计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201100090

# 基于卷积神经网络的代码注释自动生成方法

彭 斌 李 征 刘 勇 吴永豪
北京化工大学信息科学与技术学院 北京 100029
(1252031372@qq.com)

**摘 要** 自动化代码注释生成技术通过分析源代码的语义信息生成对应的自然语言描述文本,可以帮助开发人员更好地理解程序,降低软件维护的时间成本。大部分已有技术是基于递归神经网络(Recurrent Neural Network,RNN)的编码器和解码器神经网络实现的,但这种方法存在长期依赖问题,即在分析距离较远的代码块时,生成的注释信息的准确性不高。为此,文中提出了一种基于卷积神经网络(Convolutional Neural Network,CNN)的自动化代码注释生成方法来缓解长期依赖问题,以生成更准确的注释信息。具体而言,通过构造基于源代码的 CNN 和基于 AST 的 CNN 来捕获源代码的语义信息。实验结果表明,与 DeepCom 和 Hybrid-DeepCom 这两种最新的方法相比,在常用的 BLEU 和 METEOR 两种评测指标下,所提方法能更好地生成代码注释,且执行时间更短。

**关键词**:程序理解;代码注释生成;卷积神经网络;长短期记忆网络
**中图法分类号** TP311

# Automatic Code Comments Generation Method Based on Convolutional Neural Network

PENG Bin,LI Zheng,LIU Yong and WU Yong-hao
College of Information Science and Technology,Beijing University of Chemical Technology,Beijing 100029,China

**Abstract** Automatic code comment generation technology can analyze the semantic information of source code and generate corresponding natural language descriptions,which can help developers understand the program and reduce the time cost during software maintenance. Most of the existing technologies are based on the encoder and decoder model of the recurrent neural network (RNN). However,this method suffers from long-term dependency problem,which means it cannot generate high-quality comments when analyzing far-away code blocks. To solve this problem,this paper proposes an automatic code comment generation method,which uses the convolutional neural network(CNN) to alleviate the inaccurate comments information caused by the long-term dependence problem. More specifically,this paper uses two CNNs,one source-code based CNN and one AST-based CNN,to capture source code's semantic information. The experimental results indicate that,compared to the two most recent methods, DeepCom and Hybrid-DeepCom,the method proposed in this paper generates more useful code comments and takes less time to execute.

**Keywords** Program comprehension,Code comment generation,Convolutional neural network,Long short-term memory network

## 1 引言

在软件开发和维护过程中,开发人员会耗费接近 60%的时间来理解源代码[1]。代码注释指通过自然语言描述一段代码,是开发人员理解软件代码最直观、有效的方法。高质量的代码注释在软件维护和重用中起着重要的作用。然而,由于项目进度紧迫[2],或者某些注释在软件更新过程中未及时更新而过时,许多软件项目没有提供完整、有效的注释,降低了程序的可读性和可维护性。此外,编写代码注释在软件开发中也是一项繁琐且耗时的任务,需要开发人员付出大量的时间成本[3]。

自动化代码注释生成技术[4-5]是解决这些问题的有效方法。给定代码片段,自动化代码注释生成技术可以为该代码片段自动化地生成对应的自然语言描述。当前的代码注释生成方法可以分为两类:基于关键词模板的方法和基于人工智能(Artificial Intelligence,AI)的方法。基于关键词模板的方法通常预定义一组自然语言模板,并用目标代码段的内容填充这些模板[6-7]。尽管基于关键词模板的方法已经取得了重大进展,但是这些基于模板的方法仍然存在局限性[8]。例如,人工构建模型需要耗费大量的时间,并且这些方法对模板之外的代码很难产生有效的注释。

基于 AI 方法的研究更多地倾向于将编码器、解码器框

到稿日期:2020-11-11 返修日期:2021-04-09
基金项目:国家自然科学基金(61902015,61872026)
This work was supported by the National Natural Science Foundation of China(61902015,61872026).
通信作者:吴永豪(appmlk@outlook.com)

架[9]应用于代码注释生成。在此框架中，研究人员通常将循环神经网络（RNN）用作编码器和解码器。RNN 应用于代码注释生成时，将源代码作为输入序列，并生成代码注释作为输出。但是，源代码作为结构严谨的文本，包含丰富的代码结构信息，对程序的语义理解很重要[10]。为了解决这个问题，最新的研究方法基于代码的抽象语法树（Abstract Syntax Tree，AST）来提取代码结构信息并生成注释[2,8,11-13]。

但是，根据我们的分析，这些方法仍然存在两个问题。第一个问题是长期依赖问题，即彼此相距很远的几个代码块之间的信息很难被总结；第二个问题是语义编码问题。现有的基于 AST 的方法通过遍历将 AST 序列转化为令牌序列，并通过 RNN 提取特征。但是，RNN 是为编码序列而设计的，无法很好地捕获结构语义。

本文提出了一种基于语义卷积神经网络（CNN）的方法，用于自动化生成 Java 函数的代码注释。Java 函数是 Java 编程语言的基本功能单元，我们仅选择 Javadoc 的第一句作为相应 Java 函数的注释，因为它通常根据 Javadoc 指南对 Java 函数的功能进行正式的描述。由于 CNN 能通过滑动窗口有效地捕获不同块之间的代码特征[10]，因此本文方法使用 CNN 来有效捕获不同代码块之间的特征，从而缓解长期依赖问题[14]。此外，为了同时提取代码的文本信息和结构信息，本文方法包含两部分组件，分别为基于源代码文本的 CNN 和基于 AST 的 CNN。同时，本文提出了一种面向约简的抽象语法树结构遍历方法，用于对源代码的 AST 进行特征提取。

本文方法使用两个 CNN 来编码源代码的语义信息。一个 CNN 从代码令牌中提取文本词法信息，另一个 CNN 从 AST 中提取代码结构信息。在完成编码之后，我们使用具有注意力机制的长短期记忆神经网络（Long Short-Term Memory Model，LSTM）作为解码器并生成代码注释。

为了评估本文方法的有效性，我们在两个大型 Java 数据集上进行了实验[15-16]。实验结果表明，与当前最新的研究方法相比，本文方法在 BLEU[17] 和 METEOR[18] 指标方面实现了更好的性能。

## 2 相关概念与基本理论

本文使用的神经网络模型主要分为两类：文本卷积神经网络和长短期记忆网络[19-20]。

### 2.1 文本卷积神经网络

卷积神经网络（CNN）是深度学习领域中的代表性神经网络之一，研究人员采用基于 CNN 的方法在图像分析[21]和计算机视觉[22]领域取得了许多研究成果。CNN 模型在自然语言处理（Natural Language Processing，NLP）的研究中被广泛使用，并且在句子分类[23]和网页搜索[24]任务中可以取得很好的效果。

基于 CNN 的技术通常包括许多卷积层和池化层。在卷积层中，CNN 将计算输入数据区域和权重矩阵之间的点积。过滤器将在整个输入数据上滑动，并重复相同的点积计算操作。

池化层中将进行池化操作，其中平均池化和最大池化是两种常用的池化方法。最大池化层将使用输入区域中最大的特征点，而平均池化层将使用输入区域的平均特征点。在 CNN 中，池化操作可以在不减小网络深度的情况下缩小空间尺寸。

近年来，研究人员已经提出了众多基于 CNN 的方法[25]，并且不同的方法具有处理不同问题的优势。由于篇幅所限，我们以文本神经网络（Text Convolutional Neural Network，TextCNN）为例介绍 CNN 的基本思想。TextCNN[23] 是一种 CNN，旨在处理与文本相关的问题。

图 1 给出了 TextCNN 的一个卷积内核的卷积层，该卷积层用于提取输入数据的矢量特征。

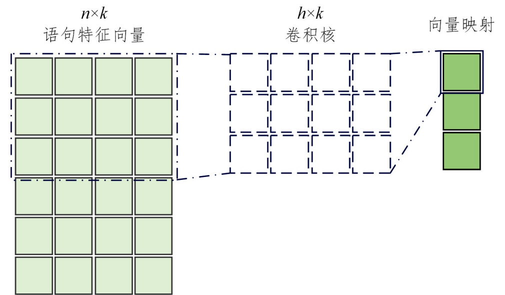



图 1 TextCNN 网络结构

Fig. 1 TextCNN network structure

在 TextCNN 中，长度为 $n$ 的句子向量可以表示为 $\boldsymbol{x}_1,\cdots,\boldsymbol{x}_n$，其中 $\boldsymbol{x}_i$ 是与句子中第 $i$ 个单词相对应的向量。此卷积可以通过式(1)来计算：

$$c_i=f(W\cdot \boldsymbol{x}_{i:i+h-1}+b) \tag{1}$$

其中，$W$ 是卷积核，用于提取 $h$ 个相邻向量的特征，$W$ 的大小为 $h\times k$，$k$ 具有与输入向量相同的维度，$h$ 是滑动窗口大小的集合；$\boldsymbol{x}_{i:i+h-1}$ 表示相邻的 $k$ 个向量；$b$ 是偏置；$f$ 是非线性函数；$c_i$ 是从相邻的 $k$ 个向量中提取的特征。

### 2.2 长短期记忆网络

长短期记忆网络被广泛用于代码注释研究。标准 RNN 将序列数据作为输入，并在序列的演化方向上进行递归。最后，标准 RNN 将链中的所有节点（循环单元）连接起来。具体来说，RNN 能够逐一读取句子中的单词，并预测可能的后续单词。由于 RNN 具有处理序列数据的能力，因此该方法可以在某种程度上处理结构信息。

但是，标准 RNN 模型在反向传播过程中，可能会出现梯度爆炸或梯度消失，尤其当输入序列中存在长期依存关系时。为了缓解这个问题，研究人员改进了标准 RNN 模型，并提出了长短期记忆神经网络。

LSTM 用 3 个门来控制存储单元的状态。这 3 个门分别为遗忘门、输入门和输出门，其中遗忘门可以决定应丢弃或保留哪些信息；输入门用于更新设备状态；输出门可以确定下一个隐藏状态的值，该值包含先前输入的相关信息。图 2 给出了一个 LSTM 的典型存储单元，其中 $\boldsymbol{C}_{t-1}$ 是先前状态的内容向量，而 $\boldsymbol{h}_{t-1}$ 是先前状态的隐藏状态。

图 2 中，$\delta$ 是一个 S 型的处理单元，它根据 $\boldsymbol{h}_{t-1}$ 和 $\boldsymbol{x}_t$ 提供的信息输出一个介于 0 和 1 之间的向量。$\boldsymbol{f}_t$ 由“遗忘门”生成，该门决定应丢弃或保留哪些信息。$\boldsymbol{i}_t$ 由输入门生成，它决

定在单元状态下应存储哪种新信息。$\hat{C}_t$ 是在当前状态下添加的新信息。$o_t$ 由输出门生成，该门决定应为输出设置哪种值。$C_t$ 是当前状态的内容向量。

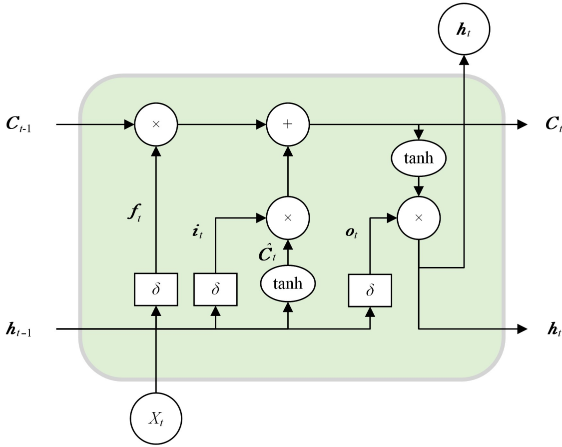



图 2　长短期记忆网络结构

Fig. 2　Long short-term memory structure

## 3　基于卷积神经网络的代码注释生产方法

### 3.1　方法框架

本文方法的框架如图 3 所示，该方法包含 3 个主要步骤：数据预处理、信息抽取和代码注释生成。

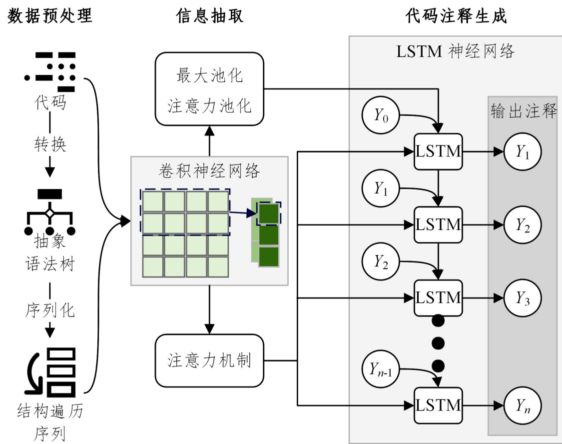



图 3　本文方法的框架图

Fig. 3　Framework of our proposed method

与自然语言的文本数据不同，源代码是一种结构化数据，并且相应的词汇量非常大[2]。本文方法使用卷积神经网络(CNN)提取源代码的语义信息。在数据预处理步骤中，我们将源代码转换为代码令牌向量和 AST 向量。为了缓解词汇量过大的问题，我们使用驼峰式转换将标识符和 AST 的叶节点拆分为多个单词。

我们使用两个 CNN 从源代码中提取语义信息。最后，在代码注释生成步骤中，本文使用具有注意力机制的 LSTM 来解码语义信息并生成注释。

### 3.2　数据预处理

由于 CNN 模型的输入数据格式为向量，因此我们需要在数据预处理步骤中将所有输入数据转换为向量。

#### 3.2.1　源代码预处理

源代码由关键字、运算符、标识符和符号组成，标识符中包含大量的词汇信息。这些词汇信息可通过神经网络算法实现自动化提取。

为了构造神经网络算法的输入序列，我们使用了 javalang 工具[16]将源代码转换为令牌。此外，为了解决词汇量过大的问题，我们使用驼峰式转换将每个标识符拆分为多个单词，并将所有代码令牌都转换为小写。对于代码令牌的序列数据，令 $\boldsymbol{X}_s^{(\text{code})}=\langle \boldsymbol{x}_{s1}^{(\text{code})},\cdots,\boldsymbol{x}_{sn}^{(\text{code})}\rangle$，然后使用词嵌入将其转换为向量 $\boldsymbol{X}^{(\text{code})}=\langle \boldsymbol{x}_1^{(\text{code})},\cdots,\boldsymbol{x}_n^{(\text{code})}\rangle$，其中 $\boldsymbol{x}_i^{(\text{code})}$ 表示第 $i$ 个代码令牌 $\boldsymbol{x}_i$ 的 $k$ 维向量。因此，$\boldsymbol{X}^{(\text{code})}$ 是一个 $n\times k$ 维的向量。

#### 3.2.2　面向约简的抽象语法树结构遍历

虽然代码令牌可用于提取代码的词法信息，但是其中不包含代码的结构信息。因此，本文基于 AST 进一步提取源代码的结构信息。

我们使用 javalang 工具解析源代码以生成其 AST。除此之外，本文还提出了一种面向约简的抽象语法树结构遍历(Reduction-oriented Abstract Syntax Tree Structure Traversal，RAST)方法来提取 AST 特征。本文方法基于 Hu 等[2]提出的基于结构的遍历方法(Structure-Based Traversal，SBT)，旨在在语法信息提取过程中最大程度地保留源代码的结构信息。

尽管 SBT 方法在代码注释生成任务上取得了优秀的效果，但是 SBT 生成的原始序列包含了太多重复内容。而且，SBT 使用一对括号来表示树结构，这不利于编码结构信息。SBT 序列还包含源代码的标识符，因此它也包含大量的词汇信息。

为了解决输入数据冗余的问题，我们提出了一种新颖的 AST 遍历方法，称为面向约简的抽象语法树结构遍历(RAST)方法，该方法可以更好地编码代码的结构信息。图 4 使用一个简单的示例来说明 RAST 方法如何遍历 AST。

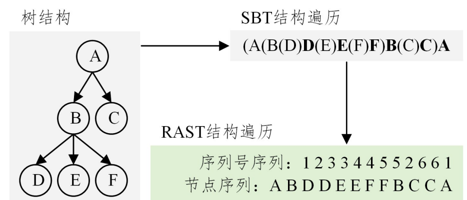



图 4　RAST 示例

Fig. 4　RAST example

首先，我们使用 SBT 遍历 AST 以生成 SBT 序列。然后，我们通过 AST 先序遍历的序列号替换 SBT 序列中的括号，并将 SBT 序列分为两部分：序列号序列和节点序列。最后，为了解决词汇量过大的问题，我们使用驼峰式转换来拆分节点序列的值，并将原始节点对应的序列号赋值给每个拆分后的词。例如，节点的值为“isLayered”，其序列号为“3”。将节点的值“isLayered”拆分为“is Layered”，并将序列号“3”复制为“3 3”，然后将“is”和“Layered”的对应序列都赋值为“3”。根据以上分析可以看出，通过使用 RAST 方法，可以从生成的序列中明确地还原 SBT 序列。因此，我们对原始 SBT 的改进仍然可以保留所有结构信息。

此外,为了构造 CNN 模型的输入,我们使用 CNN 对 RAST 生成的序列号序列和节点序列进行更进一步的编码。具体来说,对于序列号序列 $\boldsymbol{X}_s^{(serial)}=\langle \boldsymbol{x}_{s1}^{(serial)},\cdots,\boldsymbol{x}_{sm}^{(serial)}\rangle$ 和节点序列 $\boldsymbol{X}_s^{(node)}=\langle \boldsymbol{x}_{s1}^{(node)},\cdots,\boldsymbol{x}_{sm}^{(node)}\rangle$,我们使用词嵌入技术将它们转换为相应向量 $\boldsymbol{X}^{(serial)}=\langle \boldsymbol{x}_1^{(serial)},\cdots,\boldsymbol{x}_m^{(serial)}\rangle$ 和 $\boldsymbol{X}^{(node)}=\langle \boldsymbol{x}_1^{(node)},\cdots,\boldsymbol{x}_m^{(node)}\rangle$,其中 $\boldsymbol{x}_i^{(serial)}$ 和 $\boldsymbol{x}_i^{(node)}$ 均为 $k$ 维向量。为了提取其特征,我们使用 CNN 将两个序列整合到一个序列中得到 $\boldsymbol{X}^{(RAST)}=\langle \boldsymbol{x}_1^{(RAST)},\cdots,\boldsymbol{x}_m^{(RAST)}\rangle$。该卷积的计算如下:

$$x_i^{(RAST)}=\mathrm{ReLU}(W^{(RAST)}[\boldsymbol{x}_i^{(node)};\boldsymbol{x}_i^{(serial)}]) \tag{2}$$

其中,$\boldsymbol{W}^{(RAST)}$ 是基于 RAST 的 CNN 内核的权重,而 $\boldsymbol{x}_i^{(RAST)}$ 是 $\boldsymbol{x}_i^{(node)}$ 和 $\boldsymbol{x}_i^{(serial)}$ 通过卷积提取的向量;ReLU(Rectified Linear Unit)是神经网络中广泛使用的非线性激活函数。

### 3.3 信息抽取

本文方法使用 CNN 模型提取语义信息,我们将在本节详细介绍此步骤的内容。

#### 3.3.1 卷积神经网络模块

本文方法使用两个基于 CNN 的模型来捕获源代码的语义信息:一个从代码令牌中提取词法信息,另一个从 AST 中提取语法信息。

代码令牌的向量可以表示为 $\boldsymbol{X}^{(code)}=\langle \boldsymbol{x}_1^{(code)},\cdots,\boldsymbol{x}_n^{(code)}\rangle$,其中 $\boldsymbol{x}_i^{(code)}$ 是 $k$ 维输入向量,$n$ 表示向量的数目。然后,我们使用一系列卷积层提取其特征 $\boldsymbol{Y}^{(code)}=\langle \boldsymbol{y}_1^{(code)},\cdots,\boldsymbol{y}_n^{(code)}\rangle$。该卷积的计算方式如下:

$$\boldsymbol{y}_i^{(code)}=f(\boldsymbol{W}^{(code)}\cdot \boldsymbol{x}_{i:i+h-1}^{(code)}) \tag{3}$$

其中,$\boldsymbol{W}^{(code)}$ 是卷积内核,它是可训练的参数,并在训练时进行更新,大小为 $h\times k$,$k$ 表示与输入向量的相同维数,$h$ 为滑动窗口的大小;$\boldsymbol{x}_{i:i+h-1}^{(code)}$ 表示相邻的 $k$ 个向量;$f$ 是非线性函数(如 ReLU 函数);$\boldsymbol{y}_i^{(code)}$ 是从 $\boldsymbol{x}_{i:i+h-1}^{(code)}$ 中提取的特征向量。

本文方法中的卷积操作与 TextCNN[23] 略有不同。为了在卷积前后保持相同的特征向量维数,我们对每个卷积层使用 $k$ 个卷积核。为了在卷积前后保持相同数量的特征向量,我们向输入向量中填充数字“0”。因此,每个卷积层提取的特征向量具有与原始输入向量相同的维数,本文方法可以使用残差连接。为了解决网络退化的问题,即随着神经网络层深度的增加,准确率先上升直至饱和,然后继续增加深度会导致准确率下降,我们在激活功能之前使用残差连接。

与代码令牌类似,对于 RAST 信息的向量 $\boldsymbol{X}^{(RAST)}$,我们使用相同的卷积运算来提取特征向量 $\boldsymbol{Y}^{(RAST)}=\langle \boldsymbol{y}_1^{(RAST)},\cdots,\boldsymbol{y}_m^{(RAST)}\rangle$。

最后,为了获得语义向量,我们使用 Tensorflow 提供的 concat 函数来连接 $\boldsymbol{Y}^{(code)}$ 和 $\boldsymbol{Y}^{(RAST)}$。得到语义向量表示为 $\boldsymbol{Y}^{(sem)}$,是一个 $(n+m)\times k$ 维的向量。

#### 3.3.2 池化操作

本文方法使用池化操作构造 LSTM 初始状态。LSTM 初始状态包括上下文向量(***c_state***)和隐藏状态(***h_state***)。为了用语义信息构造内容向量,我们首先对从代码中提取的特征 $\boldsymbol{Y}^{(code)}$ 应用 Max 池化[25] 得到控制向量 ***code_pooling***。然后我们使用 ***code_state*** 和 $\boldsymbol{Y}^{(RAST)}$ 进行注意力池化获得 ***c_state***,目的是使 ***c_state*** 包含语义信息。类似地,特征 $\boldsymbol{Y}^{(RAST)}$ 用于 Max 池化以获得控制向量 ***RAST_pooling***。然后我们使用 ***code_state*** 和 $\boldsymbol{Y}^{(RAST)}$ 进行注意力池化获得 ***h_state***。最后,我们将 ***c_state*** 和 ***h_state*** 用作 Decoder 中 LSTM 的初始状态。

### 3.4 代码注释生成

在本文方法的第三步中,我们使用具有注意力机制的 LSTM 来解码源代码的语义信息并生成代码注释。

#### 3.4.1 注意力机制

我们使用注意力机制为 CNN 模型提取的每个语义向量分配权重,权重值用于确定此向量对输出目标词的重要性。具体来说,本文方法使用 Bahdanau 等[26] 提出的经典注意力方法作为注意力机制。

注意力机制需要通过式(4)来计算用于预测每个目标词 $y_i$ 的上下文向量 $\boldsymbol{c}_i$:

$$\boldsymbol{c}_i=\sum_{j=1}^{m+n}a_{ij}\boldsymbol{y}_j^{(sem)} \tag{4}$$

其中,$\boldsymbol{y}_j^{(sem)}$ 是 CNN 模型提取的语义向量,$a_{ij}$ 表示 $\boldsymbol{y}_j^{(sem)}$ 的相应注意力权重。

为了计算 $a_{ij}$,我们首先计算一个对齐模型得分 $e_{ij}$ 来衡量每个输入与解码器当前输出的匹配程度,其计算式如下:

$$e_{ij}=a(\boldsymbol{h}_{i-1},\boldsymbol{y}_j^{(sem)}) \tag{5}$$

其中,$a$ 为对齐模型,是一个前馈神经网络。

然后,我们使用 softmax 函数对得分进行归一化以获得注意力权重,具体定义如下:

$$a_{ij}=\frac{\exp(e_{ij})}{\sum_{k=1}^{m+n}\exp(e_{ik})} \tag{6}$$

#### 3.4.2 序列的产生

解码器的目的是对 CNN 模型生成的语义向量进行解码,并生成代码注释 $y_1,\cdots,y_n$。这里,$y_i$ 通过以下公式预测:

$$y_i=P(y_i|y_1,\cdots,y_{i-1},x)=g(y_{i-1},\boldsymbol{h}_i,\boldsymbol{c}_i) \tag{7}$$

其中,$g$ 是随机输出层,用于估计单词 $y_i$ 的概率;$\boldsymbol{c}_i$ 是上下文向量;$\boldsymbol{h}_i$ 是当前隐藏状态。

在图 3 所示的注释生成过程中,我们使用〈START〉作为解码器的第一个输入字($Y_0$),将〈EOS〉作为最后一个单词($Y_n$)。

## 4 实验设置

### 4.1 研究问题

针对生成 Java 注释的准确性和效率,我们通过比较本文方法与最新对照方法来评估本文方法的有效性。具体来说,我们关注以下 3 个研究问题:

RQ1:本文方法的性能能否超越最新的代码注释生成对照方法?

此 RQ 旨在验证本文方法的代码注释生成的有效性。为了回答这个问题,我们进行了一系列的经验研究,并将本文方法与其他最新的代码注释生成对照方法进行了对比,包括 DeepCom,Hybrid-DeepCom。

RQ2:代码和注释长度如何影响本文方法的性能?

此 RQ 旨在调查不同长度的源代码和注释对本文方法的代码注释生成有效性的影响。为了回答这个 RQ，我们收集并分析了使用不同长度的源代码生成注释时的实验结果。

RQ3：本文方法的训练时间能否超越最新的代码注释生成对照方法？

此 RQ 旨在评估不同的自动化代码注释生成方法的训练时长。为了回答这个 RQ，我们对比了不同方法的训练时间消耗。

### 4.2　数据集

本文在两个 Java 数据集上进行实验[15-16]，这两个数据集都是从 GitHub 收集的〈方法，注释〉对，表 1 列出了上述数据集的相关统计信息。其中数据集 1 包括 87 136 对〈方法，注释〉，所有代码的平均长度为 98.8，所有注释长度为 17.70。数据集 2 包括 485 812 对〈方法，注释〉，所有代码的平均长度为 93.93，所有注释长度为 11.24。本文按照 Hu 等[2]的方法预处理数据集，并将 Javadoc 的第一句提取为代码注释，用于解释 Java 方法的功能。Java 方法中的标识符通过驼峰式分割以缓解词汇量不足的问题。

表 1　数据集统计信息

Table 1　Dataset statistics

| 数据信息 | 数据集 1 | 数据集 2 |
|---|---|---|
| 训练集 | 69 708 | 445 812 |
| 验证集 | 8 714 | 20 000 |
| 测试集 | 8 714 | 20 000 |
| 平均代码长度 | 98.8 | 93.93 |
| 平均注释长度 | 17.70 | 11.24 |

### 4.3　评价指标

为了评估本文方法的有效性，我们选择了两个基于机器翻译的指标 BLEU 和 METEOR。这两个评估指标已被广泛应用于以前的代码注释生成研究中。

BLEU 分数是自然语言处理中文本生成任务[27]广泛使用的性能指标，并已用于评估生成的代码注释的质量[2,16,28]。它计算生成的序列和参考序列之间的相似度。BLEU 分数的值介于 0 到 100%之间，BLEU 分数越高，说明生成的序列越接近参考序列。

BLEU 被定义为 n-gram 匹配准确度分数的几何平均值乘以简单的惩罚因子，惩罚因子用于防止生成非常短的句子得到较高的分数。BLEU 通过式(8)来计算：

$$BLEU=BP\cdot(\sum_{i=0}^{n}w_n\log p_n) \tag{8}$$

其中，$p_n$是修改后的 n-gram 精度的几何平均值，而 $w_n$ 是正权重。式(8)中，BP 是简单的惩罚因子，当 $x>r$ 时，$BP=1$；当 $c\leqslant r$ 时，

$$BP=\mathrm{e}^{1-\frac{r}{c}} \tag{9}$$

其中，$c$ 表示生成的序列的长度，$r$ 表示参考序列的长度。

具体来说，我们在本文研究中使用句子级别的 BLEU 作为性能指标，这是已有的代码注释生成研究中广泛采用的选择。

METEOR[18]是一种广泛使用的基于机器翻译的指标。它通过将翻译的序列与参考序列对齐并计算句子级别的相似度得分来评估翻译假设。METEOR 的特点是引入同义词匹配，METEOR 可通过式(10)来计算：

$$METEOR=(1-P_{en})\cdot F_{mean} \tag{10}$$

其中，$P_{en}$ 是惩罚系数，$F_{mean}$ 是查准率和召回率加权调和平均。$P_{en}$ 的计算式为：

$$P_{en}=\gamma(\frac{ch}{m})^{\beta} \tag{11}$$

其中，$ch$ 是块的数量(在本文中，块代表参考句中的几个相邻单词)，$m$ 是匹配项的数量，$\gamma$ 确定最大惩罚($0\leqslant\gamma\leqslant1$)，$\beta$ 决定块与罚分之间的功能关系。$\gamma$ 和 $\beta$ 分别设置为 0.20 和 0.60[29]。

$F_{mean}$ 的计算式为：

$$F_{mean}=\frac{P\cdot R}{\alpha\cdot P+(1-\alpha)\cdot R} \tag{12}$$

其中，$P$ 是查准率，$R$ 是召回率，$\alpha$ 设置为 0.85[29]。

两种基于机器翻译的指标和代码注释生成技术的性能之间的相关性也为正。较高的 BLEU 或 METEOR 值表示较好的代码注释生成技术。

### 4.4　对照方法

本文采用了两种最新的代码注释生成方法[2,16]作为对照方法，并且这两种方法是当前最领先的代码注释生成方法。这些对照方法的详细说明如下。

对照方法 1：DeepCom[2]是一种基于注意力的 Seq2Seq 模型，用于生成 Java 方法的注释。DeepCom 将 AST 作为输入，并使用 SBT 方法将这些 AST 转换为特殊格式的序列。

对照方法 2：Hybrid-DeepCom[16]是基于注意力的 Seq2-Seq 模型的一种变体，用于为 Java 方法生成注释。Hybrid-DeepCom 使用源代码文本及其 AST 结构生成代码注释。

### 4.5　超参数设置

参照 Hu 等[2]的设置，我们分别用特殊标记〈UNM〉和〈STR〉替换源代码中的常数和字符串。代码序列和 RAST 序列的最大长度分别设置为 200 和 300。我们使用特殊符号〈PAD〉填充短序列，较长的序列将被剪切。最大注释长度设置为 30。我们在注释中添加特殊标记〈START〉和〈EOS〉，其中〈START〉是解码序列的开始，而〈EOS〉表示解码序列的结束。根据 Hu 等的设置，两个数据集代码和 RAST 的最大词汇量分别设置成 30 000，30 000，数据集 1 的注释最大词汇量设置成 23428，数据集 2 的最大词汇量设置成 30000。超过最大词汇量的词汇将通过〈UNK〉标签代替。

本文方法的超参数及其设置的描述如下：

(1)本文使用 SGD 算法来训练参数，并且将从训练示例中随机选择给定数量的样本的最小批处理大小设置为 128。

(2)编码器使用 9 层 CNN。所有卷积核的大小均为 4×500，卷积步长设置为 4。

(3)解码器模型使用一层 LSTM，其中隐藏状态为 500 维，并且嵌入单词的维度也为 500。

(4)初始学习率设置为 1.0。我们使用指数衰减法来降低学习率，并将学习率衰减系数设置为 0.99。

(5)梯度剪裁设为 5。本文在训练过程中使用 drop 策略，并将 drop 设置为 0.8。

(6)使用交叉熵最小化作为成本函数。

本文方法基于 python 3.6 和 Tensorflow 5 框架来实现。本文进行的所有实验都在 Linux 服务器(Ubuntu 14.0, Intel Xeon Gold 6240 2.60 GHz, 32 GB Tesla V100 GPU)上运行。

## 5 实验结果

### 5.1 RQ1:本文方法的性能是否超越最新的代码注释生成对照方法?

如第 4.4 节所述,本文使用了两种最先进且经过广泛比较的代码注释生成技术作为对照方法。

为了对比本文方法和其他对照方法的性能,我们使用两个基于机器翻译的指标 BLEU 和 METEOR 来衡量自动生成的注释和手动编写的注释之间的差距。表 2 和表 3 列出了相应的实验结果。从表中可以发现,在两个数据集上,本文方法在两个指标方面均优于两种对照方法。更具体地,在数据集 1 上,本文方法对比 DeepCom 方法,在 BLEU 指标上提高了 4.36%,在 METEOR 指标上提高了 4.57%;对比 Hybrid-DeepCom 方法,在 BLEU 指标上提高了 2.61%,在 METEOR 指标上提高了 1.92%。

表 2 不同方法在数据集 1 上的平均性能

Table 2 Average performance of different methods on dataset 1

(单位:%)

| 方法 | BLEU | METEOR |
|---|---|---|
| DeepCom | 40.51 | 22.21 |
| Hybrid-DeepCom | 42.26 | 24.86 |
| 本文方法 | 44.87 | 26.78 |

表 3 不同方法在数据集 2 上的平均性能

Table 3 Average performance of different methods on dataset 2

(单位:%)

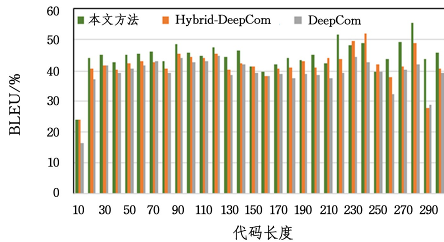
| 方法 | BLEU | METEOR |
|---|---|---|
| DeepCom | 38.22 | 22.63 |
| Hybrid-DeepCom | 39.51 | 24.46 |
| 本文方法 | 45.15 | 28.72 |

在数据集 2 上,本文方法对比 DeepCom 方法,在 BLEU 指标上提高了 6.93%,在 METEOR 指标上提高了 6.09%;对比 Hybrid-DeepCom 方法,在 BLEU 指标上提高了 5.64%,在 METEOR 指标上提高了 4.26%。上述实验结果表明,本文方法可以学习源代码的语义信息,并生成比其他对照方法更高质量的代码注释,证明了本文方法的性能优于两种对照方法。

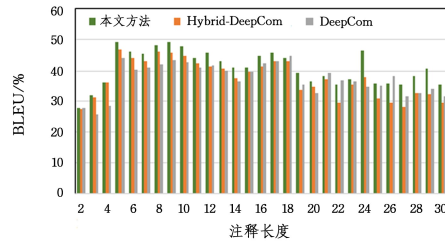
在分析表 2 和表 3 时,我们可以得到更多发现。例如,可以看出 Hybrid-DeepCom 的性能优于 DeepCom,原因是 DeepCom 仅学习语法信息,缺少词法信息,而 Hybrid-DeepCom 学习了语法和词法信息。该实验结果说明,学习代码文本信息有助于提高代码注释生成技术的性能。从两个表中可以看出,本文方法优于 Hybrid-DeepCom,与 Hybrid-DeepCom 不同的是,我们在编码阶段使用了 CNN 来对代码的令牌和 AST 进行编码,而 Hybrid-DeepCom 使用的是 RNN,结果显示 CNN 比 RNN 更适用于带有结构信息的代码分析。

### 5.2 RQ2:代码和注释长度如何影响本文方法的性能?

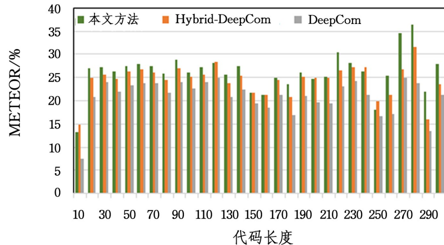
在此 RQ 中,我们需要分析本文方法在不同长度的源代码和注释中的预测准确性。我们继续使用 Hybrid-DeepCom 和 DeepCom 两种对照方法来评估本文方法的性能。图 5 给出了本文方法和其他两种对照方法的平均结果。



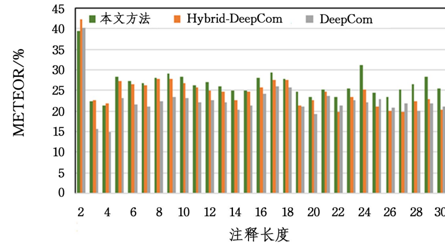

(a)不同代码长度的 BLEU 评估结果




(b)不同注释长度的 BLEU 评估结果




(c)不同代码长度的 METEOR 评估结果




(d)不同注释长度的 METEOR 评估结果

图 5 代码和注释长度对模型性能的影响

Fig. 5 Impact of code and comment length on model performance

图 5(a)和图 5(c)给出了当考虑不同的代码长度时,这 3 种技术在 BLEU 和 METEOR 度量方面的性能变化。当考虑代码长度的影响时,我们发现代码长度的范围太大,并且生成的图像不清晰,很难发现代码长度如何影响模型的性能。因此,我们对代码长度进行采样,每 10 个点进行一次采样。

从图 5(a)和图 5(c)可以看出,在大多数情况下,本文方法的性能都优于两种对照方法。随着代码长度的增加,本文方法的 BLEU 和 METEOR 都将首先增加,然后保持相似的准确性。在图 5(a)和图 5(c)所示的不同代码长度中,本文方法可以在这两个子图中分别有 24 和 23 个点达到最大值,它们的比例分别为 80.00%和 76.67%。

图 5(b)和图 5(d)给出了当考虑不同的注释长度时,这 3

种技术在 BLEU 和 METEOR 度量方面的变化。如图 5(b)和图 5(d)所示，在大多数情况下，本文方法的 BLEU 和 METEOR 得分都最高。由图 5(b)和图 5(d)所示的不同注释长度的影响可知，本文方法可以在这两个子图中的顶部 24 和 26 个案例中获得最佳性能，其比例分别为 82.76% 和 89.66%。更具体地说，随着评论长度的增加，本文方法的 METEOR 得分将首先增加，然后保持相似的准确性；本文方法的 BLEU 得分先增加后降低。当注释仅包含 5 到 10 个单词时，本文方法将获得最高的 BLEU 分数。

### 5.3 RQ3：本文方法的训练时间能否短于最新的代码注释生成对照方法？

神经模型的训练时间是模型的一个重要性能，训练时间短的模型可以节约大量的资源。本节将进一步研究本文方法和其他对照方法的训练时间。此外，为了验证 RAST 的有效性，我们还比较了带有 RAST 序列和带有原始 SBT 序列的训练时间。表 4 列出了这 4 种技术的训练时间。由于神经网络每轮训练的训练时间是相似的，因此我们选择前三轮训练时间进行比较。

表 4　3 轮训练的运行时间比较

Table 4　Comparison of running time of three rounds of training

(单位：s)

| 方法 | 第一轮训练 | 第二轮训练 | 第三轮训练 | 平均 |
|---|---|---|---|---|
| DeepCom | 1100 | 1093 | 1092 | 1095 |
| Hybrid-DeepCom | 1113 | 1108 | 1110 | 1110 |
| 本文提出的 SBT 方法 | 1109 | 1112 | 1113 | 1111 |
| 本文提出的 RAST 方法 | 856 | 858 | 860 | 858 |

如表 4 所列，本文方法(RAST)的训练时间最短。与本文方法(SBT)相比，使用 RAST 可以缩短约 22.77% 的训练时间。表 5 列出了本文提出的 RAST 方法和本文提出的 SBT 方法的相似性能，证明使用 RAST 方法能够在不显著损失性能的前提下提高训练速度。

表 5　RAST 和 SBT 在数据集 1 上的平均性能

Table 5　Average performance of RAST and SBT on dataset 1

(单位：%)

| 方法 | BLEU | METEOR |
|---|---|---|
| 本文提出的 SBT 方法 | 44.49 | 26.52 |
| 本文提出的 RAST 方法 | 44.87 | 26.78 |

如表 4 所列，与 Hybrid-DeepCom 相比，本文方法的训练时间平均缩短了约 21.64%。在 Hybrid-DeepCom 的编码器阶段，其使用一层 GRU 网络，而本文方法(SBT)使用 9 层 CNN 网络，实验结果表明，CNN 比 GRU 需要更少的训练时间。与 DeepCom 相比，本文方法的训练时间缩短了约 22.70%。实验结果表明，CNN 的训练时间优于 LSTM。

**结束语**　本文提出了一种基于神经网络的新技术，用于生成 Java 方法的代码注释。本文方法使用 CNN 缓解了源代码分析的长期依赖性问题，并设计了一些新颖的组件来捕获源代码的语义信息。具体来说，我们设计了一个基于源代码的 CNN 组件来学习代码文本信息，并设计了一个基于 AST 的组件来学习代码结构信息。随后，我们使用具有注意力机制的 LSTM 作为解码器并生成代码注释。

本文在广泛研究的两个大规模 Java 方法数据集上进行了全面的实验。实验结果表明，本文方法在 BLEU 和 METEOR 指标方面达到了最佳，并且本文方法的执行时间成本显著低于两种对照方法。

在未来的研究中，我们将会使用程序分析工具来收集更丰富的信息，并结合其他基于深度学习的方法来进一步提高本文提出的代码注释生成方法的有效性。

## 参 考 文 献


[1] XIA X，BAO L，LO D，et al. Measuring program comprehension：A large-scale field study with professionals[J]. IEEE Transactions on Software Engineering，2017，44(10)：951-976.

[2] HU X，LI G，XIA X，et al. Deep code comment generation[C]//2018 IEEE/ACM 26th International Conference on Program Comprehension(ICPC). IEEE，2018：200-210.

[3] CHEN X，YANG G，CUI Z Q，et al. State-of-the-Art survey of Automatic Code Comment Generation[J]. Journal of Software，2021，32(7)：2118-2141.

[4] SONG X，SUN H，WANG X，et al. A survey of automatic generation of source code comments：Algorithms and techniques[J]. IEEE Access，2019，7：111411-111428.

[5] ZHU Y，PAN M. Automatic Code Summarization：A Systematic Literature Review[J]. arXiv：1909.04352，2019.

[6] RODEGHERO P，LIU C，MCBURNEY P W，et al. An eye-tracking study of java programmers and application to source code summarization[J]. IEEE Transactions on Software Engineering，2015，41(11)：1038-1054.

[7] MORENO L，APONTE J，SRIDHARA G，et al. Automatic generation of natural language summaries for java classes[C]//2013 21st International Conference on Program Comprehension(ICPC). IEEE，2013：23-32.

[8] LECLAIR A，JIANG S，MCMILLAN C. A neural model for generating natural language summaries of program subroutines[C]//2019 IEEE/ACM 41st International Conference on Software Engineering(ICSE). IEEE，2019：795-806.

[9] SUTSKEVER I，VINYALS O，LE Q V. Sequence to sequence learning with neural networks[C]// Advances in Neural Information Processing Systems. 2014：3104-3112.

[10] SUN Z，ZHU Q，MOU L，et al. A grammar-based structural cnn decoder for code generation[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2019，33：7055-7062.

[11] LECLAIR A，HAQUE S，WU L，et al. Improved code summarization via a graph neural network[J]. arXiv：2004.02843，2020.

[12] SHIDO Y，KOBAYASHI Y，YAMAMOTO A，et al. Automatic source code summarization with extended tree-lstm[C]// 2019 International Joint Conference on Neural Networks(IJCNN). IEEE，2019：1-8.

[13] CHEN Q，HU H，LIU Z. Code Summarization with Abstract Syntax Tree[C]//International Conference on Neural Information Processing. Cham：Springer，2019：652-660.

[14] LECHNER M，HASANI R. Learning Long-Term Dependencies in Irregularly-Sampled Time Series [J]. arXiv：2006.04418，2020.

[15] HU X，LI G，XIA X，et al. Summarizing source code with trans-

ferred api knowledge[C]//2018 27th International Joint Conference on Artificial Intelligence. 2018:1-9.

[16] HU X, LI G, XIA X, et al. Deep code comment generation with hybrid lexical and syntactical information[J]. Empirical Software Engineering, 2020, 25(3):2179-2217.

[17] PAPINENI K, ROUKOS S, WARD T, et al. BLEU: a method for automatic evaluation of machine translation[C]//Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. 2002:311-318.

[18] BANERJEE S, LAVIE A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments[C]//Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation AND/OR Summarization. 2005:65-72.

[19] XIA Q, YEH C H, CHEN X Y. A Deep Bidirectional Highway Long Short-Term Memory Network Approach to Chinese Semantic Role Labeling[C]//2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019:1-6.

[20] CHUNG J, GULCEHRE C, CHO K H, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling[J]. arXiv:1412.3555, 2014.

[21] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015:1-9.

[22] REN S, HE K, GIRSHICK R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[J]. Advances in Neural Information Processing Systems, 2015, 28:91-99.

[23] GUO B, ZHANG C, LIU J, et al. Improving text classification with weighted word embeddings via a multi-channel TextCNN model[J]. Neurocomputing, 2019, 363:366-374.

[24] SHEN Y, HE X, GAO J, et al. Learning semantic representations using convolutional neural networks for web search[C]//Proceedings of the 23rd International Conference on World Wide Web. 2014:373-374.

[25] GU X, ZHANG H, ZHANG D, et al. Deep API learning[C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016:631-642.

[26] BAHDANAU D, CHO K, BENGIO Y. Neural machine translation by jointly learning to align and translate[J]. arXiv:1409.0473, 2014.

[27] LI Y, WANG Q, XIAO T, et al. Neural Machine Translation with Joint Representation[C]//AAAI. 2020:8285-8292.

[28] WEI B, LI G, XIA X, et al. Code generation as a dual task of code summarization[C]//Advances in Neural Information Processing Systems. 2019:6563-6573.

[29] DENKOWSKI M, LAVIE A. Meteor universal: Language specific translation evaluation for any target language[C]//Proceedings of the Ninth workshop on Statistical Machine Translation. 2014:376-380.

**PENG Bin**, born in 1994, postgraduate. His main research interests include code comment generation and artificial intelligence.

**WU Yong-hao**, born in 1995, Ph.D, candidate. His main research interests include software testing and fault localization.