均引入 loss，改动后的 loss function 如公式所述。

$$
\begin{aligned}
L_{dp} &= \frac{1}{\mathrm{B}} \sum_{\text{sentence}=1}^{\mathrm{B}} (-\frac{1}{\mathrm{S}} \sum_{t=1}^{\mathrm{S}} (1 - P'_{dp})^{\gamma} \log P'_{dp}) \\
P'_{dp} &= \frac{p(y_t \mid x)}{\sum_{i \in \text{sentence}} p(y_i \mid x)}
\end{aligned}
\tag{8}
$$

其中 B 为 batch size，S 为句子长度 seq_length

eval　关于 NLG 模型评估方法，目前比较常用的包括 BLEU、METEOR、ROUGE、CIDEr 等。但针对于错误语法纠正的模型评估，若仅统计多元词组的出现个数，可能会引入冗余分数。对此，可以考虑采用字符串最大匹配算法，使用最大匹配分数评估模型。对于 gap 惩罚值和不同字符的惩罚分数，可通过学习得到。

数字的处理　发现论文中的数字未被处理，在文本生成时会出现”幻象“，故有必要对阿拉伯数字和其他为中文字母和符合进行单独处理，处理方法为保留原值即可。

5.4　实验设置与复现结果

数据集使用 HybridSet[14]。HybridSet 是根据基于 ASR[15] 和 OCR[16] 的结果构建的混淆集。该数据集是一个定长数据集, 包含 271329 个句子，最小长度 4，最大长度 140，平均长度 42.5，总错误数 381962，平均每句错误数 1.4。

HybridSet 总共包含约 27 万对样本，其语法错误修改均属固定长度。为了体现 TtT 模型在变长语法纠错场景下的表现情况，论文在 HybridSet 的基础上进行删除、插入和局部交换的操作以获取错误样本，最终总共得到 54 万个样本。

实验中使用预训练好的中文 BERT 来初始化模型。对于 CRF 层中估计转移矩阵，设置矩阵 E_1 和 E_2 的维度为 32。训练时 learning rate 为 $1e-5$, dropout rate 为 0.1，受限于机器硬件存储与计算能力，batch-size 设为 50，在数据集 HybridSet[14] 上训练 10 个 epoch。

TtT 在 HybridSet 运行 10 个 epoch 的过程中, 验证集上字级别的 F1 分数值如图11所示，可以看到 F1 分数在前几个 epoch 上有着明显的提升，在后几个 epoch 上提高不大，逐渐收敛。根据论文所述，如果条件允许，可以设置更大的 epoch 来获取更好的结果。

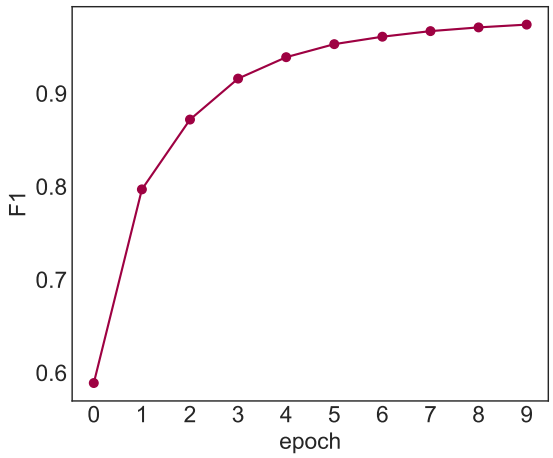

图 11　TtT 在 HybridSet 上不同 epoch 的 F1 分数

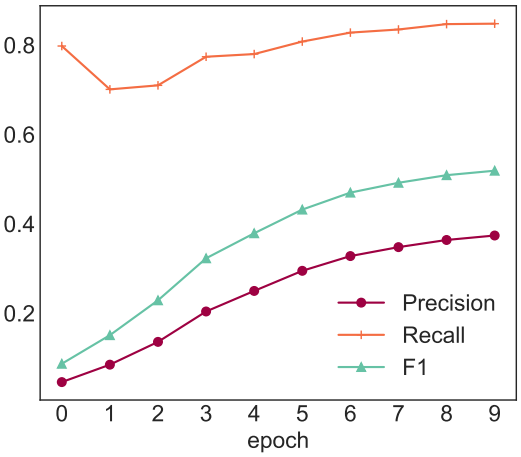

图 12　错误检测的正确率、召回率和 F1 分数

在测试集上，为了更好地对模型进行评测，分

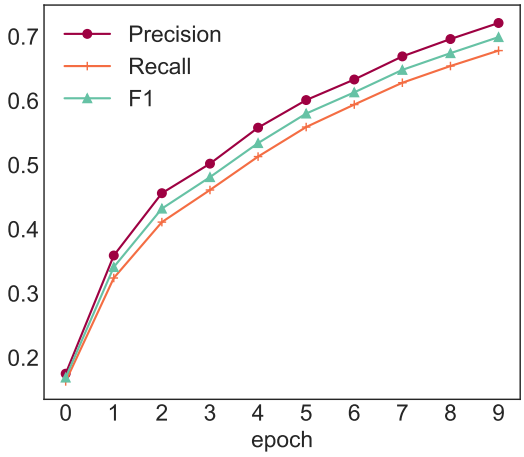

图 13 错误纠正的正确率、召回率和 F1 分数

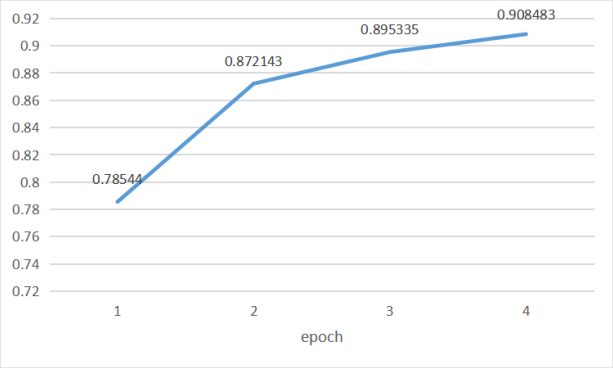

图 14 训练过程中验证集上的 F1 分数

为错误检测（给定句子中不正确字符的位置）和错误纠正（错误字符的位置和对应的纠正）两部分。在训练过程中这两部分的正确率、召回率和 F1 分数分别如图12和图13所示，可以看到训练过程中在测试集上各项指标都在不断提高且没有收敛的迹象，也就是说还能通过设置更多 epoch 继续训练来提高指标，但受限于硬件条件无法训练达到最优结果。

5.5 创新点实现与实验结果

数据预处理 将数据经过分词输出，调用 jieba 库进行实现，随之要重新生成词表。其训练过程中验证集上的 F1 分数变化如图14所示. 最终在 test 集上的 f1 分数与原始算法的比较如表1所示, 由于时间所限，仅训练了 5 个 epoch。发现结果不太理想，这说明有必要同时考虑字粒度和词粒度二者的结合，而单独的模型字粒度要更胜一筹。同时，考虑到训练 epoch 比较少模型并未收敛，最终模型的结果也有待进一步验证。此外，在训练过程中发现，由于 token 变少，词粒度的模型训练速度明显快于字粒度的模型训练速度。

表 1 不同粒度的数据处理的 F1 score 对比

	TtT ori	TtT wordSeg
detection F1 score	0.376	0.234
correctoion F1 score	0.520	0.433

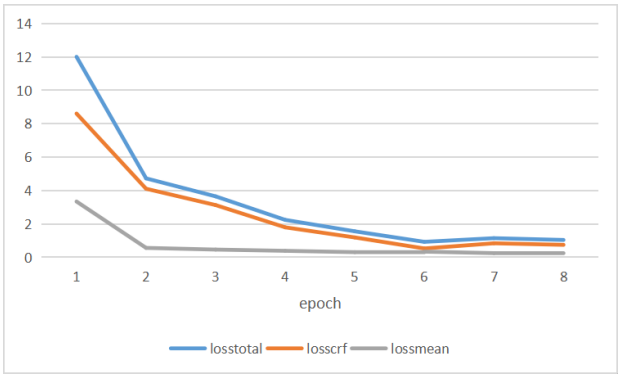

图 15 训练过程中 loss 函数变化

loss function 重新编写 loss 函数部分，具体参见代码。训练过程的 loss 函数变化如图15所示, 可以看到其正常收敛，最终在 test 集上的 f1 分数与原始算法的比较如表2所示。由于时间所限，仅训练了 5 个 epoch。在改正任务中，f1 分数略微增加，由于同时考虑降低其他的预测概率，这是符合预期的。

表 2 不同粒度的数据处理的 F1 score 对比

	TtT ori	TtT mean loss
detection F1 score	0.376	0.373
correctoion F1 score	0.520	0.528

eval 字符串最大匹配算法，使用动态规划算法实现，对于两个字符串 X 和 Y，设其长度为 n 和 m，则 DP 算法的递推函数为：

$$MAX\_MATCH(i,j) =$$

$$\begin{cases} j\delta & \text{if i = 0} \\ \min\begin{cases}\alpha_{x_i y_j} + MAX\_MATCH(i-1,j-1) \\ \delta + MAX\_MATCH(i-1,j)\end{cases} & \\ \delta + MAX\_MATCH(i,j-1) & \text{otherwise} \\ i\delta & \text{if j = 0} \end{cases} \tag{9}$$

其中，δ 为空值惩罚值，$\alpha_{x_i y_j}$ 为 x_i 与 y_j 不同的惩罚值，$MAX\_MATCH(i,j)$ 为 X 的第 i 个字与 Y 的第 j 个字之前的字符串的最大匹配值（MAX_MATCH），$MAX\_MATCH(n,m)$ 即为最终结果，为适应问题，

使用 $1 - MAX\_MATCH(n, m)$ 作为最终的评价指标。对如下不同的句子，取 $\delta = \alpha_{x_i y_j} = 1/seq\_length$，计算得到的分数如表3所示。

- 答案：今天天气真好
- sentence1：今天天气你好
- sentence2：今天今天天天天气气气真好好好好

表 3　BLEU 与 OPT 的 eval 值对比

	BLEU	1-MAX_MATCH
sentence 1	0.833	0.833
sentence 2	1	0.4

可以发现，最初的 BLEU 无法判断出句子 2 的错误，虽然后续发展了很多改进方法，但 MAX_MATCH 方法是十分简单和直观的，并且十分适用于文本纠错这类答案与原句相似度较高的场景。时间所限，还有还多细节有待商讨，未来将进一步细化和丰富。

6　Encode, Tag, Realize: High-Precision Text Editing 论文复现

6.1　相关工作

最近的工作讨论了学习神经解码器进行文本生成的一些困难。传统的 seq2seq 方法需要大量的训练数据，难以控制且难以约束到所需的输出。同时，许多看起来像是完整的文本生成的 NLP 任务都是使用简单方法的自然测试平台。

文本简化（Text Simplification）

文本简化是一种释义任务，已知可以从建模编辑操作中受益。这种类型的简单例子是句子压缩系统，该系统在字符/短语级别应用 drop 操作，而更复杂的系统也应用拆分，重新排序和词法替换。另外，该任务还尝试了针对基于短语的 MT 开发的系统以及神经编码器-解码器模型进行简化。

这个任务很适用 edit operations modeling 的方法解决。Dong[17] 等人提出了一个文本编辑模型，类似于本文，主要差异是：

- 使用了 interpreter module 语言模型，来实现本文 Realize 部分的功能
- 使用了 full vocabulary 来生成 added tokens，而本文使用了 optimized set of frequently added phrases 集合

虽然 Dong 的模型生成更多样化的输出，但它可能会在推理时间、精度和数据效率上产生负面影响。另外一个跟本文相似的论文为 Gu[18] 等人提出的 Levenshtein Transformer 模型，通过 sequence of deletion and insertion actions 来生成文本。

单文档摘要 (Single-document summarization)

单文档摘要是一项在要求系统保留含义不变的条件下缩短文本的任务。它已在字符级别和句子级别上使用基于删除的方法进行了处理。其他论文也使用神经编码器-解码器方法进行生产摘要，该摘要不仅可以删除，还可以进行编辑。

这个任务可以使用在 token-level 和 sentence-level 上的 deletion-based 方法解决。也有论文使用了 seq2seq 做抽象摘要，但其缺陷是产生的操作不仅仅是删除。因此，Jing 和 McKeown(2000)[19] 的工作使用还原、组合、句法转换、词汇释义、泛化和重新排序操作解决。也有论文使用 copy 机制来使模型更容易复制 source 端词汇。

语法错误纠正 (Grammatical Error Correction)

在语法错误纠正中，系统通常由语言学习者编写，并负责检测和修复语法错误（以及其他错误）。完成此任务的方法通常包含特定于任务的知识，例如，通过针对特定错误类型设计分类器，而无需手动标记数据即可对其进行训练，或者通过调整统计机器翻译来进行方法。错误检测子问题的方法在本质上与句子压缩系统相似，因为它们被实现为基于单词的神经序列标记器。神经编码器-解码器方法也通常用于纠错任务，但是由于缺乏训练数据，这就是为什么需要针对特定任务的技巧。

6.2　问题定义

在一些文本生成任务中，例如最近引入的句子分裂和句子融合任务，输出文本与输入高度重叠。在这个设置中，学习一个 seq2seq 模型来从头开始生成输出文本直觉上似乎是浪费。复制机制 (Guet al.,2016[20]；See et al.,2017[21]) 允许在复制源记号和生成任意记号之间进行选择，但是尽管这种混合模型有助于词汇外的单词，但是它们仍然需要大的训练集，因为它们依赖于与标准 seq2seq 方法所使用的一样大的输出词汇。

相比之下，我们建议学习一个文本编辑模型，它对输入序列应用一组编辑操作来重建输出。我们表明，通常使用相对较小的一组表示文本删除、重新措辞和单词重新排序的输出标签就足以再现训

练数据中的大部分目标。这导致学习问题，词汇量小得多，输出长度固定为源文本中的单词数。这反过来大大减少了训练精确模型所需的训练样本的数量，这在只有少量人为标记的数据可用的应用中尤其重要。

6.3 方法

6.3.1 序列标注任务

本文使用的方法是将 text edit 转换为序列标注的问题，主要包含 3 个部分：

1. 定义 tagging operation
2. 优化 Phrase Vocabulary
3. 将训练数据的 plain-text target 转换为 tagging 格式
4. 将 tag 转变为输出文本

1. 标记操作

基本定义:

我们的标记器为每个输入字符分配一个标记。标签由两部分组成：基本标签和添加短语。基本标签是 keep 或 delete，它指示是否在输出中保留字符。Tag：base tag(KEEP/DELETE) + added phrase（表示在 token 前需要增加 phrase，可以为空）。添加短语 P（可以为空）强制将 P 添加到相应的字符之前。P 属于词汇 V，词汇 V 定义了一组单词和短语，可以将这些单词和短语插入输入序列以将其转换为输出。

基本标签 B 和添加短语 P 的组合被视为单个标签，并用 PB 表示。唯一标签的总数等于基本标签的数量乘以短语词汇量的大小，因此一共有 2|V| 个唯一标签。。

不同任务可以有 task-specific 的 tag，比如在 sentence fusion 任务中可以把 SWAP 标记添加

到第一句话的最后部分（如图16所示）；还比如为了用代词替换命名实体，可以用 PRONOMINALIZE 标签，在 realize 阶段通过查找知识库中命名实体的 gender 信息来用 she、he、they 替换（这比用 she^DELETE, he^DELETE, they^DELETE 要好）

Source: Dylan won Nobel prize . Dylan is an American musician .
Tags: DELETE KEEP KEEP KEEP SWAP KEEP ^{comma}DELETE KEEP KEEP KEEP ^{comma}DELETE
Realization: Dylan , an American musician , won Nobel prize .

图 16

2. 优化短语词汇

短语词汇表包含可以在源词之间添加的短语。一方面，我们希望最小化短语的数量，以使输出标签的词汇量保持较小。另一方面，我们希望使用可用的标记操作将可以从源重构的目标文本的百分比最大化。这导致以下组合优化问题。

问题 1。

给定短语集 $A_1, A_2, ..., A_m$ 的集合，其中 $A_i \subseteq P$ 且 P 是所有候选短语的集合，选择最多 l 个短语（即 $|V| \leqslant l$）的词汇，以使覆盖短语集的数量最大化。当且仅当 A_i 时，才会覆盖短语集 A_i。

这个问题与最小的 k-union 问题紧密相关，后者是 NP-hard。后一个问题要求一组 k 个词组，以便其并集的基数最小。如果我们能够在多项式时间内解决问题 1，我们也可以通过找到最小短语词汇量 l 使得覆盖的短语集数量至少为 k 来解决多项式时间内的最小 k 联合问题。最小 k 联合问题的减少为我们提供了以下结果：

定理 1。问题 1 是 NP-hard 的。

为了确定要包含在词汇表中的候选短语，我们首先将训练数据中的每个源文本 s 与目标文本 t 对齐。这是通过计算两个单词序列之间最长的公共子序列（LCS）来实现的，可以使用时间复杂度为 O(|S|*|T|) 动态规划来完成。目标文本中不属于 LCS 的 n-gram 是短语，必须包含在短语词汇中才能从 s 构造 t。 在实践中，短语词汇应由经常添加到目标的短语组成。因此，我们采用以下简单方法来构建短语词汇表：按照短语出现的短语集的数量对短语进行排序，并选择 l 个最常用的短语。这被发现可以基于手动检查产生有意义的短语词汇，例如，句子融合的主要短语包括许多语篇连接词。

我们还考虑了一种贪婪的方法，该方法一次构造一个词汇，始终选择具有最大增量覆盖的词汇。但是，这种方法对于我们的用例而言并不理想，因为某些频繁使用的短语（例如"(" 和")"）是紧密耦合的。仅选择 "(" 的增量覆盖率几乎为零，但与 ")" 一起使用时，它们可以覆盖许多示例。

3. 将训练目标转换为标签

确定短语词汇后，我们可以将训练数据中的目标文本转换为标签序列。给定短语词汇，我们无需计算 LCS，但可以利用更有效的方法，该方法遍历输入中的单词，并贪婪地尝试将它们与目标中的单词进行匹配，如果存在无匹配，则对词汇 V 中的短语进行匹配。这可以在 O(|s|×n_p) 时间完成，其中 n_p 是 V 中最长短语的长度，如算法所示。 需

要添加词汇表 V 中不存在的短语的训练目标不会转换为标签序列，而是会被过滤掉。在缩小训练数据集的同时，这也可能有效地过滤出质量低下的目标。请注意，即使无法使用输出标签词汇从输入中重构目标，我们的方法仍可能会使用可用短语生成合理的输出。例如，目标可能需要使用不频繁的":" 字符，这不在我们的词汇表中，而是模型可以选择预测一个更常见的"," 字符。

具体算法如下：

Algorithm 1 Converting a target string to tags.

Input: Source text $s = [s(1), \ldots, s(n_s)]$, target text $t = [t(1), \ldots, t(n_t)]$, phrase vocabulary V, and the maximum added phrase length n_p.
Output: Tag sequence x of length n_s or of length 0 if conversion is not possible.

$x(i) = \texttt{DELETE}, \quad \forall i = 1, \ldots, n_s$ ▷ Initialize tags.
$i_s = 1$ ▷ Current source word index.
$i_t = 1$ ▷ Current target word index.
while $i_t \leq n_t$ **do**
  if $i_s > n_s$ **then**
    return [] ▷ Conversion infeasibile.
  if $s(i_s) == t(i_t)$ **then**
    $x(i_s) = \texttt{KEEP}$
    $i_t = i_t + 1$
  else
    $p = []$ ▷ Added phrase (word sequence).
    match_found $= 0$
    for $j = 1, \ldots, n_p$ **do**
      p.append$(t(i_t + j - 1))$
      if $s(i_s) == t(i_t + j)$ **and** $p \in V$ **then**
        match_found $= 1$
        break
    if match_found **then**
      $x(i_s) = {}^{p}\texttt{KEEP}$
      $i_t = i_t + |p| + 1$
  $i_s = i_s + 1$
return x ▷ Target has been consumed, so return tags.

图 17

4. 标记转输出文本

获得预测的标签序列后，我们将其转换为文本（“实现”步骤）。虽然经典的文本生成工作在计划和实现之间进行了区分，但端到端的神经方法通常会忽略这种区分，只有少数工作例外。

对于保留，删除和添加的基本标记操作，实现是一个简单的过程。此外，我们调整句子边界的大小写。如果我们引入特殊的标记，例如 3.1 节中提到的 PRONOMINALIZE，则实现将变得更加复杂。对于此标签，我们需要从知识库中查找被标签实体的性别。拥有单独的实现步骤是有益的，因为我们可以仅在对适当代词有信心的情况下才决定对代词进行名词化，否则可以使实体提及保持不变。

6.3.2 模型整体框架

标记器：

标记器由两个部分组成：一个编码器，它为输入序列中的每个元素生成激活向量；一个解码器，将编码器的激活转换为标签标记。

编码器

我们选择 BERT Transformer 模型作为我们的编码器，因为它展示了许多句子编码任务的最新结果。我们使用基于 BERT 的体系结构，该体系结构包含 12 个自我注意力层。

解码器

在原始 BERT 论文中，简单的解码机制用于序列标记：通过在编码器 logits 上应用 argmax，可在单次前馈过程中生成输出标记。这样，无需对序列中标签之间的依赖关系进行建模，即可独立预测每个输出标签。当应用于 BERT 编码器时，这种简单的解码器展示了命名实体识别任务的最新结果。

为了更好地建模输出标签之间的依赖关系，我们提出了一种更强大的自回归解码器。具体来说，我们在 BERT 编码器的顶部运行一个单层 Transformer 解码器（参见图18）。在每个步骤中，解码器都会使用先前预测的标签的嵌入以及来自编码器的激活。

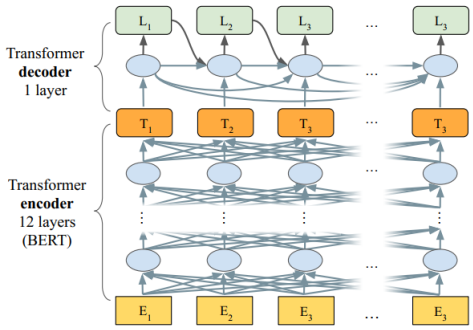

Figure 3: The architecture of LaserTagger$_{AR}$.

图 18

6.3.3 复现难点

一方面是趋动平台的使用不是很熟练，另一方面，配置实验环境的时候总是会报一些奇怪的错误，需要及时在网上寻找解决方法。整体实现过程在论文和 GitHub 的项目中有比较详细的指导，按部就班即可。

6.3.4 实现细节

使用 LaserTagger 进行实验包括以下步骤:

1. 优化可由 LaserTagger 添加的短语的词汇。
2. 将目标文本转换成目标标签序列。
3. 微调预训练的 BERT 模型以预测标签。

4. 计算预测。

5. 评估预测。

1. 短语词汇优化

下载 [wiki split](https://github . com/Google-research-datasets/wiki-split) 数据集，并运行下面的命令来查找该模型的一组短语允许添加。

export WIKISPLIT_IR=/path/to/wikisplit
export OUTPUT_DIR=/path/to/output
python phrase_vocabulary_optimization.py
–input_file=$WIKISPLIT_DIR/train.tsv
–input_format=wikisplit
–vocabulary_size=500
–max_input_examples=1000000
–output_file=$OUTPUT_DIR/label_map.txt

请注意，也可以将“max_input_examples”设置为较小的值，以获得合理的词汇表，但是应该对数据集行进行排序。这些行是按字母顺序排列的，所以取其中的前 k 行可能不会给你一个有代表性的数据样本。

2. 将目标文本转换为标签

下载预训练的 BERT 模型 (https://github.com/Google-research/Bert 、)。在所有的实验中，我们都使用了 12 层的“BERT-Base, Cased”模型。然后将原始 TSV 数据集转换成 TFRecord 格式。

export BERT_BASE_DIR=/path/to/cased_L-12_H-768_A-12
python preprocess_main.py
–input_file=$WIKISPLIT_DIR/tune.tsv
–input_format=wikisplit
–output_tfrecord=$OUTPUT_DIR/tune.tf_record
–label_map_file=$OUTPUT_DIR/label_map.txt
–vocab_file=$BERT_BASE_DIR/vocab.txt
–output_arbitrary_targets_for_infeasible_examples=true
python preprocess_main.py
–input_file=$WIKISPLIT_DIR/train.tsv
–input_format=wikisplit
–output_tfrecord=$OUTPUT_DIR/train.tf_record
–label_map_file=$OUTPUT_DIR/label_map.txt
–vocab_file=$BERT_BASE_DIR/vocab.txt
–output_arbitrary_targets_for_infeasible_examples=false

3. 模型训练

模型超参数在 [lasertagger_config.JSON](configs/laser tagger_config.JSON) 中指定。此配置文件扩展了“bert_config.json”，它带有压缩的预训练 bert 模型。

注意，如果想从使用 LaserTagger_FF 切换到 LaserTagger_AR，应该设置”use_t2t_decoder：true”。后者通常更多准确，而前者运行推理更快。

在第二步创建的.num_examples 文件中检查以下参数

export NUM_TRAIN_EXAMPLES=310922
export NUM_EVAL_EXAMPLES=5000
export CONFIG_FILE=configs/lasertagger_config.json
export EXPERIMENT=wikisplit_experiment_name
python run_lasertagger.py
–training_file=$OUTPUT_DIR/train.tf_record
–eval_file=$OUTPUT_DIR/tune.tf_record
–label_map_file=$OUTPUT$_{DIR}$/*label_map.txt*
–model_config_file=$CONFIG$_{FILE}$
–output_dir=$OUTPUT_DIR/models/$EXPERIMENT
–init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt
–do_train=true
–do_eval=true
–train_batch_size=256
–save_checkpoints_steps=500
–num_train_examples=$NUM_TRAIN_EXAMPLES
–num_eval_examples=$NUM_EVAL_EXAMPLES

4. 计算预测。

首先，需要导出模型。

python run_lasertagger.py
–label_map_file=$OUTPUT$_{DIR}$/*label_map.txt*
–model_config_file=$CONFIG_FILE
–output_dir=$OUTPUT_DIR/models/$EXPERIMENT
–do_export=true
–export_path=$OUTPUT_DIR/models/$ EXPERIMENT/export

5. 计算评估分数。

python score_main.py

–prediction_file=$PREDICTION_FILE

6.4 实验

实验数据集：分隔复述任务：[wiki split 数据集](https://github.com/Google-research-datasets/wiki-split)

实验设置：如实现细节所述

复现结果：分割复述如图19所示

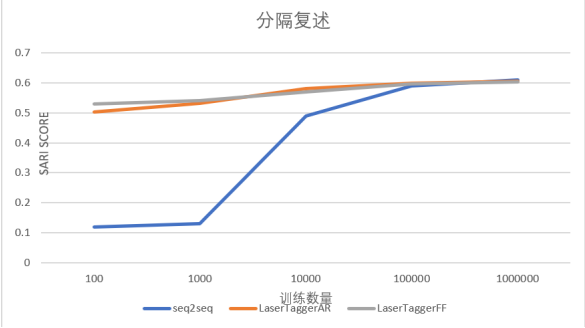

图 19

Model	P	R	$F_{0.5}$
seq2seq	6.15	15.32	7.01
LaserTagger(FF)	43.25	23.83	38.82
LaserTagger(AR)	46.53	24.85	40.58

表 4　GEC 性能评估

我们在分割复述任务以及语法纠错任务上进行了测试。

实验结果表明 LaserTagger 的表现与强大的基于 BERT 的 seq2seq 基线相当，而后者使用了大量的训练示例；并且当训练示例的数量有限时，LaserTagger 的性能显然优于后者。

7 总结与展望

本文针对中文和英文的文本检测和纠错问题。首先，进行了调研。然后，复现了三篇影响较大的论文，并提出了创新的方法，对实验结果进行了分析。接下来，将在已有实验结果的基础上，对创新方法做进一步的改进与实验。

参考文献

[1] GE T, WEI F, ZHOU M. Fluency boost learning and inference for neural grammatical error correction[C]//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). [S.l.: s.n.], 2018: 1055-1065.

[2] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

[3] ZHAO W, WANG L, SHEN K, et al. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data[J]. arXiv preprint arXiv:1903.00138, 2019.

[4] SUN X, GE T, WEI F, et al. Instantaneous grammatical error correction with shallow aggressive decoding[J]. arXiv preprint arXiv:2106.04970, 2021.

[5] ALIKANIOTIS D, RAHEJA V. The unreasonable effectiveness of transformer language models in grammatical error correction[J]. arXiv preprint arXiv:1906.01733, 2019.

[6] WANG H, KUROSAWA M, KATSUMATA S, et al. Chinese grammatical correction using bert-based pre-trained model[J]. arXiv preprint arXiv:2011.02093, 2020.

[7] KANEKO M, MITA M, KIYONO S, et al. Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction[J]. arXiv preprint arXiv:2005.00987, 2020.

[8] AWASTHI A, SARAWAGI S, GOYAL R, et al. Parallel iterative edit models for local sequence transduction[J]. arXiv preprint arXiv:1910.02893, 2019.

[9] OMELIANCHUK K, ATRASEVYCH V, CHERNODUB A, et al. Gector–grammatical error correction: tag, not rewrite[J]. arXiv preprint arXiv:2005.12592, 2020.

[10] LI P, SHI S. Tail-to-tail non-autoregressive sequence prediction for chinese grammatical error correction[C]//Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). [S.l.: s.n.], 2021: 4973-4984.

[11] DEVLIN J, CHANG M W, LEE K, et al. BERT: Pre-training of deep bidirectional transformers for language understanding[C/OL]// Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, 2019: 4171-4186. https://aclanthology.org/N19-1423. DOI: 10.18653/v1/N19-1423.

[12] LAFFERTY J D, MCCALLUM A, PEREIRA F C N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data[C]//ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001: 282–289.

[13] LIN T Y, GOYAL P, GIRSHICK R, et al. Focal loss for dense object detection[J/OL]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 42(2):318-327. DOI: 10.1109/TPAMI.2018.2858826.

[14] WANG D, SONG Y, LI J, et al. A hybrid approach to automatic corpus generation for Chinese spelling check[C/OL]//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels, Belgium: Association for Computational Linguistics, 2018: 2517-2527. https://aclanthology.org/D18-1273. DOI:

10.18653/v1/D18-1273.

[15] YU D, DENG L. Automatic speech recognition: volume 1[M]. [S.l.]: Springer, 2016.

[16] TONG X, EVANS D A. A statistical approach to automatic ocr error correction in context[C]//Fourth workshop on very large corpora. [S.l.: s.n.], 1996.

[17] YUE DONG M R, Zichao Li, CHEUNG J C K. Editnts: An neural programmer-interpreter model for sentence simplification through explicit editing.[C]//In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics). [S.l.: s.n.], 2019.

[18] JIATAO GU C W, ZHAO. J. Levenshtein transformer[C]//[S.l.: s.n.], 2019.

[19] JING H, MCKEOWN K. Cut and paste based text summarization[C]// In 1st Meeting of the North American Chapter of the Association for Computational Linguistics. [S.l.: s.n.], 2000.

[20] JIATAO GU H L, Zhengdong Lu, LI. V O. Incorporating copying mechanism in sequence-to-sequence learning.[C]//In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). [S.l.: s.n.], 2016: 1631–1640.

[21] ABIGAIL SEE P J L, MANNING. C D. Get to the point: Summarization with pointergenerator networks.[C]//In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),. [S.l.: s.n.], 2017: 1073–1083.

附录 A 运行结果截图

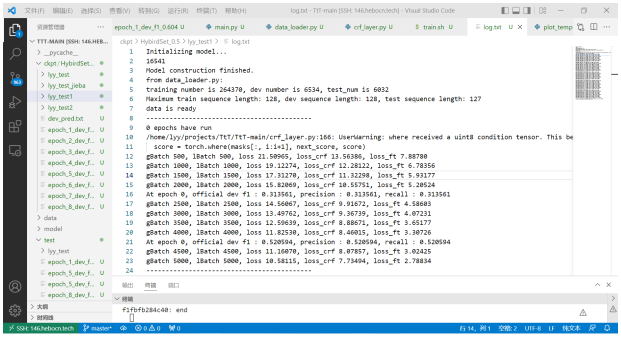

图 20 TtT 实验过程截图

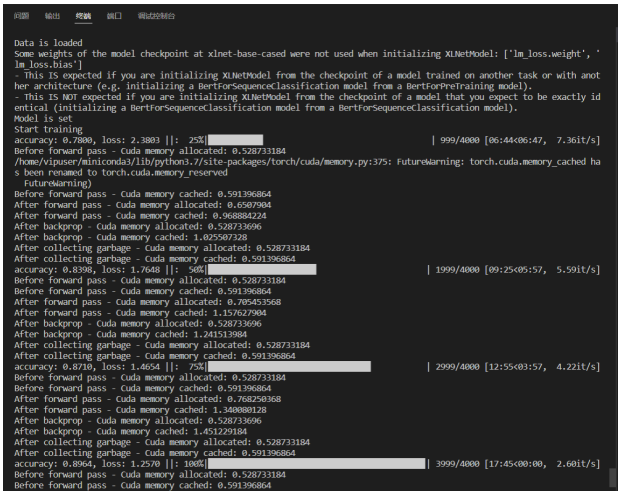

图 21 训练过程终端截图

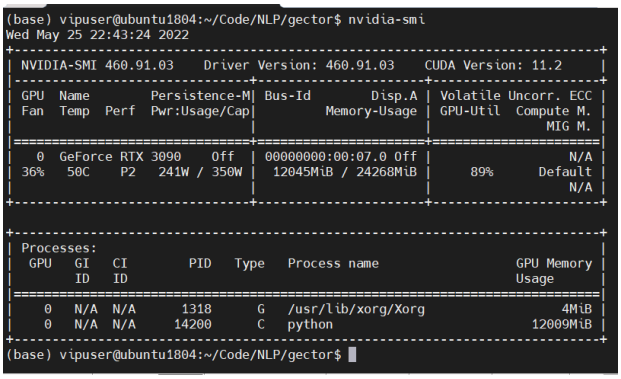

图 22 训练过程 GPU 使用情况截图

Yuying Lin, 学号: 2012174, 负责复现《Tail-to-Tail Non-Autoregressive Sequence Prediction for Chinese Grammatical Error Correction》

Rongqi Xu, 学号: 1911281, 负责复现《Cross-Sentence Grammatical Error Correction》

Yixuan Huang, 学号: 2012067, 负责复现《GECToR –Grammatical Error Correction: Tag, Not Rewrite》

Yuying Lin, 学号: 2012174, 负责复现《Tail-to-Tail Non-

Autoregressive Sequence Prediction for Chinese Grammatical Error Correction》

Rongqi Xu, 学号: 1911281, 负责复现《Cross-Sentence Grammatical Error Correction》

Yixuan Huang, 学号: 2012067, 负责复现《GECToR –Grammatical Error Correction: Tag, Not Rewrite》