计算机科学
COMPUTER SCIENCE

基于自然语言句法信息的正则表达式生成

王昊, 吴军华

引用本文

王昊, 吴军华. 基于自然语言句法信息的正则表达式生成[J]. 计算机科学, 2024, 51(11A): 231200017-6.

WANG Hao , WU Junhua. Regular Expression Generation Based on Natural Language Syntax Information [J]. Computer Science, 2024, 51(11A): 231200017-6.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231200017

基于自然语言句法信息的正则表达式生成

王 昊 吴军华
南京工业大学计算机与信息工程学院 南京 211816
(17660457668@163.com)

摘 要 正则表达式由一系列字符和元字符组成,定义了一种匹配规则,可以用来检查一个字符串是否与所需的模式匹配。在软件开发过程中,很多开发人员发现编写正则表达式较为困难。因此,根据自然语言需求描述生成正则表达式成为研究热点。近年来,将自然语言描述转化为正则表达式的系统取得了一些研究成果,但往往只针对简单的序列化文本。探讨了将自然语言查询转化为可以执行其功能的正则表达式的方法。鉴于自然语言处理中句法解析的成功应用,模型使用自然语言的结构信息,以分层聚合的方式对语法解析树进行嵌入,并使用适用于输入树结构的 Tree-transformer 架构对自然语言描述进行自注意编码。解码器使用交叉注意力来预测正则表达式。在两个公共数据集上对模型进行了验证。实验证明,所提模型有效地提高了生成的正则表达式的质量,并在 DFA-Equal-Acc 评估指标中优于现有模型。

关键词:正则表达式生成;Tree-Transformer;句法解析
中图分类号 TP391

Regular Expression Generation Based on Natural Language Syntax Information

WANG Hao and WU Junhua
College of Computer and Information Engineering,Nanjing Tech University,Nanjing 211816,China

Abstract Regular expressions are composed of a series of characters and metacharacters,defining a matching pattern that can be used to check whether a string matches the desired criteria. Many developers find it is difficlult to write regular expressions during the software development process. Therefore,generating regular expressions based on natural language requirements has become a research focus. In recent years,systems that transform natural language descriptions into regular expressions have achieved some research results,but often only for simple serialized texts. This paper explores methods for converting natural language queries into regular expressions that can execute their intended functionality. Given the successful application of syntactic parsing in natural language processing,our model utilizes the structural information of natural language by embedding syntax parse trees in a hierarchically aggregated manner. We employ the Tree-transformer architecture,suitable for input tree structures,to perform self-attention encoding on natural language descriptions. The decoder uses cross-attention to predict the regular expression. The model is validated on two public datasets. Experimental results demonstrate that our model effectively improves the quality of generated regular expressions. It outperforms existing models in the DFA-Equal-Acc evaluation metric.

Keywords Regular expression generation,Tree-Transformer,Syntactic parsing

1 引言

将自然语言描述转化为计算机可执行的程序是计算机语言学中非常重要的问题。对于一些任务来说,即使是有经验的程序员,根据自然语言需求描述编写程序也可能会耗费很多时间,而且容易出错。从自然语言描述中自动生成程序有助于缓解上述问题,对最终用户和开发人员都有益。

早期的方法基于规则,其中识别了自然语言中的语法模式,并使用手工制定的规则将其映射到代码中。这些研究通常集中在领域特定语言(Domain Specific Language,DSL)的生成。Milosavljević等[1]提出了一种用于数据库的 Web 应用程序架构,遵循了将 JavaBean 组件映射到数据库模式的简单规则,生成组件和一套标准化的 JSP 页面。Zettlemoyer 等[2]提出了 PCCG 算法,用于将自然语言映射到 λ 演算。这些方法通常使用手动定义的规则,仅适用于生成具有许多约束的领域特定逻辑。

鉴于深度学习在神经机器翻译方面的成就,研究人员

基金项目:江苏省高等学校教育技术研究会高等教育信息化研究课题重点课题(2021JSETKT023)
This work was supported by the Key Research Project on Higher Education Informatization of the Higher Education Technology Research Association in Jiangsu Province(2021JSETKT023).
通信作者:吴军华(wujh@njtech.edu.cn)

已经开始使用深度学习技术来生成代码。Hindle[3]指出，代码与自然语言一样，是可重复、规律性和可预测的，这为使用深度学习生成代码奠定了基础。受神经机器翻译任务的启发，最近的研究已经使用基于输出的语法方法来解决不同领域的任务，例如正则表达式[4]、Python 程序[5]和 SQL 语句[6]。

本文专注于从自然语言中生成正则表达式。正则表达式是一系列字符，用于定义特定的匹配规则，通常用于确定字符串的格式或从中提取内容。到目前为止，正则表达式已广泛用于各种操作系统(如 Windows 和 Linux)，并得到大多数编程语言的支持(如 PHP，C++，Java 和 Python)。然而，对于那些对正则表达式知识有限或需要处理复杂表达式的人来说，编写准确的正则表达式并不容易，编写不正确的正则表达式可能导致在实际应用中出现意外，因为正则表达式之间的微小差异可能导致它们表示完全不同的字符串集合。因此，研究人员已经开始开发可以从人类提供的自然语言描述中生成正则表达式的模型。这些模型可以降低由不正确的正则表达式引起的错误可能性[7]。

Locascio 等[4]设计了基于序列到序列[8]模型的 Deep-Regex，在学习阶段使用了较少的领域知识，但仍然能够准确地从自然语言中预测正则表达式。Zhong 等[9]提出了一种基于逻辑的语义表示方法，将自然语言规范表示为逻辑表达式并将其转化为正则表达式。Park 等[10]在 Seq2Seq 的编码器-解码器模型中使用 LSTM 单元，编码器根据给定的自然语言描述生成隐藏向量，同时解码器接收来自编码器的隐藏向量并产生输出。然而，这些研究将自然语言视为一个简单输入序列。实际上，自然语言还包含丰富的结构信息。

为了更好地让模型学习自然语言的意图，引入了自然语言的句法结构信息。首先，使用句法分析工具对数据集进行预处理，以获取句法成分和词性标记。然后，在模型的编码部分设计了一个语法树编码模型，使用注意力机制从编码信息中提取特征。编码信息综合了自然语言序列和语法信息的表示，然后输入到后续模型中生成正则表达式。通过调整超参数形成的最佳模型在各个数据集上的表现皆优于现有的模型，并通过消融实验证明了模型中组件的有效性。

2 正则表达式生成模型

标准的 Transformer 编码器并不是为处理树状结构而设计的，因此研究人员尝试开发专门用于处理这种结构的神经网络，例如用于程序翻译的 Tree2tree[11]模型和用于实现对话任务的 Tree2Sequence[12]。然而，这些网络采用了循环机制，无法像 Transformer 那样实现并行处理。本文设计了 Tree_Transformer 作为编码器-解码器，用于处理语法树。模型首先对自然语言进行语法分析，将其转化为语法树的形式。然后，将这个语法树的节点按层次逐步聚合，并输入到自注意力编码器中进行编码。编码后的节点被传递到解码器中，用于预测生成正则表达式的字符序列。整个模型的框架如图 1 所示。

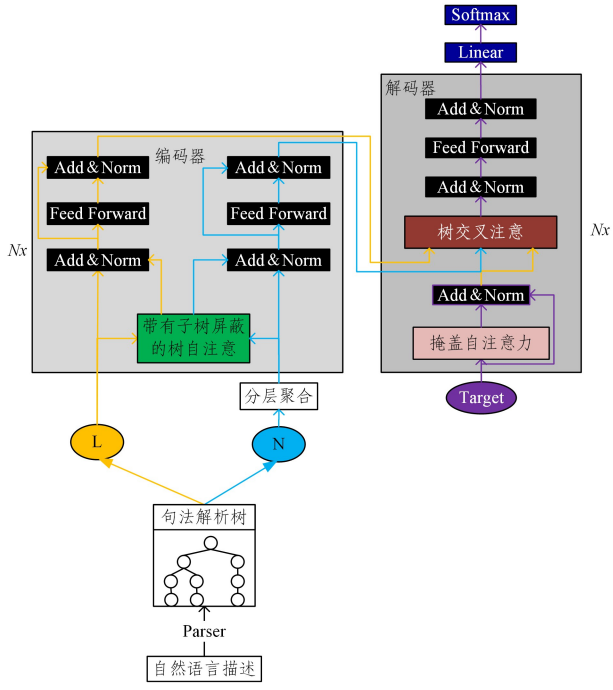


图 1 模型框架图

Fig. 1 Model framework diagram

2.1 句法解析树

自然语言指的是人类日常交流和互动所使用的语言系统，是由语法、词汇和语音的组合构建而成的复杂语言体系。在自然语言中，词汇是构建句子和表达含义的基本单元。语法是描述语言的结构和组织的规则系统，定义了语言中合法陈述或句子的形式和结构，并规定了如何将这些陈述组合成更复杂的结构。在以前的研究中，许多研究人员认为神经网络可以很好地学习自然语言的序列形式，而不考虑语法结构信息[14]。然而，后来的研究表明，将句法信息纳入的机器翻译模型[15]优于一般的序列到序列模型。

事实上，自然语言中的单词之间存在一种结构性关系，这种关系是由语法构成的，而不仅仅是表面上的简单顺序关系。通常，自然语言的底层结构可以用树状结构来表示。通过使用斯坦福大学提供的开源语法分析工具，即斯坦福分析器(Stanford Parser)[13]，可以获得自然语言句法结构的树形表示，如图 2 所示，其中的短语类型或词性如表 1 所列。

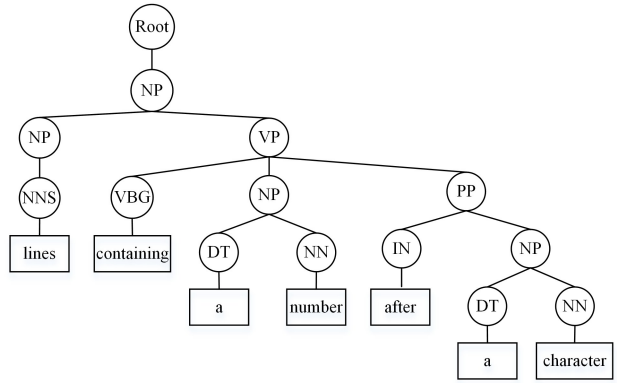


图 2 句法树示例

Fig. 2 Syntax tree example

表 1　短语类型或词性参考表

Table 1　Phrase types or parts of speech

缩写	短语类型或词性
NP	名词短语
VP	动词短语
NNS	名词复数形式
VBG	动词的现在分词
PP	介词短语
DT	限定词
NN	名词
IN	介词

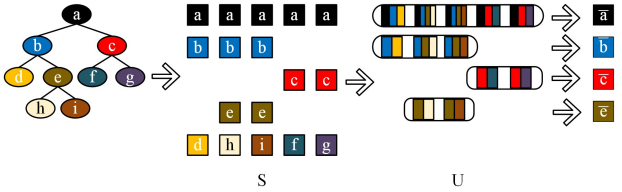
句法分析是将句子中的每个单词或短语赋予一个标签的过程,例如最常见的成分名词短语(Noun Phrasr,NP)。将自然语言中的句法信息引入神经网络有助于它们更有效地理解自然语言的含义并执行下游任务。

一个句子 $X=(x_1,x_2,x_3,\cdots,x_n)$ 可以被解析为 $T(X)=F(L,N,R)$,其中 x_i 表示句子中的每个单词,$L=\{l_1,l_2,\cdots,l_q\}$ 表示解析树中的叶节点,$N=\{n_1,n_2,\cdots,n_p\}$ 表示解析树中的非叶节点,R 表示树上各节点间的依赖关系,例如,$R(n_1)$ 代表以 n_1 为根的子树上所有节点的集合。如图 2 所示,句中的一个单词都会被解析为树上的叶节点,所以 X 的长度和 L 是一致的。

在得到句子的句法解析树后,需要对树节点进行嵌入,将其转换为指定维度的向量:$\boldsymbol{N}=[\boldsymbol{n}_1;\boldsymbol{n}_2;\cdots;\boldsymbol{n}_p]$,$\boldsymbol{L}=[\boldsymbol{l}_1;\boldsymbol{l}_2;\cdots;\boldsymbol{l}_q]$,其中矩阵 $\boldsymbol{n}_i$ 和 $\boldsymbol{l}_j$ 是行数为 1、列数为 $d_{\text{embedding}}$ 的矩阵,分号代表矩阵纵向拼接,逗号代表矩阵的横向拼接,$d_{\text{embedding}}$ 是嵌入维度,是模型的超参数。

2.2　分层聚合

在句法树中,具有相同标签的非叶子节点具有固定的表示,但这种固定的表示不能完全反映它们在树中的位置信息,因此需要加入节点聚合操作以携带更多信息。分层聚合的目的是并行编码树结构,图 3 展示了分层聚合的过程。



注:对于给定的树,将其插值到张量 S 中,然后自下而上垂直累计产生 U 张量,最后通过加权聚合将节点的分支表示组合成新的节点表示 $\bar{\boldsymbol{n}}_i$。多个色块表示各颜色节点的聚合操作。

图 3　分层聚合过程

Fig. 3　Hierarchical aggregation process

对于自然语言序列,句子中单词的位置和顺序对于句子的语法结构构成以及表达语义方面都至关重要。在标准的 Transformer 框架中,使用位置编码来编码序列中单词的位置。类似地,尽管从一个非叶子节点到以其为根的子树上所有叶子节点的路径上都包含一组独特的后代节点,但这些节点的层次差异以及各路径之间的兄弟关系并没有明确表示出来。因此,引入反映潜在子树级别层次结构的偏差可能会有益。引入了用于层次结构的嵌入,以融合节点的层次信息,在消融实验中也证明了其有效性。

若定义 $H(n_i)$ 为非叶节点 n_i 在树上的深度,其中根节点 root 的深度为 1,则根节点的直接子节点的深度为 2。在一棵树的深度 $(1,2,\cdots,H)$ 的范围内,每个深度都定义嵌入 $\boldsymbol{e}_h\in R^{d_{\text{embedding}}}$,这个嵌入将累加到对应层次的非叶节点的表示上。

为了执行聚合操作,定义一个张量 $S\in R^{(p+1)*q*d_{\text{embedding}}}$:

$$\boldsymbol{S}_{i,j}=\begin{cases}\boldsymbol{l}_j, & i=p+1\\ \boldsymbol{n}_i+\boldsymbol{e}_h, & l_j\in R(n_i),h=H(n_i)\\ \boldsymbol{0}, & \text{other conditions}\end{cases}\tag{1}$$

其中,p 是非叶节点个数,q 为叶节点个数,$p+1$ 行的第 j 列表示第 j 个叶节点,前 p 行的第 i 行代表一个非叶节点 n_i,若叶节点 l_j 不在以 n_i 为首的子树上,则 $\boldsymbol{S}_{i,j}=0$,否则 $\boldsymbol{S}_{i,j}=\boldsymbol{n}_i+\boldsymbol{e}_h$。

接着是对 $\boldsymbol{S}$ 执行向上累积平均操作 $\boldsymbol{U}$,在树结构中自下而上地组合节点表示:

$$\boldsymbol{U}(\boldsymbol{S})_{i,j}=\begin{cases}\boldsymbol{0}, & S_{i,j}=0\\ \sum_{\boldsymbol{s}_{t,j}\in C_j^i}\boldsymbol{S}_{t,j}/|C_j^i|, & \text{other conditions}\end{cases}\tag{2}$$

其中,$\boldsymbol{C}_j^i=\{\boldsymbol{S}_{p+1,j}\}\cup\{\boldsymbol{S}_{t,j}\mid\boldsymbol{n}_t\in\boldsymbol{R}(\boldsymbol{n}_i)\}\cup\{\boldsymbol{S}_{i,j}\}$ 是 n_i 到其子节点 l_j 路径上的节点集合所对应的 S 值。

最后,将每个非叶节点的分支表示 $\boldsymbol{S}_{i,j}$ 横向聚合成一个向量 $\bar{\boldsymbol{n}}_i$,以封装子树中的所有元素,这是通过加权聚合操作实现的。聚合函数以 $\boldsymbol{U}(\boldsymbol{S})_{i,j}$ 作为输入和权重向量 $\boldsymbol{w}\in R^q$,计算最终的节点表示 $\bar{\boldsymbol{N}}=(\bar{\boldsymbol{n}}_1,\cdots,\bar{\boldsymbol{n}}_p)$,其中每个向量 $\bar{\boldsymbol{n}}_i$ 的计算如下:

$$\bar{\boldsymbol{n}}_i=\frac{1}{|\{l_j\mid l_j\in R(n_i)\}|}\sum_{j=1}^{q}w_j\odot\boldsymbol{U}(\boldsymbol{S})_{i,j}\tag{3}$$

其中,符号 $\odot$ 表示元素逐个相乘,以反映不同分支的影响。经过这些操作后,非叶节点可以携带来自其所有子节点的信息,这有助于神经网络理解句法树。

2.3　子树屏蔽

注意力屏蔽是过滤不相关信号的常用方法。在标准的 Transformer 中,查询 q_i 和键值 k_j 间的注意值在 $j>i$ 时被屏蔽,以避免未来的键值被关注,因为来自未来的键值在推理时不可用。

在 Tree_Transformer 中,本文将会为编码器自注意层引入子树屏蔽。如果 q_i 是非叶节点并且 k_j 在以 q_i 为根的子树上,或者是 q_i 和 k_j 都是叶节点,则保留它们之间的相似度,否则将它们间的相似度置为负无穷,这样经过 softmax,它们的注意力分数将变成 0。换句话说,每个节点的查询只能访问自己的子树后代,不能访问它的祖先或兄弟。子树屏蔽操作 μ 计算如下:

$$\mu(a_{ij})=\begin{cases}a_{ij}, & \text{if } (q_i\in N \text{ and } k_j\in R(q_i)) \text{ or } q_i,k_j\in L\\ -\infty, & \text{otherc onditions}\end{cases}\tag{4}$$

2.4　基于 Tree_Transformer 的编码器解码器模型

编码器需要自注意机制,以允许源序列中的每个单元捕获来自树上其他节点的信息。

具体地,首先,树的非叶子节点和叶子节点相互进行比较,并分别计算它们之间的相似性 $\boldsymbol{A}_{\text{NL}}\in R^{p*q}$,$\boldsymbol{A}_{\text{LL}}\in R^{q*q}$,$\boldsymbol{A}_{\text{NN}}\in R^{p*p}$,$\boldsymbol{A}_{\text{LN}}\in R^{q*p}$:

$$\boldsymbol{A}_{\text{NL}}=\frac{(\boldsymbol{NW}^Q)(\boldsymbol{LW}^K)^{\text{T}}}{\sqrt{d_k}}\tag{5}$$

$$\boldsymbol{A}_{\text{LL}}=\frac{(\boldsymbol{LW}^Q)(\boldsymbol{LW}^K)^{\text{T}}}{\sqrt{d_k}}\tag{6}$$

$$\boldsymbol{A}_{\text{NN}}=\frac{(\boldsymbol{NW}^Q)(\boldsymbol{NW}^K)^{\text{T}}}{\sqrt{d_k}}\tag{7}$$

$$\boldsymbol{A}_{\mathrm{LN}}=\frac{(\boldsymbol{L}\boldsymbol{W}^{Q})(\boldsymbol{N}\boldsymbol{W}^{K})^{\mathrm{T}}}{\sqrt{d_k}} \tag{8}$$

其中，$\boldsymbol{W}^{Q},\boldsymbol{W}^{K}\in R^{d_{\mathrm{embedding}}*d_k}$。将得到的叶节点与非叶节点之间的相似度进行连接，并通过$\overline{\boldsymbol{N}}'=\overline{\boldsymbol{N}}\boldsymbol{W}^{V}$和$\overline{\boldsymbol{L}}=\boldsymbol{L}\boldsymbol{W}^{V}$计算节点和叶的最终注意力$\boldsymbol{Att}_N\in R^{p*d_v}$，$\boldsymbol{Att}_L\in R^{q*d_v}$：

$$\boldsymbol{Att}_N=\mathrm{softmax}(\mu([\boldsymbol{A}_{\mathrm{NN}},\boldsymbol{A}_{\mathrm{NL}}]))[\overline{\boldsymbol{N}}';\overline{\boldsymbol{L}}] \tag{9}$$

$$\boldsymbol{Att}_L=\mathrm{softmax}(\mu([\boldsymbol{A}_{\mathrm{LN}},\boldsymbol{A}_{\mathrm{LL}}]))[\overline{\boldsymbol{N}}';\overline{\boldsymbol{L}}] \tag{10}$$

其中，$\boldsymbol{W}^{V}\in\boldsymbol{R}^{d_{\mathrm{embedding}}*d_v}$。

最后，在归一化层(LN)[16]和前馈层(FFN)之后，获得了编码器最终的表示$\hat{\boldsymbol{N}}\in R^{p*d_{\mathrm{embedding}}}$和$\hat{\boldsymbol{L}}\in R^{q*d_{\mathrm{embedding}}}$。

$$FFN(\boldsymbol{X})=\max(0,\boldsymbol{X}\boldsymbol{W}_1+b_1)\boldsymbol{W}_2+\boldsymbol{b}_2 \tag{11}$$

$$\hat{\boldsymbol{N}}=LN(FFN(LN(\boldsymbol{Att}_N\boldsymbol{W}^{o}+\boldsymbol{N}))+LN(\boldsymbol{Att}_N\boldsymbol{W}^{o}+\boldsymbol{N})) \tag{12}$$

$$\hat{\boldsymbol{L}}=LN(FFN(LN(\boldsymbol{Att}_L\boldsymbol{W}^{o}+\boldsymbol{L}))+LN(\boldsymbol{Att}_L\boldsymbol{W}^{o}+\boldsymbol{L})) \tag{13}$$

其中，$\boldsymbol{W}^{o}\in R^{d_v*d_{\mathrm{embedding}}}$，$\boldsymbol{W}_1\in R^{d_{\mathrm{embedding}}*d_{ff}}$，$\boldsymbol{W}_2\in R^{d_{ff}*d_{\mathrm{embedding}}}$，$\boldsymbol{b}_1\in R^{d_{ff}}$，$\boldsymbol{b}_2\in R^{d_{\mathrm{embedding}}}$。

对于生成序列的任务，解码器具有交叉注意力，使得目标侧的查询可以利用源侧的编码信息。具体来说，首先对解码器的每个时间步t输入$\boldsymbol{Q}^{t}$和已生成的序列$\boldsymbol{K}^{t}$，计算得到自注意力得分：

$$\boldsymbol{A}_{QK}=\frac{(\boldsymbol{Q}^{t}\boldsymbol{W}^{Q})(\boldsymbol{K}^{t}\boldsymbol{W}^{K})^{\mathrm{T}}}{\sqrt{d_k}} \tag{14}$$

$$\boldsymbol{Att}_Q=\mathrm{softmax}(Mask(\boldsymbol{A}_{QK}))[\overline{\boldsymbol{N}};\overline{\boldsymbol{L}}] \tag{15}$$

值得注意的是，本模型在训练阶段同样执行标准 transformer 框架的掩码操作 Mask，防止模型使用未来的信息来预测，导致测试时效果不好。之后，在归一化层(LN)[16]和前馈层(FFN)之后，获得了解码器输出的表示$\overline{\boldsymbol{Q}}^{t}\in R^{t*d_{\mathrm{embedding}}}$：

$$\overline{\boldsymbol{Q}}^{t}=LN(FFN(LN(\boldsymbol{Att}_Q\boldsymbol{W}^{o}+\boldsymbol{Q}^{t}))+LN(\boldsymbol{Att}_Q\boldsymbol{W}^{o}+\boldsymbol{Q}^{t})) \tag{16}$$

然后根据解码器自注意模块的输出$\overline{\boldsymbol{Q}}^{t}$以及自注意编码器输出的叶子节点和非叶子节点表示，先计算相似性分数$\boldsymbol{A}_{QL}\in R^{t*q}$和$\boldsymbol{A}_{QN}\in R^{t*p}$：

$$\boldsymbol{A}_{QN}=\frac{(\overline{\boldsymbol{Q}}^{t}\boldsymbol{W}^{Q})(\hat{\boldsymbol{N}}\boldsymbol{W}^{K})^{\mathrm{T}}}{\sqrt{d_k}} \tag{17}$$

$$\boldsymbol{A}_{QL}=\frac{(\overline{\boldsymbol{Q}}^{t}\boldsymbol{W}^{Q})(\hat{\boldsymbol{L}}\boldsymbol{W}^{K})^{\mathrm{T}}}{\sqrt{d_k}} \tag{18}$$

然后通过$\hat{\boldsymbol{N}}\boldsymbol{W}^{V}$和$\hat{\boldsymbol{L}}\boldsymbol{W}^{V}$计算节点和叶的交叉注意力分数$\boldsymbol{Att}_Q\in R^{t*d_v}$：

$$\boldsymbol{Att}_Q=\mathrm{softmax}([\boldsymbol{A}_{QN};\boldsymbol{A}_{QL}])[\hat{\boldsymbol{N}}\boldsymbol{W}^{V};\hat{\boldsymbol{L}}\boldsymbol{W}^{V}] \tag{19}$$

在经过归一化层(LN)[15]和前馈层(FFN)后，获得了解码器最终的表示$\hat{\boldsymbol{Q}}^{t}\in R^{t*d_{\mathrm{embedding}}}$：

$$\hat{\boldsymbol{Q}}^{t}=LN(FFN(LN(\boldsymbol{Att}_Q\boldsymbol{W}^{o}+\boldsymbol{Q}))+LN(\boldsymbol{Att}_Q\boldsymbol{W}^{o}+\boldsymbol{Q})) \tag{20}$$

最后，通过全连接层(Linear)将解码器所得到的$\hat{\boldsymbol{Q}}$的最后一个维度转换为词表大小(vocab_size)，并用 softmax 计算出第t个时间步的概率分布：

$$\boldsymbol{P}_t=\mathrm{softmax}(Linear(\hat{\boldsymbol{Q}}^{t})) \tag{21}$$

训练阶段将采用交叉熵作为损失函数，来反映神经网络对训练集的拟合程度：

$$Loss=-\sum_{t=1}^{T}\boldsymbol{p}_t\log\boldsymbol{q}_t \tag{22}$$

其中，T为最大的解码步长，$\boldsymbol{p}_t$为真实的概率分布，$\boldsymbol{q}_t$为预测的概率分布。

3 实验设置

3.1 数据集

本文采用两个公共数据集进行模型训练和测试。

KB13：包含 824 对自然语言(Natural Language，NL)描述和正则表达式(regex)。在数据集的创建过程中，注释者首先生成了自然语言规范，然后程序员根据这些自然语言规范生成了相应的正则表达式。实验数据被分成了 65%的训练集、10%的验证集和 25%的测试集。

NL-RX-Turk：源自 NL-RX-Synth[15]数据集。与直接使用合成的自然语言描述不同，NL-RX-Turk 的作者要求注释者重新表述合成的规范。Rabinovich 等[17]受到 Wang[18]的启发，开发了一个用于正则表达式的抽象语法描述语言(ASDL)，如图 4 所示，并将该数据集重新编写为符合 ASDL 约束的中间表示，如表 2 所列。这里的编码器不再预测正则表达式中的字符，而是构建符合 ASDL 约束的操作，然后通过一个转换系统将其转化为正则表达式。该数据集包含 10000 对自然语言描述和正则表达式。为了进行实验，这些数据对被分成了 65%的训练集、10%的验证集和 25%的测试集。

```
int, cc, tok

regex = Not(regex arg)
    | Star(regex arg)
    | Optional(regex arg)
    | Concat(regex left, regex right)
    | Or(regex left, regex right)
    | And(regex left, regex right)
    | StartWith(regex arg)
    | EndWith(regex arg)
    | Contain(regex arg)
    | RepeatAtleast(regex arg, int k)
    | CharClass(cc arg)
    | Const(tok arg)
```

图 4 正则表达式的 ASDL

Fig. 4 ASDL for regular expressions

表 2 ASDL 中间表示及对应的正则表达式示例

Table 2 ASDL intermediate representation and examples of corresponding regular expression

ASDL 中间表示	正则表达式
contain(and(⟨num⟩,⟨let⟩))	.*([0-9])&([A-Z a-z]).*
concat(repeatatleast(⟨num⟩,6),⟨let⟩)	(([0-9]){6,}).*([A-Z a-z]).*

为了准确评估模型，本文使用 DFA-Equal-Acc(DFA 等效率)。之所以检查功能等效性，是因为有许多编写正则表达式的方式是等效的，例如(a|b)在功能上等效于(b|a)，尽管它们的字符串表示不同。这个评估指标被广泛应用于基线模型上。本文的 DFA-Equal-Acc 使用了 Kushman[19]的实现来直接比较结果。

3.2 实验参数设置

本文的模型基于 NVIDIA GeForce RTX 3080 显卡，并使用 Pytorch 框架在 GPU 上进行训练。数据集的词嵌入维数设置为 128，初始学习率$lr=0.003$，dropout 为 0.3，神经网络

中的$d_k=d_v=32$。对于 KB13 数据集,batch_size 设置为 64;对于 NL-RX-Turk 数据集,batch_size 设置为 32,编码器和解码器的堆叠层数N_x为 8。

4　实验结果与分析

4.1　性能分析

将本文模型与其他模型在这两个数据集的正则表达式生成任务上进行了比较。这些模型是:(1)Deep-RegEx[4];(2)torchASN[18],用于代码生成和语义解析的抽象语法网络;(3)SemRegex(Oracle)[9],一种基于语义的用于从自然语言规范生成正则表达式的方法;(4)SoftRegex[10],使用软化的正则表达式等价性从自然语言描述生成正则表达式;(5)DeepSketch[20],草图驱动的正则表达式生成;(6)InfeRE[21],通过推理链逐步生成正则表达式。

从表 3 可以看出,Tree_Transformer 在 NL-RX-Turk 数据集中具有比其他模型更高的精度,比 DeepSketch 模型提高了 2.9%,与其他模型相比,至少提升了 25.3%。模型在 KB13 数据集上的性能也都超越了其他模型。本文模型具有比已有模型更高的精度,这表明语义解析在自然语言理解中具有重要作用。值得注意的是,Tree_Transformer 相较于 DeepSketch 模型的提升远不如相较于其他 4 个模型的提升,这是因为 DeepSketch 也运用到了对自然语言处理后再输入,这说明自然语言的特殊化处理对神经网络性能的提升是巨大的。与 Deep-RegEx 和 torchASN 等使用自然语言线性输入的模型相比,对树结构的编码使本文模型能够更好地理解单词之间的语义关系,这有助于从源序列到目标序列的信息提取。

表 3　实验结果

Table 3　Experimental results

(%)

数据集	KB13	NL-RX-Turk
Deep-RegEx	65.6	58.2
torchASN	—	61.6
SemRegex(Oracle)	78.2	62.3
SoftRegex	78.2	62.8
DeepSketch	84.0	85.2
Tree_Transformer(本文模型)	84.5	88.1

4.2　组件分析

本文进行了消融实验,分析表 4 中各组件的贡献。无分层聚合(No Hierarchy Aggregation)意味着非叶节点使用初始嵌入输入编码器。无子树屏蔽(No Subtree Masking)意味着对节点之间的关注没有限制,查询可以关注其他子树中的键值。No Both 表示模型既没有分层聚合,也没有子树屏蔽。在消融实验中,没有分层聚合和子树屏蔽的模型都显示出比完整模型更低的预测准确度,特别是没有子树屏蔽的模型。与完整模型相比,没有分层累积的模型在 KB13 上的准确度下降了 7.9%,在 NL-RX-Turk 上下降了 11.9%;而没有子树屏蔽的模型在 KB13 上的准确度下降了 11.2%,在 NL-RX-Turk 上下降了 13.4%;No Both 的表现更差。通过比较,发现没有子树屏蔽的模型表现比没有分层累积的模型更差,这表明子树屏蔽的重要性。然而,没有子树屏蔽的模型的性能比 No Both 要好,因此分层累积也不应该被忽视。

表 4　消融实验结果

Table 4　Ablation experiment results

(%)

数据集	KB13	NL-RX-Turk
Nohierarchy aggregation	76.6	76.2
No subtree masking	73.3	74.7
No Both	70.2	71.5
Tree_Transformer	84.5	88.1

结束语　本文提出了一种基于深度学习的文本生成方法,用于将自然语言描述转换成正则表达式。该模型利用句法分析树的分层聚合结合自注意编码来预测正则表达式基于 ASDL 规则的中间表示,打破了自然语言单一的表示形式,增强了解码信息的丰富性,并通过 ASDL 的约束增强了所生成的正则表达式的准确性。与其他的模型相比,Tree_Transformer 模型在准确性方面表现更好。然而,该模型只从源端学习了一些隐含信息,并没有直接将精确的源信息对应到目标端。在未来,将尝试使用提取源端主要 token 的方式,直接将源信息应用到目标端,以减少查找词表的次数,提升模型的性能。

参 考 文 献

[1] MILOSAVLJEVIĆ B, VIDAKOVIĆ M, KONJOVIĆ Z. Automatic code generation for database-oriented web applications [C]// Proceedings of the Inaugural Conference on the Principles and Practice of Programming, 2002 and Proceedings of the Second Workshop on Intermediate Representation Engineering for Virtual Machines, 2002: 59-64.

[2] ZETTLEMOYER L S, COLLINS M. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars[J]. arXiv: 1207.1420, 2012.

[3] HINDLE A, BARR E T, GABEL M, et al. On the naturalness of software[J]. Communications of the ACM, 2016, 59(5): 122-131.

[4] KUSHMAN N, BARZILAY R. Using semantic unification to generate regular expressions from natural language[C]// North American Chapter of the Association for Computational Linguistics(NAACL). 2013.

[5] YIN P, NEUBIG G. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation[J]. arXiv: 1810.02720, 2018.

[6] ZHONG V, XIONG C, SOCHER R. Seq2sql: Generating structured queries from natural language using reinforcement learning[J]. arXiv: 1709.00103, 2017.

[7] LIU X, JIANG Y, WU D. A lightweight framework for regular expression verification[C]// 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE). IEEE, 2019: 1-8.

[8] SUTSKEVER I, VINYALS O, LE Q V. Sequence to sequence learning with neural networks[J]. Advances in Neural Information Processing Systems, 2014, 27.

[9] ZHONG Z, GUO J, YANG W, et al. Semregex: A semantics-based approach for generating regular expressions from natural language specifications[C]// Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 2018.

[10] PARK J U, KO S K, COGNETTA M, et al. Softregex: Genera-

ting regex from natural language descriptions using softened regex equivalence[C]// Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 2019:6425-6431.

[11] CHEN X, LIU C, SONG D. Tree-to-tree neural networks for program translation[J]. Advances in Neural Information Processing Systems, 2018, 31.

[12] RAO J, UPASANI K, BALAKRISHNAN A, et al. A tree-to-sequence model for neural nlg in task-oriented dialog[C]// Proceedings of the 12th International Conference on Natural Language Generation. 2019:95-100.

[13] SCHUSTER S, MANNING C D. Enhanced english universal dependencies: An improved representation for natural language understanding tasks[C]// Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16). 2016:2371-2378.

[14] SHI X, PADHI I, KNIGHT K. Does string-based neural MT learn source syntax? [C]// Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016: 1526-1534.

[15] CHEN H, HUANG S, CHIANG D, et al. Improved neural machine translation with a syntax-aware encoder and decoder[J]. arXiv:1707.05436, 2017.

[16] LOCASCIO N, NARASIMHAN K, DELEON E, et al. Neural generation of regular expressions from natural language with minimal domain knowledge[J]. arXiv:1608.03000, 2016.

[17] RABINOVICH M, STERN M, KLEIN D. Abstract syntax networks for code generation and semantic parsing[J]. arXiv:1704.07535, 2017.

[18] WANG D C, APPEL A W, KORN J L, et al. The Zephyr Abstract Syntax Description Language[C]// DSL. 1997.

[19] KUSHMAN N, BARZILAY R. Using semantic unification to generate regular expressions from natural language[C]// North American Chapter of the Association for Computational Linguistics (NAACL). 2013.

[20] YE X, CHEN Q, WANG X, et al. Sketch-driven regular expression generation from natural language and examples[J]. Transactions of the Association for Computational Linguistics, 2020, 8:679-694.

[21] ZHANG S, GU X, CHEN Y, et al. InfeRE: Step-by-Step Regex Generation via Chain of Inference[C]// 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2023:1505-1515.

WANG Hao, born in 1999, postgraduate. His main research interests include software engineering and natural language processing.

WU Junhua, born in 1965, Ph.D, professor. Her main research interests include software engineering, program analysis and natural language processing.