北京大学学报(自然科学版) 第 57 卷 第 1 期 2021 年 1 月
Acta Scientiarum Naturalium Universitatis Pekinensis, Vol. 57, No. 1 (Jan. 2021)
doi: 10.13209/j.0479-8023.2020.086

具有选择性局部注意力和前序信息解码器的代码生成模型

梁婉莹[1,2] 朱佳[1,2,†] 吴志杰[1,2] 颜志文[1,2] 汤庸[1,2] 黄晋[1,2] 余伟浩[1,2]

1. 华南师范大学计算机学院, 广州 510631; 2. 广州市大数据智能教育重点实验室, 广州 510631;
† 通信作者, E-mail: jzhu@m.scnu.edu.cn

摘要 提出一种基于语法的代码生成模型, 该模型具有选择性局部注意力和包含前序信息的长短期记忆(LSTM)神经网络解码器, 通过更改上下文向量的计算范围, 并在解码过程中融合更多的前序信息, 增强单词之间的相关性。在 Hearthstone 和 Django 两个数据集上进行的代码生成实验证实了所提模型的有效性, 与最新的模型相比, 所提模型不仅表现出更出色的准确率和双语评估学习成绩, 还可以使计算工作量最小化。

关键词 代码生成; 抽象语法树; 包含前序信息的长短期记忆神经网络(LSTM); 选择性局部注意力

Syntax-based Code Generation Model with Selective Local Attention and Pre-order Information LSTM Decoder

LIANG Wanying[1,2], ZHU Jia[1,2,†], WU Zhijie[1,2], YAN Zhiwen[1,2], TANG Yong[1,2], HUANG Jin[1,2], YU Weihao[1,2]

1. School of Computer, South China Normal University, Guangzhou 510631; 2. Guangzhou Key Laboratory of Big Data and Intelligent Education, Guangzhou 510631; † Corresponding author, E-mail: jzhu@m.scnu.edu.cn

Abstract This paper proposes a syntax-based code generation model with selective local attention and a pre-order information decoder based on long-short term memory (LSTM) neural network, which aims to enhance the relevance by changing the calculation scope of the context vector and fuse more pre-order information during the decoding process. Code generation experiments in two dataset Hearthstone and Django confirm the effectiveness of this model. Compared with state-of-the-art models, the proposed model not only achieves excellent accuracy and bilingual evaluation understudy score, but also minimizing computational effort.

Key words code generation; abstract syntactic tree; pre-order information LSTM; selective local attention

代码生成是机器翻译的一个分支问题, 比自然语言生成或翻译[1]要求更高。代码符合规定的文法以及格式时, 才能被机器真正识别并运行。随着机器学习领域的发展, 人们逐渐找到代码生成的解决方案。Brown 等[2]提出仅依赖于统计分析的统计机器翻译模型(statistical machine translation, SMT), 尽管其准确率相对较低, 却是当时代码生成问题的新突破。随后, 基于 Seq2Seq[3]框架的神经机器翻译模型(neural machine translation, NMT)[4]被提出, 可以从自然语言序列映射到目标代码序列, 支持可变长度的输入和输出, 在翻译、对话和单词概括中表现出色。之后, 一些研究人员尝试使用指针网络来改进 NMT 模型, 例如致力于解决输出严重依赖于输入问题的潜伏预测器网络(latent predictor networks, LPN)[1]、通过长短期记忆单元(Long-Short Term Memory, LSTM)[5]生成多层树的代码逻辑形

国家自然科学基金(61877020, U1811263, 61772211)、广东省重点领域研发计划(2018B010109002)、广州市科学技术计划项目(201904010393)和广州市大数据智能教育重点实验室(201905010009)资助
收稿日期: 2020–06–08; 修回日期: 2020–08–08

式的Seq2Tree模型[6]、基于句法神经网络的Seq2Seq模型(syntactic neural network model, SNM)[7]、抽象语法网络(abstract syntax networks, ASN)模型[8]、基于语义分析和代码生成的抽象解析器(TRANX)[9]、采用Transformer[10]架构并捕获长依赖性句子的TreeGen[11]模型以及基于语法结构的卷积神经网络(CNN decoder model)解码器模型[12]。

上述模型的双语评估研究指标(BLEU)和准确率上逐渐得到提高, 但在代码生成方面仍存在以下缺陷: 首先, 传统的序列到序列(Seq2Seq)模型(如LPN模型[1]和SNM模型[7])的解码器包含的结构信息较少, 可能导致前序信息对当前运算的影响不足, 使得当前向量在抽象语法树中位置选择错误; 其次, 单纯使用软注意力(即将所有编码器输出向量置入上下文向量的计算中), 会导致输入序列的先验语义向量被后验语义向量覆盖, 缺乏对输出语法与模型特征[1]之间模块化或紧密耦合的考虑。

针对以上问题, 本文提出一种基于语法的代码生成模型。该模型具有选择性局部注意力和带有前序信息的长短期记忆单元解码器PI-LSTM(preorder information LSTM), 使用序列到序列(Seq2-Seq)的编码–解码框架。在编码器部分, 使用门控循环单元(gate recurrent unit, GRU)[13]作为计算单元。与SNM模型中的朴素LSTM相比, GRU相对简单高效, 在减少训练时间的同时, 对准确率或双语评估学习成绩的BLEU分数产生可忽略的负面影响, 适用于训练大规模的预处理数据。在注意力机制部分, 为了增强上下文之间的相关性, 我们使用选择性局部注意力机制, 可以根据预设或输入语句的长度采用不同的注意力机制, 从而减少无关单词对当前单词权重计算的干扰。与单纯使用软注意力机制相比, 选择性局部注意力可以更好地适应不同大小的数据集。在解码器部分, 为了包含更多的结构信息, 我们使用基于朴素LSTM改进的PI-LSTM作为解码器计算单元。通过对输出值的二次更新, PI-LSTM提升了前序信息在解码运算中的参与度, 更好地修正了解码运算, 减少了模型在抽象语法树中生成错误分支的可能性。

1 代码生成

1.1 SNM 模型

基于句法神经网络的Seq2Seq模型(SNM)[7]是目前最具代表性的代码生成模型。SNM模型不必恢复原有的基础语法, 只需要集中精力学习现有语法规则之间的可组合性[7]。它配备了标准的递归神经网络(RNN)[14]解码器, 允许更多的神经连接反映抽象语法树的递归结构。SNM模型应用双向LSTM(bidirection-LSTM)[15]编码器、LSTM解码器和软注意力来训练给定的预处理数据集, 与2017年前推出的模型相比, SNM模型的准确率非常出色。

1.2 抽象语法树

抽象语法树(abstract syntax tree, AST)[16]是领域专用语言(即代码)的树型结构。AST仅用于代码摘要, 不代表实际语法的每个细节, 其每个节点都代表源代码中的一个分支。抽象语法树有两个特征: 1) 不依赖具体的语法, 在解析输入语句时, 系统会构造相同的语法树, 且该语法树给编译器后端提供一个清晰、统一的接口; 2) 构造语法树时不依赖语言的细节, 并且, 针对不同的语言有不同的表示形式[7]。

2 编码器–解码器代码生成模型

代码生成是不定长输入到不定长输出的转换过程。本文提出的基于语法的代码生成模型如图1所示。该模型具有GRU编码器、选择性局部注意力以及PI-LSTM解码器。模型根据输入序列的第一个单词, 生成目标抽象语法树的主干, 然后通过编码器–解码器的计算, 逐步将语法树主干填充成完整语法树。编码器对输入序列中的每个单词进行编码, 并将编码输出传送到选择性局部注意力机制中, 计算上下文向量。解码器接收到上下文向量后, 使用PI-LSTM计算并解码, 得到最终输出。

2.1 底层语法驱动

代码生成模型多数以抽象语法树(AST)为基础。在计算机科学和语言学中, 句法分析(或语法解析)[17]是一种对由一系列单词组成的输入文本的语法结构进行分析和确定的过程。底层语法驱动程序包含一组生产规则, 并通过固有的几个生产规则, 生成头节点和多个子节点, 从而生成相应的树结构, 如图2所示。如果输入序列中的第一个单词是“if”, 模型则以“If→expr [test] stmt * [body] stmt * [or else]”的形式生成主干AST。然后使用GENTO-KEN和APPLYRULE, 分别在非终端节点和终端节点上进行操作, 前者通过添加终端令牌来填充可变终端节点, 后者将生产规则应用于当前派生树, 最终达成扩展和填充空值语法树的目的。

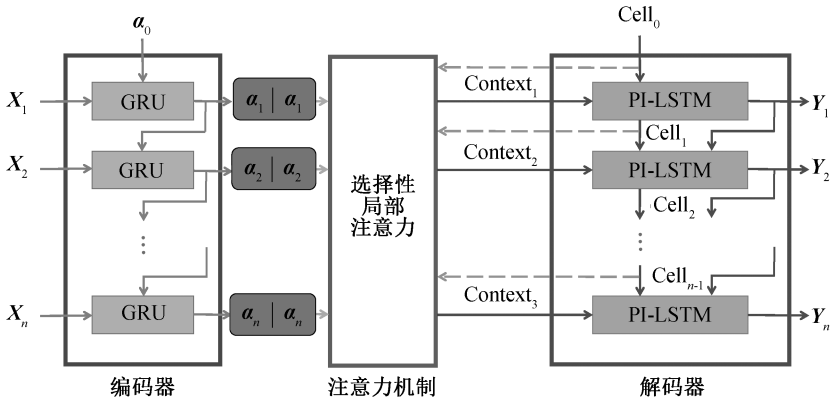
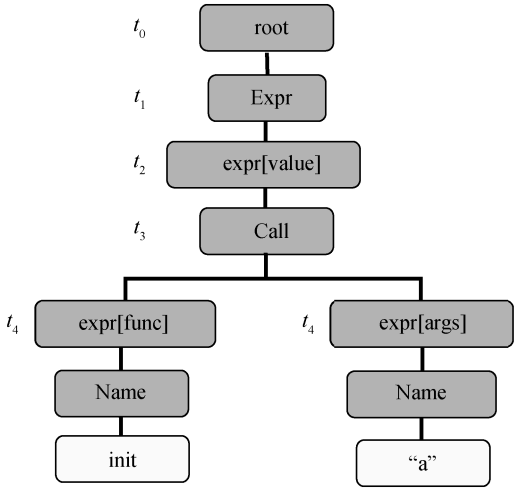


图 1　基于语法的选择性局部注意力和 PI-LSTM 解码代码生成模型

Fig. 1　A syntax-based code generation model with selective local attention and PI-LSTM decoder



t_0~t_4 指底层语法驱动程序遍历语法树的时间步

图 2　代码 init (a)的抽象语法树构造

Fig. 2　Abstract syntax tree structure of code init (a)

2.2 门控循环单元(GRU)编码器

为了减少模型训练的时间，在编码器中，我们以门控循环网络 GRU 作为计算单元。GRU 的门控通过丢弃或保留输入序列数据来调整信息流并沿序列传递相关信息，用于预测下一个输出。GRU 是 LSTM[5]的一种变体，它将遗忘门和输入门简化为单个更新门，使计算相对简单，更适合训练大规模的预处理数据。给定输入序列中的每个单词向量 $\boldsymbol{X}_i=\{X_1, X_2, ..., X_n\}$，通过 GRU 计算转换为编码向量 $\boldsymbol{a}_i$ (默认 $\boldsymbol{a}_0$ 是零向量)。

首先，将单词向量 $\boldsymbol{X}_i$ 和前序编码输出向量 $\boldsymbol{a}_{i-1}$ 通过 Sigmoid 函数[18]，计算获得更新门的结果 z_i 和重置门的结果 r_i:

$$z_i=\text{Sigmoid}\,(\boldsymbol{W}_z\cdot[\boldsymbol{a}_{i-1}, \boldsymbol{X}_i]), \tag{1}$$

$$r_i=\text{Sigmoid}\,(\boldsymbol{W}_r\cdot[\boldsymbol{a}_{i-1}), \boldsymbol{X}_i])。 \tag{2}$$

然后，将单词向量 $\boldsymbol{X}_i$、前序编码向量 $\boldsymbol{a}_{i-1}$ 和复位门结果 r_i 输入 tanh 函数，计算当前记忆内容 $\tilde{\boldsymbol{h}}_i$:

$$\tilde{\boldsymbol{h}}_i=\tanh\,(\boldsymbol{W}\cdot[r_i\cdot\boldsymbol{a}_{i-1}, \boldsymbol{X}_i])。 \tag{3}$$

以上公式中的 $\boldsymbol{W}_z$、$\boldsymbol{W}_r$ 和 $\boldsymbol{W}$ 均为待学习的权重参数，通过神经网络训练获得其最优值。

最后，将更新门结果 z_i、前序编码向量 $\boldsymbol{a}_{i-1}$ 和当前记忆内容 $\tilde{\boldsymbol{h}}_i$ 输入式(4)，获得最终记忆内容，即当前编码输出 $\boldsymbol{a}_i$:

$$\boldsymbol{a}_i=\boldsymbol{a}_{i-1}\cdot(1-z_i)+z_i\cdot\tilde{\boldsymbol{h}}_i。 \tag{4}$$

2.3 选择性局部注意力

在代码生成问题中，注意力机制[19]是相对重要的环节。无论输入序列的长短，SNM 模型使用软注意力机制，将所有编码输出匹配度 α 和编码输出 $\boldsymbol{a}$ 加入上下文相关度计算中，并通过 Softmax 函数[20]将其归一化。在每个解码时间步 j 中，将编码后的向量 $\boldsymbol{a}_i$ 和前序解码单元状态 Cell_{j-1} 连接在一起。经过式(5)和(6)的计算以及 Softmax 函数归一化，获得编码输出向量 $\boldsymbol{a}_i$ 之间的匹配度 a_{ij}:

$$e_{ij}=\tanh(\boldsymbol{W}_a[\text{Cell}_{j-1};\boldsymbol{a}_i]), \tag{5}$$

$$\alpha_{ij}=\frac{\exp(e_{ij})}{\sum_{k=1}^{T_X}\exp(e_{ik})}。 \tag{6}$$

其中，$\boldsymbol{W}_a$ 为待学习的权重参数。

单纯使用软注意力的计算方式虽然较常用，但会导致在长句翻译中，单词之间的相关性数值普遍相似，差别不明显。被偏置的相关性值将影响当前的解码输出，导致生成目标代码的准确率下降。因

此，我们参考其他注意力机制(如局部注意力机制)，如图 3 所示。与软注意力机制相比，局部注意力使用滑窗限制上下文向量的计算范围，在长句上的表现明显更好，但单纯使用局部注意力机制仍将导致在短输入序列中语境理解能力较弱，准确率相对较低。

本文提出一种用于代码生成的选择性局部注意力机制。模型首先判断输入序列的单词数量 n，然后确定可选范围的大小 K。当 $n<K$ 时，模型将使用软注意力机制[21]，即将所有编码输出匹配度 $\boldsymbol{\alpha}$ 以及编码输出 a 加入上下文相关度计算中:

$$\text{Context}^{<t>}=\sum_{i=1}^{n}\alpha(i,i')\cdot a^{i'},\ n<K。\tag{7}$$

当 $n\geq K$ 时，则使用局部注意力机制的滑动窗口来限制上下文向量的计算范围，并根据当前计算位置滑动窗口。对滑动窗口中所有编码输出匹配度 a 和编码输出 a 执行加权和计算:

$$\text{Context}^{<t>}=\sum_{i=\text{word}_t-D}^{\text{word}_t+D}\alpha(i,i')\cdot a^{i'},\ n\geq K。\tag{8}$$

通过限制上下文向量的计算范围，选择性局部注意力机制减少了无关单词向量对当前计算单词向量的干扰。当前计算位置 t 在滑动窗口的正中间，意味着当前计算向量将与左右两边的向量共同参与上下文相关度计算。如果滑动窗口的下限小于位置 1，则从位置 1 开始。如果滑动窗口的上限大于序列的最后位置，则以最后位置结束。滑动窗口的长度 $D=K/2$。我们经过多次实验证明，当 $n>12$ 时，K 值取 12 (即 $D=6$)可达最优效果；当 $n<12$ 时，则 K 值取 14 (即 $D=7$)。

2.4 包含前序信息的长短期记忆网络

在解码部分，为了将更多的前序解码信息添加到网络结构中进行计算，我们使用 PI-LSTM 作为解码器计算单元，如图 4 所示。

PI-LSTM 是基于朴素 LSTM 改进的计算单位。模型利用 PI-LSTM 对输出进行二次更新，从而减小在抽象语法树中生成错误的分支的可能性。在每个时间步中，将上下文向量 Context_t、前序 PI-LSTM

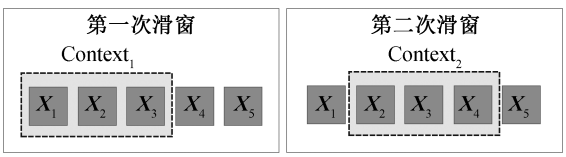


图 3 局部注意力工作示意图
Fig. 3 Schematic diagram of local attention

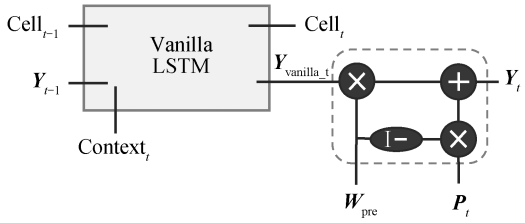


图 4 PI-LSTM 结构图
Fig. 4 Structure of PI-LSTM

解码输出 $\boldsymbol{Y}_{t-1}$ 和前序解码单元状态 Cell_{t-1} 输入 PI-LSTM 中，获得当前解码输出 $\boldsymbol{Y}_t$ 和当前操作单元状态 Cell_t。以下公式中的 $\boldsymbol{W}$ 是每个门控中实时变化的权重，最佳参数通过模型训练获得。

首先，将前序解码输出 $\boldsymbol{Y}_{t-1}$ 以及上下文向量 Context_t 输入 Sigmoid 函数，计算遗忘门 f_t 和输入门 i_t:

$$f_t=\text{Sigmoid}(\boldsymbol{W}_f\cdot[\boldsymbol{Y}_{t-1},\text{Context}_t]+\boldsymbol{b}_{\text{f}}),\tag{9}$$

$$i_t=\text{Sigmoid}(\boldsymbol{W}_i\cdot[\boldsymbol{Y}_{t-1},\text{Context}_t]+\boldsymbol{b}_{\text{i}})。\tag{10}$$

然后，将前序解码向量 Y_{t-1} 以及上下文向量 Context_t 输入 tanh 函数，计算当前记忆状态 $\tilde{c}_t$:

$$\tilde{c}_t=\tanh(\boldsymbol{W}_C\cdot[\boldsymbol{Y}_{t-1},\text{Context}_t]+\boldsymbol{b}_{\text{C}})。\tag{11}$$

最后，对遗忘门结果 f_t、输入门结果 i_t 和前序计算单元状态 Cell_{t-1} 进行如下计算，获得当前操作单元状态 Cell_t 的状态:

$$\text{Cell}_t=\text{Cell}_{t-1}\cdot f_t+i_t\cdot\tilde{c}_t。\tag{12}$$

此时，我们需要确认最终输出。将上下文向量 Context_t 输入 Sigmoid 函数中，计算输出门结果 o_t:

$$o_t=\text{Sigmoid}(\boldsymbol{W}_o\cdot[\boldsymbol{Y}_{t-1},\text{Context}_t]+\boldsymbol{b}_o)。\tag{13}$$

将当前操作单元状态 Cell_t 输入 tanh 函数中，与输出门结果 o_t 相乘，将结果作为当前 PI-LSTM 中朴素 LSTM 输出的结果 $\boldsymbol{Y}_{\text{vanilla\_}t}$:

$$\boldsymbol{Y}_{\text{vanilla\_}t}=o_t\cdot\tanh(\text{Cell}_t)。\tag{14}$$

我们设置前序信息包含的数量为 2。句首单词往往起关键性的作用，上述设置能够最大程度地保留开头两个单词的原有信息；同时确保真实地反映其后文本与前文本的关系。前序信息包含的数量越大，对其后文本的前序解码信息负向加成越大。如果当前计算位置 $t\leq 2$，解码操作将仅使用朴素 LSTM 作为计算单元，即最终解码的输出 $\boldsymbol{Y}_t$ 等同于朴素 LSTM 的输出 $\boldsymbol{Y}_{\text{vanilla\_}t}$；若 $t>2$，解码操作将使用完

整的PI-LSTM作为计算单元，即对朴素LSTM的输出 $\boldsymbol{Y}_{\text{vanilla\_}t}$ 进行二次更新。

在获得朴素LSTM输出 $\boldsymbol{Y}_{\text{vanilla\_}t}$ 的同时，解码器将前序解码的输出 $\boldsymbol{Y}_{t-2}$ 和上下文向量 Context_t 送入LSTM单元，计算出前序补充信息 P_t:

$$P_t = \text{LSTM}(\boldsymbol{Y}_{t-2}, \text{Context}_t), \tag{15}$$

使用权重 $\boldsymbol{W}_{\text{pre}}$ 对朴素LSTM的输出 $\boldsymbol{Y}_{\text{vanilla\_}t}$ 进行二次更新:

$$\boldsymbol{Y}_t = (1-\boldsymbol{W}_{\text{pre}})\cdot \boldsymbol{P}_t + \boldsymbol{W}_{\text{pre}}\cdot \boldsymbol{Y}_{\text{vanilla\_}t}。 \tag{16}$$

PI-LSTM将更多的前序解码信息集成到当前的解码操作中，使得最终输出值 $\boldsymbol{Y}_t$ 能够真实地反映当前单词与先前文本的紧密关系。前序补充信息 P_t 可以纠正错误的语法结构，并在预测语法树生成中发挥积极的作用。

3 实验结果分析

3.1 数据集

本研究使用HS (Hearthstone)[1]和Django[22]数据集，其输入序列如表1所示。HS和Django是代码生成模型中常用的数据集。HS数据集是实现纸牌游戏的Python类的集合，其输入是关于卡牌的半结构化描述(如卡名、描述和费用等属性)，序列长度相对较短。Django数据集是一个关于Django Web框架的代码行，每行代码都有自然语言描述的注释，比HS具有更长的输入语句，且涵盖更多的实际用例。

3.2 评估标准

我们选择准确率、双语评估研究指标(BLEU)和训练时间作为评估标准。准确率是生成语句与目标语句中相同单词数在句子长度中的占比。由于代码生成是相对复杂的问题，仅靠准确率并不能体现出模型的优劣性，所以采用双语评估研究指标(BLEU)。BLEU常用于机器翻译领域，用于衡量生成文本与目标文本的相似性，BLEU比准确率更具说服力。本文实验中，使用bleu-4[23]作为BLEU指标。bleu-4广泛用于大多数模型研究中，例如SMT模型[2]、ASN模型[8]和CNN模型[9]。此外，我们使用训练时间作为评估指标的一部分(单位为分钟)。

表1　数据集Django和HS的输入序列

Table 1　Input sequence of Django and HS

数据集	输入序列
Django	get translation function attribute of the object t, call the result with an argument eol message, substitute the result for result.
HS	Maexxna 6 2 8 Destroy any minion damaged by this minion. Legendary

3.3 参数设置

本文实验在GeForce 1070-ti显卡和Ubuntu 18.04 LST系统中进行，使用Adam优化器[24]，将可选注意力机制的 K 值预设为14。在Django数据集上，epoch为50；在HS数据集上，epoch为200。dropout为0.2。编码器GRU和解码器PI-LSTM的隐藏层数均为128。

3.4 对比实验

实验结果如表2所示。我们在同一数据集上选择11个经典模型进行比较：1) 基于Seq2Seq框架的神经机器翻译模型(NMT)模型[4]；2) 生成多层树代码逻辑形式的Seq2Tree模型及其改进模型；3) Seq2-Tree-UNK[5]；4) 使用指针网络的潜伏预测器网络模型(LPN)[1]；5) 基于语法的Seq2Seq神经网络模型(SNM)[7]；6) 抽象语法网络(ASN)模型及其改进；7) ASN+SUPATT[8]；8) 基于语义分析和代码生成的抽象解析器TRANX[9]；9) 基于Transformer[10]架构并优化捕获长依赖性语句的TreeGen[11]模型；10) 基于TreeGen的Self Attention消融模型；11) 基于语法结构的CNN解码器模型[12]。

从表2可以看出，本文模型在HS数据集的准确率和BLEU得分优于其他模型。与SNM模型相比，本文模型的准确率提升11.1%，BLEU分数提升11.7分，并且比其他模型提升至少7.9分。与基于Transformer的TreeGen模型相比，本文模型的准确率稍逊一筹，但双语评估标准分数高出近7分。在Django数据集上，我们选取5个模型，实验结果与HS数据集类似，本文模型也显示最佳效果，准确率比SNM模型提高4.1%，BLEU分数提高2.9分，且比解析器TRANX模型的BLEU分数提高2分。实验结果证明，选择性局部注意力机制和PI-LSTM在提高准确率和BLEU方面起关键作用。与SNM模型使用的软注意力机制相比，选择性局部注意力机制有效地提高了HS数据集上某些长句子的上下文紧密度，使BLEU值得到进一步改善。PI-LSTM通过对输出的二次更新，提高了基于SNM模型的准确率。当使用PI-LSTM作为解码器单元时，与仅使用LSTM相比，计算得到的向量包含更多的前导信息，加强了单词之间的联系(尤其在Django数据集上的长序列中)，大大地提高了准确率和BLEU。

表 2 本文模型与 11 个经典模型的对比实验结果

Table 2 Experiment results of proposed model and the other 11 models

模型	Django			HS		
	双语评估标准	准确率/%	训练时间/min	双语评估标准	准确率/%	训练时间/min
NMT	63.4	41.5	–	60.4	1.5	–
SEQ2TREE	44.6	28.9	–	53.4	1.5	–
SEQ2TREE-UNK	58.2	39.4	–	62.8	13.6	–
LPN	77.6	62.3	–	67.1	6.1	–
SNM	84.5	71.6	341	75.8	16.2	376
ASN	–	–	–	77.6	18.2	–
ASN+SUPATT	–	–	–	79.2	22.7	–
TRANX	–	73.7	–	–	–	–
TREEGEN-Structural	–	–	–	80.8	**33.3**	–
TREEGEN-SelfAttention	–	–	–	81.0	28.8	–
CNN	–	–	–	79.6	27.3	–
本文模型	**87.4**	**75.7**	**297**	**87.5**	27.3	**198**

说明: 粗体数字表示效果最佳, 下同。

在训练时间方面, 由于服务器的限制, 我们只有 SNM 模型的结果。在 HS 数据集上, 本文模型的训练时间为 198 分钟, 仅为 SNM 模型训练时间(376 分钟)的一半; 在 Django 数据集上的训练时间为 297 分钟, 为 SNM 训练时间(341 分钟)的 87%。GRU 单元简化了 LSTM 中的门控运算, 减少计算时间, 且对精度的负面影响很小。

3.5 消融实验

为了评估本文模型的效果, 我们进行消融实验。根据模型的结构, 分别进行框架消融实验和注意力消融实验, 以便验证模型各部分的必要性。

3.5.1 框架消融实验

框架消融测试用于验证 GRU 的有效性, 实验中, 替换了编码器和解码器的计算单元, 结果如表 3 所示。

用 BiLSTM 替换编码器单元, 用朴素 LSTM 替换解码器单元, 并将注意力机制(此处使用软注意力)视为无关变量。鉴于 GRU 在绝大多数情况下仅减少计算时间, 对准确率的影响不大, 因此对 GRU-LSTM 进行实验意义不大。框架消融实验中 3 个模型分别为 BiLSTM-LSTM, BiLSTN-PI-LSTM 和 GRU-PI-LSTM。

与 BiLSTM-LSTM 相比, GRU-PI-LSTM 表现出更好的效果。在 HS 和 DJANGO 数据集上, 准确率提高 2.7%~5%, BLEU 得分提高 1.3%~1.5%。除 GRU-PI-LSTM 的 BLEU 低于 BiLSTN-PI-LSTM 外, 其他评估标准也有所改进。HS 数据集的训练时间减少 178 分钟, 仅为 BiLSTM-LSTM 的一半, Django 数据集则减少 42 分钟。

与 BiLSTM 相比, GRU 简化了计算过程, 将训练时间缩短约 50%。在使用包含前序信息的长短期记忆网络 PI-LSTM 后, GRU 对准确率的负面影响可被忽略, PI-LSTM 使更多的前序解码信息参与解码, 提高了准确率和 BLEU 得分。因此, GRU 和 PI-

表 3 消融实验结果

Table 3 Results of ablation test

消融实验	模型	Django			HS		
		双语评估标准	准确率/%	训练时间/min	双语评估标准	准确率/%	训练时间/min
框架消融实验	BiLSTM-LSTM	84.5	71.6	341	75.8	16.2	376
	BiLSTM-PI-LSTM	80.6	73.7	322	**83.8**	**21.2**	352
	GRU-PI-LSTM	**86.8**	**74.3**	**299**	**81.9**	**21.2**	**198**
注意力消融实验	GRU-PI-LSTM-Soft	86.8	74.3	299	81.9	21.2	**198**
	本文模型	**87.4**	**75.7**	**297**	**87.5**	**27.3**	**198**

LSTM 的组合显示出相对好的效果。

3.5.2 注意力消融实验

消融实验用于验证选择性局部注意力机制对提高准确率和BLEU的积极影响。以本文模型为基础，在注意力部分消融模型中，将选择性局部注意力替换为软注意力，该模型称为 GRU-PI-LSTM-SOFT (等同于框架消融实验中的GRU-PI-LSTM)。实验结果如表 3 所示。

从实验结果看出，与 GRU-PI-LSTM-SOFT 相比，本文模型显示出更好的效果，准确率在 Django 数据集上提高 1.4％，在 HS 数据集上提高 6.1％；BLEU 分数在 Django 数据集上提高 0.6 分，在 HS 数据集上提高 5.6 分。注意力消融测试表明，选择性局部注意力机制的 BLEU 得分均超过 85，有助于提高生成代码的准确率，还能使 HS 数据集上的长句子得到更好的理解以及输出。注意力消融实验中各模型的训练时间相近，证实注意力机制不对训练时间产生影响。

4 结语

本文提出一种基于语法的代码生成模型。该模型基于使用 Seq2Seq 框架、GRU 编码器和 PI-LSTM 解码器的语法结构，解决了在解码中未充分包含预定信息的问题，并纠正了注意力机制的偏重，更适合训练大规模的预处理数据。与其他 11 个模型和 3 个消融模型相比，本文模型在准确率、BLEU 评分和训练时间方面具有很大的优势。本文模型有望实现自动编程和在线代码纠正等需求任务并可用于智能交互式设备自更新和 AI 机器人等现实场景。在未来的工作中，我们将尝试使用双向 GRU[25]作为编码器计算单元，希望可以通过紧密编码，再次提高模型的准确率和 BLEU 分数。

致谢　研究工作得到华南师范大学计算机学院朱佳教授、汤庸教授以及广州市大数据智能教育重点实验室的支持，表示衷心感谢。

参考文献

[1] Ling W, Grefenstette E, Hermann K M, et al. Latent predictor networks for code generation [EB/OL]. (2016–06–08)[2020–05–04]. https://arxiv.org/abs/1603.06744

[2] Brown P F, Cocke J, Della Pietra S A, et al. A statistical approach to machine translation. Computational Linguistics, 1990, 16(2): 79–85

[3] Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks // Advances in Neural Information Processing Systems. Cambridge, MA, 2014: 3104–3112

[4] Mandal S, Naskar S K. Natural language programing with automatic code generation towards solving addition-subtraction word problems // Proceedings of the 14th International Conference on Natural Language Processing. Venice, 2017: 146–154

[5] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780

[6] Dong L, Lapata M. Language to logical form with neural attention [EB/OL]. (2016–06–06) [2020–05–04]. https://arxiv.org/abs/1601.01280

[7] Yin P, Neubig G. A syntactic neural model for general-purpose code generation [EB/OL]. (2017–04–06)[2020–05–04]. https://arxiv.org/abs/1704.01696

[8] Rabinovich M, Stern M, Klein D. Abstract syntax networks for code generation and semantic parsing [EB/OL]. (2017–04–25)[2020–05–04]. https://arxiv.org/abs/1704.07535

[9] Yin P, Neubig G. TRANX: a transition-based neural abstract syntax parser for semantic parsing and code generation [EB/OL]. (2018–10–05) [2020–05–04]. https://arxiv.org/abs/1810.02720v1

[10] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need // Advances in Neural Information Processing Systems. Red Hook, 2017: 5998–6008

[11] Sun Z, Zhu Q, Xiong Y, et al. TreeGen: a tree-based transformer architecture for code generation // Association for the Advancement of Artificial Intelligence. New York, 2020: 8984–8991

[12] Sun Z, Zhu Q, Mou L, et al. A grammar-based structural CNN decoder for code generation // Proceedings of the AAAI Conference on Artificial Intelligence. New York, 2019, 33: 7055–7062

[13] Chung J, Gulcehre C, Cho K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling [EB/OL]. (2014–12–11)[2020–05–04]. https://arxiv.org/abs/1412.3555

[14] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation [EB/OL]. (2014–09–03)[2020–05–04]. https://arxiv.org/abs/1406.1078

[15] Schuster M, Paliwal K K. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 1997, 45(11): 2673–2681

[16] Joshi A K, Levy L S, Takahashi M. Tree adjunct grammars. Journal of Computer and System Sciences, 1975, 10(1): 136–163

[17] Schwarz C. Automatic syntactic analysis of free text. Journal of the American Society for Information Science, 1990, 41(6): 408–417

[18] Saul L K, Jaakkola T, Jordan M I. Mean field theory for sigmoid belief networks. Journal of Artificial Intelligence Research, 1996, 4: 61–76

[19] Luong M T, Pham H, Manning C D. Effective approaches to attention-based neural machine translation [EB/OL]. (2015–09–20)[2020–05–04]. https://arxiv.org/abs/1508.04025

[20] Mikolov T, Kombrink S, Burget L, et al. Extensions of recurrent neural network language model // IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Prague, 2011: 5528–5531

[21] Xu K, Ba J, Kiros R, et al. Show, attend and tell: Neural image caption generation with visual attention // International Conference on Machine Learning. Lille, 2015: 2048–2057

[22] Oda Y, Fudaba H, Neubig G, et al. Learning to generate pseudo-code from source code using statistical machine translation // 2015 30th IEEE/ ACM International Conference on Automated Software Engineering (ASE). Lincoln, 2015: 574–584

[23] Papineni K, Roukos S, Ward T, et al. BLEU: a method for automatic evaluation of machine translation // Proceedings of the 40th annual Meeting of the Association for Computational Linguistics. Philadelphia, PA, 2002: 311–318

[24] Kingma D P, Ba J. Adam: A method for stochastic optimization [EB/OL]. (2017–01–30) [2020–05–04]. https://arxiv.org/abs/1412.6980

[25] Ortega-Bueno R, Rosso P, Pagola J E M. UO_UPV$_2$ at HAHA 2019: BiGRU neural network informed with linguistic features for humor recognition // Proceedings of the Iberian Languages Evaluation Forum (IberLEF'19), Co-Located with the 35th Conference of the Spanish Society for Natural Language Processing (SEPLN'19). Bilbao, 2019: 212–221