改进的神经语言模型及其在代码提示中的应用

张 献 贲可荣

(海军工程大学电子工程学院 武汉 430033)

摘 要 语言模型旨在刻画文本段的发生概率,作为自然语言处理领域中的一类重要模型,近年来其被广泛应用于不同软件分析任务,例如代码提示。为提高模型对代码特征的学习能力,文中提出了一种改进的循环神经网络语言模型——CodeNLM。该模型通过分析词向量形式表示的源代码序列,能够捕获代码规律,实现对序列联合概率分布的估计。考虑到现有模型仅学习代码数据,信息的利用不充分,提出了附加信息引导策略,通过非代码信息的辅助来提高代码规律的刻画能力。针对语言建模任务的特点,提出了节点逐层递增策略,通过优化网络结构来改善信息传递的有效性。实验中,针对 9 个 Java 项目共 203 万行代码,CodeNLM 得到的困惑度指标明显优于 *n*-gram 类模型和传统神经语言模型,在代码提示应用中得到的平均准确度(MRR 指标)较对比方法提高了 3.4%~24.4%。实验结果表明,CodeNLM 能有效地实现程序语言建模和代码提示任务,并具有较强的长距离信息学习能力。

关键词 软件分析,代码提示,自然语言处理,语言模型,循环神经网络

中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/jsjkx.191100504C

Modified Neural Language Model and Its Application in Code Suggestion

ZHANG Xian BEN Ke-rong

(School of Electronic Engineering, Naval University of Engineering, Wuhan 430033, China)

Abstract Language models are designed to characterize the occurrence probabilities of text segments. As a class of important model in the field of natural language processing, it has been widely used in different software analysis tasks in recent years. To enhance the learning ability for code features, this paper proposed a modified recurrent neural network language model, called CodeNLM. By analyzing the source code sequences represented in embedding form, the model can capture rules in codes and realize the estimation of the joint probability distribution of the sequences. Considering that the existing models only learn the code data and the information is not fully utilized, this paper proposed an additional information guidance strategy, which can improve the ability of characterizing the code rules through the assistance of non-code information. Aiming at the characteristics of language modeling task, a layer-by-layer incremental nodes setting strategy is proposed, which can optimize the network structure and improve the effectiveness of information transmission. In the verification experiments, for 9 Java projects with 2.03M lines of code, the perplexity index of CodeNLM is obviously better than the contrast *n*-gram class models and neural language models. In the code suggestion task, the average accuracy (MRR index) of the proposed model is 3.4%~24.4% higher than the contrast methods. The experimental results show that except possessing a strong long-distance information learning capability, CodeNLM can effectively model programming language and perform code suggestion well.

Keywords Software analysis, Code suggestion, Natural language processing, Language model, Recurrent neural network

1 引言

近年来,随着软件的更新演化和开源思想的深入,软件仓库(如 GitHub, Stack Overflow 等)累积了海量的代码和其他元数据。这些日益扩增的软件制品促使着新的、数据驱动的方法来分析软件。目前已有一批重要成果引入自然语言处理(Natural Language Processing, NLP)、机器学习等领域的方法来解决不同软件的分析任务[1-2]。可以看出,这些技术的应用正成为近年来软件分析的热点内容。

语言模型(Language Model)是 NLP 领域中的一类重要模型,其通过语料库学习对文本段的发生概率进行估计,进而刻画出语言规律。目前,该模型已在自动分词、句法分析和机器翻译等领域得到了广泛应用[3]。近年来,不少研究者尝试采用语言模型来分析程序语言、软件代码,相关的应用包括代码提示(Code Suggestion)[4-6]、代码补全(Code Completion)[7]、API 推荐(Application Programming Interface Re-

到稿日期:2018-10-16 返修日期:2018-12-20 本文受国家安全重大基础研究计划项目(613315)资助。

张 献(1990-),男,博士生,CCF 学生会员,主要研究方向为软件缺陷挖掘、机器学习,E-mail:tomtomzx@foxmail.com;贲可荣(1963-),男,博士,教授,CCF 高级会员,主要研究方向为软件质量保证、人工智能,E-mail:benkerong08@163.com(通信作者)。

commendation)[8]、编程规范(Coding Convention)提示[9]、缺陷检测[10]、程序合成(Program Synthesis)[11]、伪代码自动生成[12]等。这些成果主要采用经典的 n-gram 语言模型或其改进形式。虽然这类模型的构建简单、直接,但由其自身特点引起的维数灾难问题十分显著,致使模型无法有效地学习长距离上下文信息。

近年来,迅速兴起的深度学习方法为解决上述问题带来了希望。由神经网络构建的语言模型在利用词间语义相似性和学习长距离序列特征方面具有天然优势,可以有效弥补传统语言模型的固有缺陷,目前已在 NLP 领域取得了显著进展[13]。为此,近几年人们也开始重点关注此类方法在程序语言建模方面的应用与拓展[2]。例如,White 等[14]面向海量代码语料库建立了循环神经网络(Recurrent Neural Network,RNN)结构的程序语言模型;Dam 等[15-16]提出了一种基于长短期记忆(Long Short-term Memory,LSTM)单元的 RNN 模型——DeepSoft;Allamanis 等[17]为了推荐合适的方法/类名称,构建了一种基于神经网络的语言模型;尹春林等[18]基于 RNN 语言模型提出了一种源代码主题建模方法,并将其应用于特征定位;Zhang 等[19]利用神经语言模型度量得到的代码熵值进行缺陷预测。这些工作都采用了深度神经网络(Deep Neural Network,DNN)架构,取得的效果相比传统模型均有较为明显的提升。但这些方法仍存在一些不足,例如在解决实际任务时,它们都仅以代码数据为分析对象,而与任务密切相关的非代码信息未得到有效利用,致使模型对软件的分析不够充分。

鉴于此,本文利用深度学习方法对程序语言进行建模,依据任务特点提出了一种改进的基于 LSTM 单元的 RNN 语言模型——CodeNLM,并以代码提示任务为例对模型应用进行验证。该模型以词化后的源代码序列为输入,利用 DNN 学习代码的上下文信息,并捕获程序特征。为增强模型刻画语言规律的能力,本文提出了附加信息引导策略,即引入任务相关的非代码信息来辅助网络学习与决策。在代码提示应用中,抽取的附加信息为代码与项目的关系信息。为提高泛化性能,还提出了节点逐层递增策略,通过限制神经网络规模优化了结构设置。最后,本文以实际软件项目为分析对象,通过对比实验验证了 CodeNLM 在程序语言建模、代码提示及长距离信息学习等方面的有效性。

本文第 2 节介绍了预备知识及相关成果;第 3 节详细阐述了本文提出的改进语言模型 CodeNLM;第 4 节在实际数据集上进行了实验,并对结果进行了分析,以评价本文方法;最后总结全文并展望下一步工作。

2　预备知识与相关工作

2.1　语言模型

语言模型旨在刻画语言规律,其通过语料库学习对序列的联合概率分布进行估计,是处理语言任务的一类基础模型[3]。形式化地,给定一个长度为 l 的语言序列 $S=w_1w_2\cdots w_l$,其中 $w_i\in\Omega$,$S\in\Omega^*$,Ω 为最小语言单元集合,Ω^* 为所有语言序列集合,本文中的最小语言单元为代码的词(token)。则序列 S 的发生概率可用其联合概率表示:

$$P(S)=P(w_1)P(w_2\mid w_1)\cdots P(w_l\mid w_1,w_2,\cdots,w_{l-1})=\prod_{i=1}^{l}P(w_i\mid w_1,\cdots,w_{i-1}) \tag{1}$$

给定一个语料库 $C\subseteq\Omega^*$,语言模型以极大似然估计方法估算参数化的序列概率 $P(S,\boldsymbol{\theta})$,其中 $S\in C$,$\boldsymbol{\theta}$ 为待估计的模型参数。估计 $P(S,\boldsymbol{\theta})$ 的方法有多种,目前最为常用的是 n-gram 模型和神经语言模型(Neural Language Model)。

(1)n-gram 语言模型

n-gram 语言模型运用统计方法估计序列的联合概率分布,是语言模型中的一个经典工作。为简化计算,该模型给定了一个马尔科夫假设:当前词的出现概率仅仅与前面 $n-1$ 个词相关,因此式(1)被近似表示为:

$$P(S)=\prod_{i=1}^{l}P(w_i\mid w_1,\cdots,w_{i-1})\approx\prod_{i=1}^{l}P(w_i\mid w_{i-(n-1)},\cdots,w_{i-1}) \tag{2}$$

对于式(2),常采用频数计数法来估算其中的条件概率:

$$P(w_i\mid w_{i-(n-1)},\cdots,w_{i-1})=\frac{count(w_{i-(n-1)},\cdots,w_{i-1},w_i)}{count(w_{i-(n-1)},\cdots,w_{i-1})} \tag{3}$$

虽然 n-gram 语言模型的概率计算过程较为简单,但其难以学习长距离信息。这主要是因为模型将语言单元视为孤立的符号,无法有效地利用词间的语义关系,导致了维数灾难问题。例如,考虑一个大小为 2 000 的词汇表,令序列最大长度为 10,那么不同的语言序列就约有 10^{33} 种可能,因此 10-gram 语言模型在式(3)的频数统计中将造成数据稀疏和内存爆炸。为此,实际应用中常选择 5 阶以下模型。

(2)神经语言模型

神经语言模型借助神经网络估计序列的联合概率分布,通过利用词间语义相似性来解决传统模型带来的维数灾难问题。该模型由 Bengio 等[20]于 2001 年正式提出,目前已在机器翻译、信息检索、对话系统等诸多 NLP 任务中起到了关键作用。神经语言模型一方面利用词嵌入(Word Embeddings)技术[21]将离散的语言符号映射为连续的一组矢量,语义相似的词将拥有相似的矢量,增强了模型对词间语义关系的学习能力;另一方面借助高度非线性的神经网络拟合序列的发生概率,可以更灵活地刻画语言规律。此外,RNN 等深层网络的兴起,还使得模型在长程依赖(Long-term Dependency)学习方面具有显著优势,其中代表性的工作包括 Mikolov 等首次提出的 RNN 语言模型[22]及其在词向量(Word Vector)生成方面的应用[23]、Zaremba 等提出的正则化 LSTM 网络语言模型[24]等(详情请参见文献[25])。

近年来,已有一批学者尝试采用语言模型来处理不同软件分析任务[2]。Hindle 等[4]运用 n-gram 语言模型首次对代码的自然性(naturalness)进行了分析,发现程序具有显著的重复性和可预测性。基于此,Ray 等[10]分析了缺陷代码的自然性,并成功将 n-gram 模型应用于缺陷检测。为增强局域预测能力,Tu 等[5]提出了一种附加“cache”组件的 n-gram 模型。Nguyen 等[6]通过综合多种代码信息构建了一种统计语义语言模型——SLAMC,并将其应用于代码提示任务。为增强传统 n-gram 模型对长序列规律的刻画,Yang 等[7]提出了一种基于中间表示的改进模型,较好地实现了代码的补全问

题，即能一次性推荐较长的代码段。Bielik 等[11]定义了一种领域特定语言，以此构建了字符级的语言模型并完成了程序合成任务。为规范代码风格，Allamanis 等[9]利用 n-gram 模型实现了编程规范提示，Oda 等[12]基于 n-gram 模型和统计机器翻译框架学习伪代码自动生成任务。

为提高代码特征学习的能力并弥补 n-gram 类模型的固有缺陷，近年来人们也开始尝试运用神经语言模型。White 等[14]利用 RNN 语言模型对大规模语料库进行了挖掘，通过多种实验验证了深度学习模型在学习软件代码规律上优于 n-gram 类模型。为提高模型对长序列依赖关系的学习能力，Dam 等[15-16]使用了 LSTM 单元构建 RNN，并提出了一种程序语言模型——DeepSoft，在 Java 语料库上该模型的性能优于原始 RNN 模型。Allamanis 等[17]提出了两种神经语言模型，然后结合词嵌入表示方法，较好地完成了方法命名任务（Method Naming Task），即为方法或类推荐功能性描述的名称，以提高代码的可读性和维护性。为增强对代码语境的学习，尹春林等[18]基于 RNN 语言模型提出了一种源代码主题建模方法，该方法利用神经网络学习代码及描述文本的语义向量，通过计算向量间的相似性进行了特征定位，并取得了较优的查准率。张献等[19]面向软件缺陷预测提出了一个框架——DefectLearner，其首先利用 RNN 语言模型对代码模块进行熵值度量，然后基于度量结果改善了预测性能。

2.2 代码提示

代码提示作为一项重要技术常被集成于软件开发环境，其通过即时推荐候选代码来辅助编程任务。准确的提示结果不仅可以提高开发效率，还能保证软件质量，因此探索更有效的代码提示方法一直是软件分析领域的一个重要研究内容。图 1 展示了 2 个代码提示示例（示例由本文方法在 Log4J 项目中得到，详情请见 4.1 节）。

```
public class JDBCTest
{……
  try {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  }
  catch (Exception e)
  {
    e.printStackTrace();
    System.out.println(e.?

  ……

  try {
    Category log = Category.getInstance("main");
    log.debug("Debug 1");
    Thread.sleep(500);
    log.info("info 1");
    Thread.sleep(500);
    log.warn("warn 1");
    Thread.sleep(?
```

提示候选项：
1. toString ✔
2. getMessage
3. getClass
4. getName
5. printStackTrace

提示候选项：
1. 500 ✔
2. 200
3. 100
4. 10
5. 50

图 1 代码提示示例（由 CodeNLM 得到）

Fig. 1 Example of code suggestion (acquired by CodeNLM)

代码提示即为当前语句推荐后续代码，其可视为文本序列预测问题。回顾式(1)，语言模型在计算序列联合概率的同时，还需要估计条件概率 $P(w_i \mid w_1, \cdots, w_{i-1})$，这些条件概率恰好可用于文本序列预测。因此，自然地，可利用语言模型进行代码提示工作。Hindle 等[4]最早进行了尝试，将 n-gram 模型应用于 Java 代码提示，实验结果表明该方法可增强 Eclipse 开发环境的提示效果。基于此，Tu 等[5]提出了一种 cache 语言模型，以增强局域预测能力，并取得了优于基本 n-gram 模型的代码提示性能。Nguyen 等[6]提出了一种语言模型 SLAMC，该模型通过综合词的属性、数据类型、从属范围等信息，提高了程序语义理解能力，并在代码提示任务中取得了较优的准确性。Bielik 等[26]提出了一种新的概率生成模型 PGOH（Probabilistic Higher Order Grammar），其较 n-gram 模型能更好地综合上下文信息，在对抽象语法树节点的代码提示任务中取得了较优结果。在文献[22]工作的基础上，White 等[14]构建了几种 RNN 结构的语言模型，通过实验对比，在代码提示任务上，这些方法较 n-gram 类模型有显著优势。本文针对任务特点构建了一个基于 LSTM 单元的改进 RNN 语言模型——CodeNLM，该模型可以有效综合历史信息、捕获代码规律，进而为当前代码段提供符合语法、贴近语义的候选项，辅助完成编程工作。下文将对本文方法进行详细阐述。

3 改进的神经语言模型

面向软件分析任务，本文提出了一种附加信息引导的神经语言模型——CodeNLM，其整体架构如图 2 所示。该模型首先依据实际任务收集目标软件项目，创建语料库；提取出任务相关的样本附加信息，即一些非代码信息，如项目名称、软件类型、软件重要性等级等（依据任务的不同，提取不同的附加信息）；之后针对源代码文件进行必要的数据预处理工作；然后将代码序列及其附加信息一并输入给模型的核心部分，即一个基于 LSTM 单元的 DNN，用于捕获代码的上下文信息、学习程序的语言特征；最终，该模型通过一个 softmax 多分类器来预测输入序列的下一词，以得到的概率分布，从而刻画整段代码的发生规律。

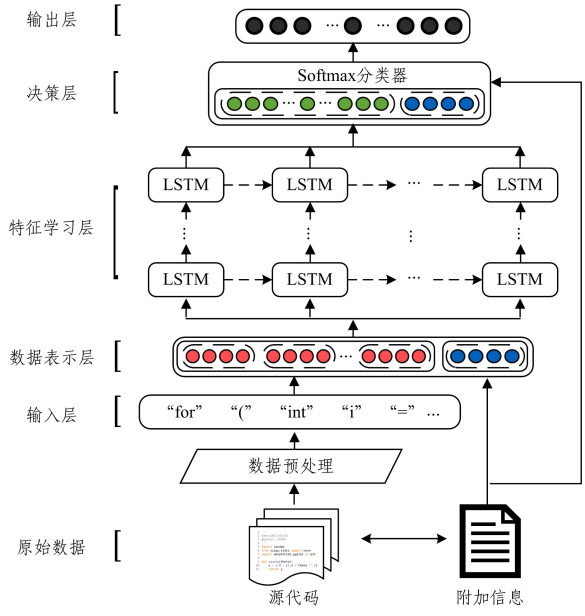

图 2 CodeNLM 框架

Fig. 2 Framework of CodeNLM

图 2 中，CodeNLM 的 DNN 部分主要包括输入层、数据表示层、特征学习层、决策层和输出层。其中，数据表示层利用分布式表示方法[21]将代码的每个词映射为一个稠密的实值矢量，语境相似的词将被赋予相似的矢量；特征学习层则针

对这些矢量序列学习代码语义特征和词序规律;最后决策层通过综合网络信息来预测序列的下一词,从而刻画出代码序列的联合概率分布。为增强模型对软件任务的理解,本文将与任务相关的样本附加信息融入网络。具体地,首先将附加信息量化为实值向量,然后分别与词向量输入和 softmax 层输入相融合,以辅助网络的学习与决策。

下文将对 CodeNLM 的主要构成进行详细介绍。

3.1　数据预处理

在训练神经网络之前,需要对语料库数据进行必要的预处理,具体方法如下:

步骤 1　提取源代码文件。从语料库中提取源代码文件作为模型训练和验证的原始数据。

步骤 2　生成有效代码。去除源代码文件中的注释和空行,保留有效代码行。

步骤 3　代码词化(tokenization)。按照语法规则将源代码进行词化处理,即生成词序列,例如"'for''('int''i''=''0'';'…"。

步骤 4　生成词汇表。合并源代码文件,统计独立的词及其词频,创建词汇表。

步骤 5　替换少见词。将模型处理的词汇总数设为 V,即保留词频数前 V 的词,其余用〈RARE〉标志替换。

步骤 6　生成实验数据集。生成固定长度的代码词序列,创建十折交叉验证数据集。

3.2　网络结构及改进策略

CodeNLM 的核心结构是一个附加信息引导的多层 RNN。在样本的附加信息的辅助下,网络利用 LSTM 单元对长距离信息的建模优势来学习程序的语言特征来捕捉代码序列的语义信息。

LSTM 单元由 Schmidhuber 等[27]提出,其通过使用内存门控机制防止梯度消失,解决了早期 RNN 难以记忆长程依赖的问题。LSTM 单元的数学模型为:

$$\boldsymbol{i}_t=\sigma(\boldsymbol{W}_{xi}\boldsymbol{x}_t+\boldsymbol{W}_{hi}\boldsymbol{h}_{t-1}+\boldsymbol{b}_i)$$
$$\boldsymbol{f}_t=\sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t+\boldsymbol{W}_{hf}\boldsymbol{h}_{t-1}+\boldsymbol{b}_f)$$
$$\boldsymbol{o}_t=\sigma(\boldsymbol{W}_{xo}\boldsymbol{x}_t+\boldsymbol{W}_{ho}\boldsymbol{h}_{t-1}+\boldsymbol{b}_o)$$
$$\boldsymbol{c}_t=\boldsymbol{f}_t\odot\boldsymbol{c}_{t-1}+\boldsymbol{i}_t\odot\tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t+\boldsymbol{W}_{hc}\boldsymbol{h}_{t-1}+\boldsymbol{b}_c)$$
$$\boldsymbol{h}_t=\boldsymbol{o}_t\odot\tanh(\boldsymbol{c}_t)$$

其中,$\sigma(x)=1/(1+e^{-x})$ 和 $\tanh(x)=(e^{x}-e^{-x})/(e^{x}+e^{-x})$ 为激活函数;$\boldsymbol{W}$ 和 $\boldsymbol{b}$ 分别表示权重矩阵和偏置向量;$\odot$ 表示按元素相乘,即 Hadamard 积;$\boldsymbol{i}_t$,$\boldsymbol{f}_t$,$\boldsymbol{o}_t$ 分别表示输入门、遗忘门和输出门操作;$\boldsymbol{x}_t$,$\boldsymbol{c}_t$,$\boldsymbol{h}_t$ 分别表示网络输入、记忆单元状态和隐含层状态;t 表示展开时间。

考虑到现有的语言模型在数据学习和网络结构设计方面仍具有一定的盲目性,本文提出了两种改进策略:附加信息引导策略(策略 1)和节点逐层递增策略(策略 2),以增强模型对软件任务的理解能力和改善程序语言的建模能力。

(1)附加信息引导策略

现有语言模型大都从源代码数据中提取知识,而有助于解决软件任务的部分样本信息(如与项目的对应关系、软件类型、软件重要性等级)难以在代码中体现,导致模型未能对软件进行充分分析。为解决这一问题,本文提出将任务相关的非代码信息融入网络,引导模型更准确地刻画语言特征,提高原有网络对数据学习的针对性。需要指出,应依据具体应用任务,抽取不同的附加信息。下文将以代码提示任务为例进行阐述。

具体地,该策略首先将抽取的附加信息以实值向量形式表示,然后分别同词向量输入和 softmax 层输入相融合,用于辅助网络学习与决策。其形式化描述为:

$$\begin{aligned}&\boldsymbol{x}_t^1=\boldsymbol{w}_t \longrightarrow \tilde{\boldsymbol{x}}_t^1=[\boldsymbol{w}_t\quad \boldsymbol{p}_t]\\&\boldsymbol{x}_t^{\text{softmax}}=\boldsymbol{h}_t^{N_{\text{layer}}} \longrightarrow \tilde{\boldsymbol{x}}_t^{\text{softmax}}=[\boldsymbol{h}_t^{N_{layer}}\quad \boldsymbol{p}_t]\end{aligned}\tag{4}$$

其中,$\boldsymbol{x}$ 表示原网络输入,$\tilde{\boldsymbol{x}}$表示改进后的网络输入,$\boldsymbol{w}$ 表示词向量,$\boldsymbol{p}$ 表示向量化的样本附加信息,$\boldsymbol{h}$ 表示 LSTM 隐含状态,N_{layer}表示隐含层层数,t 表示展开时间。

对于不同的应用任务可以为模型引入不同的附加信息。本文以代码提示应用为例进行分析。代码提示旨在为编程人员提供准确的代码候选项,以提高开发效率。由于不同的软件项目拥有不同的应用背景及开发过程,且其反映在代码产品上,存在统计差异。为使模型更精准地捕获代码规律,并更准确地完成代码提示,本文将样本与项目的关系信息融入网络,如图 3 所示。针对代码提示应用,式(4)中的附加信息 $\boldsymbol{p}$ 即为量化后的项目名称(ID),其量化使用词嵌入技术。具体实现中,通过路径识别出代码文件对应的项目 ID,然后将代码 token 的 ID 序列及其对应的项目 ID 序列输入到神经网络,这一过程是自动化的。

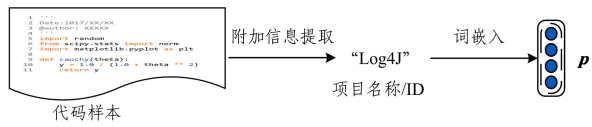

图 3　样本附加信息的提取与量化

Fig. 3　Extraction and quantification of additional information

(2)节点逐层递增策略

目前在神经网络结构设计方面缺乏经验性总结,导致实验盲目、低效。本文通过分析神经语言模型特点,提出了一种简单有效的网络规模正则化方法,即节点逐层递增策略,由输入层到特征学习层再到输出层的网络节点数按递增模式设置:

$$dim_{\text{in}}<dim_h^1<dim_h^2<\cdots<dim_h^{N_{\text{layer}}}<dim_{\text{out}}$$

其中,dim_{in}表示输入层节点数,即词嵌入维度 dim_w;dim_h^l 表示隐含层节点数,N_{layer}表示隐含层层数;dim_{out}表示输出层节点数,即 softmax 分类类别数。

这一经验策略有悖于常见的 DNN 结构设计方法,即整体上网络节点逐渐递减或节点先增后减的设计模式,例如经典的 LeNet5[28],AlexNet[29] 和 GoogLeNet[30] 等 DNN。存在这样差异的原因主要是面向任务的不同。上述这些经典模型主要面向计算机视觉任务,例如在图像识别中,考虑处理像素为 256×256 的图像,那么模型的输入维度为 65 536;输出层为分类类别,其节点一般在 10～5000 之间,远小于输入维度。因此,网络通常需要逐层压缩数据,以实现特征的抽象。而本文面向的是语言建模任务,模型的输入通常为词向量,维度一般在 100～1000 之间;输出层对应词汇表中的词,维度一般在 2000～20000 之间,远大于输入维度。这正是语言建模任务

的一个重要特点。针对这一特点,本文使用各层节点逐层递增的模式设计网络,此方法不仅有助于信息的逐层传递,还能有效限制网络规模、降低过拟合风险,从而提高模型泛化性能和训练速度。

3.3 模型训练

深度学习模型在训练过程中常会出现过拟合、梯度爆炸或梯度弥散等问题。为此,本文采用以下几种方法进行优化网络训练:

1)Dropout 方法[24]。在模型训练中,对于网络节点,该方案按照一定概率将其暂时丢弃,以阻止神经元的共适应问题,是一种有效的抗过拟合方法。

2)梯度剪裁(Gradient Clipping)[31]。通过限制梯度范数来防止梯度爆炸,此方法简单有效。

3)自适应学习率。通过动态学习率优化模型训练、提高泛化性能。本文令梯度下降算法的学习率 l_{rate} 随训练轮数动态调整:

$$l_{\text{rate}}=\begin{cases}1, & i<\lambda\\ \eta^{i-\lambda}, & \lambda\leqslant i\leqslant N_{\text{epoch}}\end{cases}$$

其中,$i,N_{\text{epoch}}\in\mathbf{Z}^{+}$ 分别表示训练轮数和最大训练轮数,$\lambda\in[1,N_{\text{epoch}}]$和 $\eta\in(0,1)$是调节学习率曲线的参数。

3.4 模型评估

语言模型的评价常采用交叉熵(cross-entropy)或困惑度(perplexity)作为度量指标[3]。交叉熵用于衡量估计模型结果与真实概率分布之间的差异情况,其值越小,模型的表现就越好。对于给定的语料库 $C=\{S_1S_2\cdots S_N\}(C\subseteq\Omega^{*})$和语言模型 M,该模型在该语料库上的交叉熵值可由下式计算:

$$\begin{aligned}CE_M(C)&=-\frac{1}{l_C}\log_2 P_M(C)\\&=-\frac{1}{l_C}\sum_{j=1}^{N}\log_2 P_M(S_j)\\&=-\frac{1}{l_C}\sum_{j=1}^{N}\sum_{i=1}^{l_j}\log_2 P_M(w_{j,i}\mid w_{j,1},\cdots,w_{j,i-1})\end{aligned}$$

其中,$S_j\in C$ 表示语料库中的序列,N 表示序列总数,l_j 表示序列 S_j 的长度,$l_C=\sum_{j=1}^{N}l_j$ 表示语料库序列的总长度,$P_M(S_j)$表示模型 M 求得的序列 S_j 的发生概率,$P_M(C)=\prod_{j=1}^{N}P(S_j)$表示语料库的发生概率。

困惑度指标是交叉熵的指数形式,同样地,其值越小,模型的表现就越好。对于给定的语料库 C 和语言模型 M,该模型在该语料库上的困惑度值可由下式得到:

$$PP_M(C)=2^{CE_M(C)}$$

其中,$CE_M\in[0,+\infty)$,$PP_M\in[1,+\infty)$。

4 实验与结果分析

本节针对实际项目数据,通过实验分析和结果对比来验证 CodeNLM 模型的有效性,4.1 节介绍实验数据集,4.2 节说明实验环境及模型参数,4.3 节将从语言模型性能、代码提示准确度和长距离信息学习能力 3 个方面来综合评价本文方法。

4.1 实验数据集

为便于模型的对比,本文选用文献[4-6]和文献[16]均使用的 Java 公开数据集,该数据集的信息如表 1 所列。从表 1 中可以看到,该数据集包含 9 个 GitHub 开源项目,共计 9060 个.java 文件,总代码行数约 203 万,有效代码行约 114 万。

表 1 实验数据集

Table 1 Information of evaluated dataset

项目名	版本	文件数	代码行	有效代码行
Ant	20110123	1193	254457	128266
Batik	20110118	1657	367293	195467
Cassandra	20110122	545	135992	97029
Log4J	20101119	353	68528	34479
Lucene	20100319	2279	429957	266573
Maven2	20101118	386	61622	38593
Maven3	20110122	847	114527	70462
Xalan-J	20091212	973	349837	171619
Xerces	20110111	827	257572	138841
总计	—	9060	2039785	1141329

4.2 实验环境与模型参数

实验运行环境为 Windows7 64 位操作系统、Intel i7-6700K CPU、16GB 内存及 1 块 GeForce GTX 1070 显卡,使用的深度学习工具为 TensorFlow 1.0.0[1)],采用的代码词化工具来自文献[10]。

实验中,CodeNLM 的超参数设置如下:词汇表大小 $V=5000$;词向量维度 $dim_w=500$;项目属性的嵌入维度 $dim_p=5$;LSTM 层数 $N_{\text{layer}}=2$;LSTM 层的隐含节点 dim_h 分别为 800 和 1200;dropout 方法的节点保留概率 $p_{\text{keep}}=0.6$;按时间反向传播展开步长 $N_{\text{step}}=35$;批处理的大小 $N_{\text{batch}}=30$;训练轮数 $N_{\text{epoch}}=15$;允许的最大梯度范数 $grad_{\max}=5$;LSTM 单元遗忘门偏置的初始值为 0,网络其他参数的初始值按[-0.08,0.08]的均匀分布随机设置;模型优化使用梯度下降算法,自适应学习率 l_{rate} 的参数 $\lambda=10$,$\eta=0.5$。不难看出,CodeNLM 的“输入层-隐含层-输出层”(即“词嵌入层-LSTM 层-softmax 层”)的维度依次为 505,800,1205 和 5000,各层节点逐层递增。

4.3 实验结果与对比

本节从语言模型性能、代码提示准确度和长距离信息学习能力 3 个方面来评价本文方法。

4.3.1 语言模型的性能评估与对比

针对实验数据集,采用 10 折交叉验证方法来验证 CodeNLM 的有效性,具体结果如图 4 所示。本文方法在测试集上取得的最优困惑度为 2.650,最劣值为 2.886,均值为 2.802,模型平均耗时约 11102s。

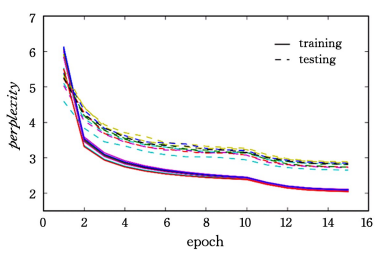

图 4 损失函数曲线(10 折交叉验证)

Fig. 4 Loss curves (10-fold cross validation)

1) https://www.tensorflow.org/

由图 4 可以看出,模型经过 15 轮数据训练,困惑度指标逐渐降低并趋于收敛,其中在第 10 轮时曲线明显变化的原因是学习率的下降。通过学习率的自适应调整,全局搜索和局部搜索实现了平衡,模型得到了充分训练。此外,10 次实验的损失曲线十分接近,测试集与训练集的困惑度相差不大,这说明 CodeNLM 具有良好的稳定性和泛化能力。

表 2 列出了不同语言模型在此数据集上的性能,进一步通过模型对比验证了 CodeNLM 的有效性。文献[4-5]分别采用 n-gram 模型和 cache n-gram 模型进行了代码分析,这两种模型获得的最优交叉熵介于 3～4 之间[5],即困惑度均大于 $8(2^3=8)$。文献[15]面向软件代码提出了一种基于 LSTM 单元的深度语言模型 DeepSoft,在相同的数据集上,其求得的最优困惑度为 4.72[16]。文献[24]提出了一种有效的正则化 LSTM 网络,该网络是语言建模领域的一项经典工作并被多次引用,不同规模的网络在表 1 所列数据集上的最优困惑度为 3.044(具体参数设置请见文献[24])。本文方法得到的指标值为 2.802,明显优于上述多种模型结果,表明了 CodeNLM 在刻画代码复杂特征、学习程序语言规律方面具有优越性。其中,CodeNLM* 表示未使用策略 1(即附加信息引导策略);CodeNLM** 表示未使用策略 1 和策略 2(即节点逐层递增策略),其词向量和隐含层维度均为 800,其他方面较 CodeNLM 不变。这两个版本的模型得到的困惑度指标分别为 2.881 和 2.946,可以看出本文提出的两种策略都起到了明显的正向促进作用,提高了语言模型的学习能力。但从另一方面也可以看到,深度学习模型带来了过高的计算成本,其中本文所提 CodeNLM 的训练时间达到了 185 min,远大于 n-gram 类模型(详情请见表 2,其中 n-gram 类模型的数据来自文献[5]),这种显著的开销对比也可在文献[11]中得到应证。因此,如何构建低功耗的 DNN 仍是当前的一个重要挑战和研究方向[2]。

表 2　不同模型的困惑度指标值

Table 2　Perplexity values of investigated models

模型	n-gram 类模型		DeepSoft[16]	Regularized LSTMs[24]			本文方法		
	n-gram[4]	Cache LM[5]		Small	Medium	Large	CodeNLM**	CodeNLM*	CodeNLM
困惑度	12～14	8～9	4.720	3.739	3.044	3.152	2.946	2.881	2.802
训练时间/ min	0.9	2.9	—	79	457	1750	150	180	185

注:* 表示未使用策略 1,* * 表示未使用策略 1 和策略 2

4.3.2　代码提示任务结果与对比

本文选用两类常用的指标来衡量模型的代码提示性能[4-6]:平均排序倒数(Mean reciprocal Rank,MRR)和 *Top-n* 准确率。指标的计算方法如下:

$$MRR=\frac{1}{|T|}\sum_{i=1}^{|T|}\frac{1}{rank_i}$$

$$Top\text{-}n=\frac{1}{|T|}\sum_{i=1}^{|T|}r_i$$

其中:

$$r_i=\begin{cases}1, & rank_i\leqslant n\\ 0, & rank_i>n\end{cases}$$

其中,T 表示测试的 token 总数,$rank_i$ 表示正确结果 t_i 获得的推荐名次。Top-n 准确率衡量了正确结果出现在前 n 个候选项中的可能性;MRR 指标则表示正确结果获得的平均排名的倒数,例如 *MRR* 值为 0.5 表明平均意义上正确结果出现在第二推荐位置。

表 3 列出了不同语言模型在表 1 数据集上的代码提示结果,以进一步评价 CodeNLM 的有效性。

表 3　不同模型在代码提示任务中的性能对比

Table 3　Performance comparison of investigated models for code suggestion

(单位:%)

	模型	n-gram 类模型			Regularized LSTMs[24]			本文方法		
		n-gram[4]	Cache LM[5]	SLAMC[6]	Small	Medium	Large	CodeNLM**	CodeNLM*	CodeNLM
性能指标	MRR	52.8	67.6	—	68.5	73.8	72.9	75.4	76.2	77.2
	Top-1	46.3	—	65.6	58.9	64.9	64.0	66.9	67.7	68.5
	Top-5	56.8	—	78.2	80.3	84.9	83.8	86.0	86.8	88.0

注:* 表示未使用策略 1,* * 表示未使用策略 1 和策略 2

由表 3 可以看出,神经语言模型整体较传统 n-gram 类模型有显著优越性,3 种准确度指标均有明显提升(为避免二次实验引入偏差,n-gram 类模型结果使用了原文数据,其中文献[6]的结果是目前此语言模型得到的最佳结果)。这是由于 LSTM 模型在数据表示、非线性特征变换、长距离信息学习等方面具有突出优势,其中 n-gram 语言模型可综合的 context 信息长度远小于神经类模型(在 4.3.3 节中将进行重点分析)。在全部 9 个项目上,本文方法均得到了最优性能,其中 MRR 指标的均值为 77.2%,较其他模型(绝对)增长 3.4%～24.4%;Top-1 指标均值为 68.5%,增长 2.9%～22.2%;Top-5 指标均值为 88.0%,增长 3.1%～31.2%。表 3 还列出了 CodeNLM* 模型和 CodeNLM** 模型的结果。以 MRR 指标为例,它们分别得到了 75.4%和 76.2%,同样优于其他方法,但劣于使用了改进策略的 CodeNLM。

通过此次对比实验可以得出,CodeNLM 能够捕获代码的复杂特征,学习程序语言规律,并较好地完成了代码提示工作,较对比模型存在显著优势。同时,实验结果也验证了本文提出的两种改进策略能够提升模型的性能,适用于程序语言建模及相关软件分析任务。

4.3.3　长距离信息学习能力分析

学习长距离信息一直是语言建模任务的一个难题。作为软件的核心构成,源代码中蕴含了复杂的语法及语义信息。

因此，对于语言模型来说，拥有良好的长程依赖学习能力至关重要。本节考察 CodeNLM 在此方面的性能。

图 5 展示由本文方法得到的序列下一词的估计概率分布，即给定长度为 l 的序列 $S=w_1w_2\cdots w_l$，模型得到的 $P(x=w_{l+1}\mid w_1,w_2,\cdots,w_l)$ 分布，其中 x 表示模型的预测结果，w_{l+1} 表示序列 S 的(真实)下一词，$w_1w_2\cdots w_l$ 称作 w_{l+1} 的“历史”。

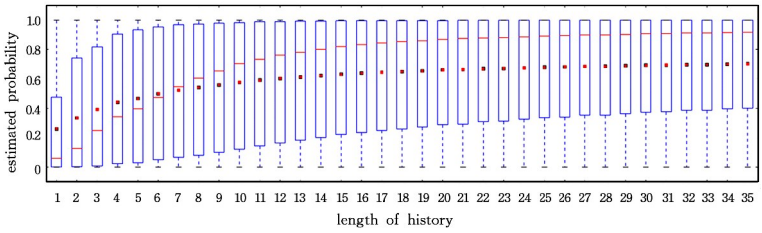

图 5 序列下一词的估计概率

Fig. 5 Estimated probability of next real token of a given sequence

由图 5 可以看出，随着给定历史信息的增加，CodeNLM 对序列下一词 w_{l+1} 的预测愈加准确，得到的估计概率分布呈上升趋势。当输入给模型的序列长度为 1 时，w_{l+1} 获得的估计概率均值仅为 0.268；当历史长度大于 18 后，估计概率均值超过了 0.650 并趋于平缓；当序列长度达到 35 时，模型赋予 w_{l+1} 的概率均值为 0.702。其中，估计概率的中位数变化更为明显，其随着给定历史信息的增长，由 0.060 升高到 0.917。而对于传统的 n-gram 语言模型，其由于自身固有的维数灾难问题，难以处理长文本数据。在文献[4-6]中，此类模型在处理代码提示任务时分析的历史长度不超过 4。因此，从上述分析中不难得出，CodeNLM 具有优越的长距离信息学习能力，能有效综合历史输入，进而捕获语言规律。

结束语 本文面向程序语言提出了一种附加信息引导的 RNN 语言模型——CodeNLM。该模型以源代码的词序列为分析对象，利用多层 LSTM 网络进行特征学习，改善了传统语言模型的固有不足，增强了长距离信息学习能力。在神经网络的构建上，提出了两种改进策略：1)将任务相关的非代码信息融入网络，引导模型更准确地刻画代码特征，改善了原有方法对数据学习的盲目性(以代码提示任务为例，引入的附加信息为代码与项目的关系信息)；2)针对语言模型的特点，令网络节点逐层递增，该策略能够限制模型规模、降低过拟合风险。

选取了含 2.03M 代码行的 Java 数据集用于模型验证。在代码提示任务(token 级)中，CodeNLM 的预测性能优于对比的 n-gram 类模型和神经语言模型，Top-1，Top-5 和 MRR 指标得到明显提升(2.9%～31.2%)。实验结果表明，本文方法能更精准地刻画语言规律和捕获代码特征，较好地实现了程序语言建模和代码提示任务。下一步工作将考虑不同方面附加信息的提取与融合，并将模型应用于缺陷检测等其他软件分析问题。

参 考 文 献

[1] 王千祥，张健，谢涛，等. 软件分析：技术、应用与趋势/ CCF 2015-2016 中国计算机科学技术发展报告[M]. 北京：机械工业出版社，2016：55-113.

[2] ZHANG X，BEN K R. Application of deep learning methods in software analysis [J]. Computer Engineering and Science，2017(12)：2260-2268. (in Chinese)
张献，贲可荣. 深度学习方法在软件分析中的应用[J]. 计算机工程与科学，2017，39(12)：2260-2268.

[3] JURAFSKY D，JAMES H M. Speech and language processing (2nd ed) [M]. Upper Saddle River：Pearson/Prentice Hall，2009：4th Chapter.

[4] HINDLE A，BARR E T，SU Z，et al. On the naturalness of software[C]// Proceedings of the 34th International Conference on Software Engineering. Piscataway：IEEE，2012：837-847.

[5] TU Z，SU Z，DEVANBU P. On the localness of software[C]// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York：ACM，2014：269-280.

[6] NGUYEN T T，NGUYEN A T，NGUYEN H A，et al. A statistical semantic language model for source code[C]// Proceedings of the 9th Joint Meeting on Foundations of Software Engineering. New York：ACM，2013：532-542.

[7] YANG Y，JIANG Y，GU M，et al. A language model for statements of software code[C]// Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. Piscataway：IEEE，2017：682-687.

[8] NGUYEN A T，HILTON M，CODOBAN M，et al. API code recommendation using statistical learning from fine-grained changes[C]// Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York：ACM，2016：511-522.

[9] ALLAMANIS M，BARR E T，BIRD C，et al. Learning natural coding conventions[C]// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York：ACM，2014：281-293.

[10] RAY B，HELLENDOORN V，GODHANE S，et al. On the naturalness of buggy code[C]// Proceedings of the 38th International Conference on Software Engineering. New York：ACM，2016：428-439.

[11] BIELIK P，RAYCHEV V，VECHEV M. Program synthesis for character level language modeling[C]// Proceedings of the 5th International Conference on Learning Representations. Toulon，OpenReview，2017.

[12] ODA Y，FUDABA H，NEUBIG G，et al. Learning to generate pseudo-code from source code using statistical machine transla-

tion[C]// Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. Piscataway: IEEE,2015:574-584.

[13] HIRSCHBERG J,MANNING C D. Advances in natural language processing[J]. Science,2015,349(6245):261-266.

[14] WHITE M,VENDOME C,LINARES-VÁSQUEZ M,et al. Toward deep learning software repositories[C]// Proceedings of the 12th IEEE/ACM Working Conference on Mining Software Repositories Piscataway. IEEE,2015:334-345.

[15] DAM H K,TRAN T,GRUNDY J,et al. DeepSoft:a vision for a deep model of software[C]// Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York:ACM,2016:944-947.

[16] DAM H K,TRAN T,PHAM T. A deep language model for software code[C]// Workshop on Naturalness of Software,Collocated with the 24th International Symposium on Foundations of Software Engineering. New York:ACM,2016.

[17] ALLAMANIS M,BARR E T,BIRD C,et al. Suggesting accurate method and class names[C]// Proceedings of the 10th Joint Meeting on Foundations of Software Engineering. New York: ACM,2015:38-49.

[18] YIN C L,WANG W,LI T,et al. Using RNNLM to conduct topic oriented feature location method[J]. Journal of Frontiers of Computer Science and Technology,2017,11(10):1599-1608. (in Chinese)
尹春林,王炜,李彤,等. 利用 RNNLM 面向主题的特征定位方法[J]. 计算机科学与探索,2017,11(10):1599-1608.

[19] ZHANG X,BEN K R,ZENG J. Cross-entropy:a new metric for software defect prediction[C]// Proceedings of the 18th IEEE International Conference on Software Quality, Reliability and Security. Piscataway:IEEE,2018:111-122.

[20] BENGIO Y,DUCHARME R,VINCENT P. A neural probabilistic language model[C]// Proceedings of the 15th Annual Conference on Neural Information Processing Systems. Massachusetts:MIT Press,2001:932-938.

[21] LECUN Y,BENGIO Y,HINTON G. Deep learning[J]. Nature, 2015,512(7553):436-444.

[22] MIKOLOV T,KARAFIÁT M,BURGET L,et al. Recurrent neural network based language model[C]// Proceedings of the Annual Conference of the International Speech Communication Association. Makuhari,ISCA,2010:1045-1048.

[23] MIKOLOV T,YIH W,ZWEIG G. Linguistic regularities in continuous space word representations[C]// Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics:Human Language Technologies. Atlanta,NAAC,2013:746-751.

[24] ZAREMBA W,SUTSKEVER I,VINYALS O. Recurrent neural network regularization[J]. arXiv:1409.2329,2014.

[25] SALEHINEJAD H,BAARBE J,SANKAR S,et al. Recent advances in recurrent neural networks[J]. arXiv:1801.01078, 2018.

[26] BIELIK P,RAYCHEV V,VECHEV M. PHOG:probabilistic model for code[C]// Proceedings of the 33rd International Conference on Machine Learning. New York: ACM, 2016: 2933-2942.

[27] HOCHREITER S,SCHMIDHUBER J. Long short-term memory[J]. Neural Computation,1997,9(8):1735-1780.

[28] LECUN Y,BOTTOU L,BENGIO Y,et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE,1998,86(11):2278-2324.

[29] KRIZHEVSKY A,SUTSKEVER I,HINTON G E. Imagenet classification with deep convolutional neural networks[C]// Proceedings of the 26th Annual Conference on Neural Information Processing Systems. Massachusetts: MIT Press, 2012: 1097-1105.

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

[31] PASCANU R,MIKOLOV T,BENGIO Y. On the difficulty of training recurrent neural networks[C]// Proceedings of the 30th International Conference on Machine Learning. New York: ACM,2013:1310-1318.