基于深度学习的代码分析研究综述

张峰逸 彭 鑫 陈 驰 赵文耘

(复旦大学软件学院 上海 201203)

(上海市数据科学重点实验室(复旦大学) 上海 201203)

摘 要 随着现代软件规模的不断增大,程序员面临着与日俱增的开发与维护负担,一种辅助他们完成开发流程的工具成为了迫切需求。基于深度学习的代码分析技术从分析代码性质入手,致力于辅助程序员生成或理解代码,成为了软件工程近期研究的热点。总结近期软件工程和人工智能领域中基于深度学习的代码分析研究。论述其一般的技术流程,并从代码表征方法、模型选择以及应用场景三个方面对现有工作进行分类,体现了不同工作的技术特点与设计思路。通过对现有工作的总结整理,认为深度学习在代码分析中的应用还处于初级阶段,既体现了其性能的优越性,又存在着诸如抽象问题描述、数据标注和领域知识理解等问题。在未来的研究中,可能产生突破性进展的发展方向包括建立可标注数据集、设计合理的评判体系以及与知识图谱等新型人工智能技术相结合等。

关键词 代码分析 深度学习 表征学习

中图分类号 TP3 **文献标识码** A DOI: 10.3969/j.issn.1000-386x.2018.06.002

RESEARCH ON CODE ANALYSIS BASED ON DEEP LEARNING

Zhang Fengyi Peng Xin Chen Chi Zhao Wenyun

(*Software School, Fudan University, Shanghai 201203, China*)

(*Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China*)

Abstract With the continuous increase in the size of modern software, programmers are faced with an ever-increasing burden of development and maintenance. A tool that assists them in completing the development process has become an urgent need. Deep learning based code analysis technology starts with analyzing the nature of code. It is committed to helping programmers generate or understand code, which has become a hot topic in recent software engineering research. This article summarized the recent code analysis research based on deep learning in the field of software engineering and artificial intelligence. It discussed its general technical process, and classified the existing work from three aspects: code representation method, model selection and application scenario, which reflected the technical characteristics and designed ideas of different jobs. By summarizing the existing work, this paper believed that the application of deep learning in the code analysis is still in its infancy, which not only embodies the superiority of its performance, but also has issues such as abstract problem description, data annotation and domain knowledge understanding. In the future research, the development direction that may produce breakthrough progress includes the establishment of a set of data that can be labeled, a rationally designed evaluation system, and a combination of new artificial intelligence technologies such as knowledge maps.

Keywords Code analysis Deep learning Representation learning

0 引 言

计算机软件是现代社会最具代表性的产物,它几乎被应用在人类生活的方方面面,无数优秀的开发者参与贡献了大量高水平的软件代码。然而由于代码本身的抽象性、复杂性与可变性[1],大型软件的开发始终是一个艰巨的任务。随着软件代码规模的日益增长,

收稿日期: 2017-12-23。张峰逸,硕士,主研领域: 智能化软件开发。彭鑫,教授。陈驰,博士。赵文耘,教授。

开发者所面临的压力越来越大。如何通过分析代码本身的规律来帮助开发者理解、编写或维护代码成为了现代软件工程所关注的主要任务。

学界对于代码分析的研究由来已久，早期的代码分析研究以形式化的逻辑演绎方法为主[2-4]。随着开源代码的兴起，大量高质量的开源项目提供了诸如源代码、代码注释、故障报告以及测试用例等各方面的程序信息。这激发了研究者们的新的思路：通过代码的统计学特性来获得人类可以理解的知识。1984 年，Don Knuth 提出了“文字编程”的概念，他认为编程实际上是人类交流的结果[5]。受到该思想的启发，Hindle 等[6]提出了代码的“自然性”假说，他们认为代码具有和自然语言相似的性质，相似的功能的代码实现往往会有不同程度的相似性，因此代码是可以预测的。代码“自然性”理论的提出像是打开了一扇大门，人们意识到运用统计规律来分析代码信息是可行的。大量的机器学习算法被运用到代码中来，代码的性质从各个方面得到了挖掘，这极大地丰富了人类对于代码的认识。但是，现有的机器学习算法在很大程度上依赖于特征工程，这是一项劳动密集型的工作，并且只针对于特定的任务有效[7]。近期的机器学习文献指出，由人类所设计的特征很有可能无法很好地反应数据的真实情况，因为人类观察的准度与广度都是非常有限的[8]。因此如何选取合适的特征成为了应用机器学习算法的一大挑战。

深度学习是机器学习中一种基于对数据进行表征学习的算法[9]，其概念源于人工神经网络的研究，通过组合低层特征形成更加抽象的高层特征表示，从而发现数据的分布特征。与传统的机器学习方法相比，深度学习的最大特点是采用了非监督或者半监督式的特征学习与提取方法，因此观察对象的特征并不需要人为地指定，而是通过大量的训练数据学习归纳而来[10]。这种“黑盒”式的特征提取与归纳方法在分析代码数据上有着巨大的优势：避免了人类观察可能的误判，数据量越大，神经网络模型对于代码特征的挖掘就越接近数据原本的特征分布。这在无形中把代码中人类难以理解的抽象性转移给了海量的训练数据。深度学习用海量的训练数据来代替人对于代码的观察，使算法自己总结出代码的性质与规律，其在诸如图像识别、机器翻译等领域中显示了远超传统机器学习算法的性能，甚至在某些任务中表现出了超越人类的智能[11-12]。在软件工程领域，深度学习同样在诸如代码生成和代码理解等任务上展现了超越传统工作的潜力。

1 研究概述

代码分析指的是通过分析挖掘代码本身性质，帮助开发者产生、维护并理解代码的一类方法。随着深度学习的快速发展，软件工程领域也开始接纳这一新兴的技术，基于深度学习的代码分析更是成为了当前研究的热点。在近期的研究工作中，这种分析方法有很多的用途，其典型的应用场景包括根据已有的代码信息来推断下一步的代码（代码的推荐与生成）、将一种程序语言的语句“翻译”成另一种程序语言（代码迁移）或为代码片段生成摘要信息（代码摘要生成）等。这些工作的成功应用展示了将深度学习技术与代码分析任务相结合的广阔应用前景。

不管是用于什么用途，深度学习模型的训练都具有相似的技术流程。如图 1 所示，在代码分析任务中，神经网络有一个初始的网络参数（多由 0 值或随机值表示），在每次读入输入代码之后，神经网络根据任务目标计算损失函数，并设立阈值来判定是否用反向传播算法（BP 算法）进行参数更新。当损失函数的值小于特定阈值之后，整个网络训练完毕，开始执行特定的代码分析任务。

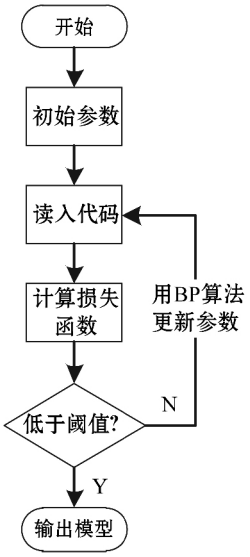

图 1 基于深度学习代码理解的一般技术流程

本文按照上述的技术流程对现有研究进行了总结。如表 1 所示，表中的每一行代表一个具有代表性的研究工作。第一列代表将代码表征成向量形式的方法，包括 one-hot 编码、词袋编码以及词嵌入模型；第二列代表代码表征的粒度，包括字符级别、词条级别以及 AST 数节点级别；第三列表示不同研究工作所选用的神经网络模型，共包括循环 RNN、CNN 以及 Seq2seq 模型三种；最后一列表示不同研究工作的实际应用场景，

包括代码推荐、代码生成、代码搜索、代码迁移、代码摘要以及故障定位六种。本文通过以上的三个标准对现有工作进行了分类与总结,归纳出了现有研究工作的优势与局限,并提出了对该方法未来发展趋势的若干建议。

表 1　现有研究工作小结

研究工作	编码方法	模型构造	粒度	应用
文献[13]	one-hot	RNN	字符	代码生成
文献[14]	one-hot	RNN	字符	代码生成
文献[16]	rVSM	无	词条	故障定位
文献[26]	词嵌入	无	词条	代码搜索
文献[24]	词嵌入	无	词条	代码同义词
文献[23]	词嵌入	无	词条	代码迁移
文献[7]	词嵌入	CNN	AST 节点	代码生成
文献[22]	词嵌入	无	词条	故障定位
文献[17]	VSM	CNN	词条	代码摘要
文献[32]	词嵌入	RNN	词条	代码推荐
文献[33]	词嵌入	RNN	词条	故障定位
文献[34]	LDA	Seq2seq	词条	代码生成
文献[27]	VSM	RNN	词条	代码搜索
文献[36]	VSM	CNN	文件	故障定位
文献[37]	VSM	RNN	词条	代码推荐
文献[38]	VSM	RNN	词条	代码摘要
文献[45]	one-hot	RNN	词条	代码生成
文献[35]	词嵌入	Seq2seq	AST 节点	代码迁移

2　代码的表示

深度学习作为一种表征学习方法,需要接受向量形式的输入。因此要讨论深度学习对代码的应用,首先应了解代码是如何被编码为向量形式的。目前比较流行的向量表示方法分为以下三种。

2.1　one-hot 编码

one-hot 编码又称一位有效编码,是数据表征中最简单直接的方法。它的主要思路是采用 N 位的状态寄存向量对 N 个状态进行编码,每个状态都对应一个独立的寄存位,并且在任意情况下都只有一位有效。one-hot 编码用最小的技术成本保证了编码之间的相互正交,使得每一个被表示的单位都有一个唯一的向量表示。这种方法的弊端是编码后的向量不表示任何具体含义,仅仅是一个独立的状态标记,而且随着状态维度的增大,其空间分布会变得极为稀疏,因此该方法比较适合低维编码任务。在代码分析任务中,这种编码多被运用于字符级别的表征方法中,比如 Alexandru 等[13]在代码生成任务中,将上下文中给的代码和注释等信息用字符序列来表示,就采用了 one-hot 编码方式。Mou 等[14]运用 seq2seq 语言模型实现了由自然语言到代码片段的映射,在处理输入自然语言数据时同样适用了字符级别的 one-hot 编码。

2.2　词袋模型

词袋模型是一种经典的统计学表征方法[15],它的特点是以每个词为单位,不考虑词与词之间的相对顺序,仅仅统计词与词之间的共现次数。这种思路有点像把词不加区别对待地装进一个袋子里,因此得名“词袋”模型。

向量空间模型是信息检索领域的经典模型,也是一个典型的“词袋”模型。其核心思想是把对文本内容的处理简化为向量空间中的向量运算,并且以空间上的相似度表达语义的相似度。向量空间模型在软件工程领域有着广泛的应用,并被证明了在许多任务上具有良好的效果。Zhou 等[16]提出的 BugLocator 工具采用了改进版的向量空间模型 rVSM。它计算 bug 报告文件与源代码文件在空间上的相似度以定位可能存在 bug 的源文件,并考虑到了代码文件的长度以及历史上相似的 bug 对预测的影响,取得了很好的效果。Nguyen 等[17]在其设计的人工深度神经网络(DNN)中也采用了 rVSM 方法来处理输入的代码元素。该工作在代码分类任务上取得了超越传统方法的准确性。

在统计学中也有一些经典方法是基于词袋模型思想的,在代码表征方面应用最广的要数隐语义索引 LSI(Latent Semantic Indexing)模型以及 LDA(Latent Dirichlet Allocation)文档主题生成模型。这两种方法都旨在通过统计共现的方法捕捉隐含在文本的潜在信息并将之表示为向量,即所谓的隐语义向量,一些经典工作通过这两种方法为代码寻找合适的语义向量表示[18-19]。

2.3　分布式假说与词嵌入

词嵌入技术是一种新兴的向量表征方法,它起源于一个关键的设想——分布式假说。分布式假说是由 Harris 于 1954 年提出的[20],他认为在自然语言中,如果两个词的上下文很相近,即使这两个词本身并不相似,它们也很可能代表了相同的语义信息,即自然语言单词的语义是由其上下文决定的。这种假说解决了向量空间模型不能归纳近义词的弊端,并且考虑到了文本分布的顺序信息,因而在近期的研究中被广泛推广。

词嵌入技术也被称为分布式语义模型。在这种分布式模型中,单个的单词不再被处理为独立的标识符,而是用一个 d 维的上下文语义向量来表示。这样具有相似上下文的单词就会具有相近的向量表示。Mikolov 等[21]提出的 word2vec 模型是目前最有影响力的词嵌入模型。它用过神经网络来直接预测每个单词的分布式向量表示,因而也被称为可预测的分布式模型。word2vec 在自然语言的众多任务中都展示了超过传统分布式向量模型的性能。

word2vec 同样是在代码分析中应用最广泛的词嵌入模型。Ye 等[22]从 API 文档以及 API 教程网站上抽取了代码片段和自然语言描述,通过 word2vec 模型在一个公共的低维向量空间中训练出各自的向量表示,用空间的相对距离来代表文档之间的相似度,很好地解决了软件搜索以及故障定位任务中代码与自然语言之间的语义鸿沟问题。Nguyen 等[23]通过 word2vec 模型分别对 Java 和 C#代码中的 API 调用进行训练,并把两者的词向量表示映射到二维空间中,揭示了在不同的语言中,具有相似功能的 API 往往具有相似的二维空间分布,从而可以解决诸如代码迁移一类的语义映射问题。Chen 等[24]则通过该模型来处理近义词问题。他们从 stack overflow 等编程问答社区中抽取了大量的代码语料,通过词向量的空间分布来判定近义词,其性能超过了传统的基于语法的同义词判定方法。除了 word2vec 模型,研究者也设计出了一些特定于代码结构的词嵌入方法。Peng 等[25]提出适用于自然语言的词嵌入模型并不一定适用于分析代码,因为代码语义相比于自然语言具有更强的结构依赖性。他们通过抽象语法树获得代码的结构信息,并参考了自然语言处理中的“平滑性”概念[15],提出了适用于代码结构的“编码标准”,使相似的代码标识符具有相似的向量表示。Nguyen 等[26]则通过加权和的模式将 word2vec 模型与传统向量空间模型结合起来,并证明这种结合后的方法在代码样例搜索任务中具有更好的表现。

2.4 代码表征的粒度

除了合适的向量表示方法,选取合适的分析粒度也是非常有必要的。在软件工程领域中,代码表征的粒度一般可以分为以下几个级别。

2.4.1 字符级别

程序代码中的字符由英文字母、数字和运算符号构成,是软件分析中最细粒度的表征级别。该表征方法把源代码中的每一个单个的字符当成独立的符号并进行编码。由于字符的种类相当有限且不包含语义信息,一般采用独热编码方法对其进行编码。这种方法的优点在于表示起来相当方便,不会出现向量维度爆炸的问题,但是由于单个字符并不包含任何语义信息,程序的语义仅仅能够通过字符的排列顺序来表达,而这样通常是远远不够的。比如在 C 语言中,string 是一个数据类型名,而 strings 是一个方法名,他们拥有不同的程序语义,但是其字符序列却非常接近。在代码分析中,字符粒度的分析常常与 one-hot 编码一起使用[13-14]。

2.4.2 词条级别

代码中的词条类似于自然语言中的单词,由于它们都是由空格符来区分的,因此往往是最为直观的表示方法。词条级别的表征方法为每一个词条赋予一个词向量,词向量的表示方法多使用词嵌入或者向量空间模型。词条是代码中最小的语义单元,对词条建模得到的向量表示蕴含了高层的语义关系,因而可以较好的处理同近义词等问题。由于程序设计中的命名规则是非常自由的,比如变量名 string、string1、string123、s、str 可能是不同用户对于同一个变量的命名,因此对词条的建模很难控制词表的大小。针对这个问题研究者也提出了一系列处理同近义词的方法,在一定程度上提高了词条建模的实用性。词条级别是代码分析中最为常见的分析粒度,这可能与词条代表了代码最基本的语义单位有关。

2.4.3 AST 树节点级别

这类表征方法通过代码的抽象语法树获得代码的结构信息,并以语法树的节点为单位进行编码。与前几种方法相比,这种方法考虑到了代码的结构信息,因此其向量表示能够更好地反映程序的实际语义。并且由于抽象语法树在构建阶段对代码元素进行了大量的抽象处理,词表的大小也得到了很好的控制。但是这种表示方法也有其局限性: 这种向量表示需要被输入到树结构的神经网络中。这就造成了神经网络训练中的批处理问题: 由于语法树的大小是不定的,所以在训练过程中无法使用批处理技术对其进行加速,这会为训练过程带来巨大的计算复杂度。目前对这种粒度的实践有 Mou[7]以及 Gu 等的工作[27]。

3 深度学习模型

深度学习在近几年获得了长足的发展,几乎社会的各行各业都在尝试使用神经网络模型来解决领域内任务,各式新颖、富有创造性的网络结构不断涌现。然而截止到目前,神经网络模型最成功、最成熟的应用还是在图像识别与自然语言处理的任务上。这两大研究领域分别对应了目前最为经典、成熟的两个神经网络

模型——卷积神经网络(CNN)与循环神经网络(RNN)。目前软件工程领域对于深度学习的实践也是主要基于这两大经典模型来展开的。

3.1　卷积神经网络

卷积神经网络是一种前馈神经网络,但是采用卷积层来代替传统的全连接层。卷积层极大地减小了网络中权重矩阵的密度,提高了神经网络的计算效率。不仅如此,卷积层还具有提取数据特征的功能,比如在图像处理中,图像被表示为二维的矩阵,卷积层可以从二维矩阵中抽取出特征映射。一般来说,最底层的特征对应图像的边、角部分的特性。随着不断地卷积映射,这部分底层的特征被映射成人类可以理解的高层特征,比如识别出图像中的物体的颜色、大小等。卷积神经网络有三个结构上的特性: 局部连接、权重共享、以及空间或时间上的次采样。这些特性使得卷积神经网络有一定程度上的平移,缩放和扭曲不变性[28]。

软件工程对 CNN 网络的应用在很大程度上借鉴了图像处理领域的看图说话: 代码片段被看成是一幅图画,每个代码元素则对应画面中的一个物体。这样 CNN 就可以像捕捉图片中的边角信息一样捕捉源代码中的底层特征,再通过不断地卷积映射得到人类可以理解的高层语义特征。在代码分析任务中,由于 CNN 网络强大的抽取高层语义的能力,它常常被用在代码的摘要、标注,自动化注释生成等任务上。Mou 等[29]提出代码相比于自然语言具有严格的结构限制,因而从代码的 AST 树来代表这种结构信息,并设计了一套“编码标准”来计算代码向量表示。最后通过 CNN 网络捕捉源代码中的高层语义,实现了代码分类与搜索的功能。Nguyen 等[17]则设计了一种 4 层的人工神经网络,将输入的源代码逐层进行抽象与特征映射,最终得到了人类可以理解的项目级代码描述。

3.2　循环神经网络

由于前馈神经网络的输入和输出维度都是固定的,所以不能有效地处理可变长的序列化数据,循环神经网络就是为了解决这个问题而诞生的。在前馈神经网络中,连接只存在于层与层之间,层内的节点是无连接的,而循环神经网络则通过带自反馈的神经元,将同层的节点通过时序连接起来,不定长的序列信息长度被表示为时序序列的长度,因此能够处理任意长度的序列[30]。循环神经网络被广泛的应用在语音识别、机器翻译以及自然语言生成等任务上。

在循环神经网络的应用过程中,人们发现在处理具有长距离数据依赖的序列信息时,由于神经网络的反向传播特性,损失函数关于权重矩阵的梯度往往会爆炸性的增长或消失。这给神经网络的训练带来了很大的困难,这个问题被人们称为“长时依赖问题”。长时记忆神经网络(LSTM)[31]是循环神经网络的一个重要变体,它可以简单有效地解决长时依赖问题。LSTM 的关键是引入了一组记忆单元,允许神经网络可以学习得出哪些历史信息应该被记住,什么时候应该用新的数据更新记忆单元。这种有选择的计算方法显著地减轻了计算长距离依赖信息的计算负担,有效地了解决长时依赖问题。

由于代码具有与自然语言相似的性质与结构,对 RNN 网络的应用在软件工程研究中更加的普遍,也出现了很多特定于分析代码性质的网络结构。White 等[32]通过对比试验证明了 RNN 模型在代码推荐任务中有着远远超过传统 n-gram 方法的性能。Dam 等[33]提出了 DeepSoft 模型,通过运用代码的自然性,本地性等特性构建了一个基于 LSTM 神经网络的语言模型,从而实现了较好的代码生成效果。Alexandru[13]同样运用了 LSTM 结构来实现代码生成任务,不过他除了考虑前文代码之外,还考虑了代码的注释、标签等自然语言信息,并采用了逐字生成的策略。

3.3　Seq2seq 模型

该模型用于解决 seq2seq 问题。所谓的 seq2seq 问题指的是根据一个输入序列 i,来生成一个输出序列 o。该问题被广泛的应用在诸如机器翻译,文档摘要和问答系统等场景中。seq2seq 模型是一种编码-解码模型,所谓的编码指的是通过一个编码器将输入序列转化为一个固定长度的特征向量,而解码指的是通过一个解码器将这个特征向量再转化成输出序列。编码-解码模型最大的特点在于其高度的自由性: 在实际应用中,编码器和解码器都不是固定的,使用者可以根据需求自由地选择编码和解码模型。这种模型的局限性在于编码和解码之间唯一的联系就是一个固定长度的特征向量。因此如何合理设计编码与解码算法,使得特征向量可以有效的传递输入信息的特征,避免编解码之间的信息丢失成为了应用这种网络时需要首要考虑的问题。

许多代码分析任务也是一种 seq2seq 的问题,比如说代码迁移可以理解为从一种编程语言到另一种编程语言的翻译; 代码摘要和注释生成都可以理解为师编程语言到自然语言的映射等。Murali 等[34]的研究工作是这种模型的典型应用,他们提出了代码推荐工具 Bayou。这种工具融合了贝叶斯模型与深度学习的优点,整个模型通过隐语义向量链接,通过反向传播算法进行训练。用户可以输入诸如前文代码,需求描述

或者包信息等作为输入，这些信息通过基于贝叶斯的统计模型被转化成隐语义向量，再通过 RNN 网络实现代码信息的推荐。Gu 等[27] 借鉴了机器翻译的思路，使用 Seq2seq 模型在同一向量坐标系下共同训练了自然语言与代码的向量表示，实现了代码迁移的功能。

4 代码分析应用

代码分析任务的核心问题是填补自然语言与程序代码之间的语义鸿沟，因此在本质上可以看作是代码和自然语言之间的映射问题。本文根据这种映射关系将代码分析常见的应用场景归纳为以下三类。

4.1 自然语言到代码

在这种映射关系下，比较常见的应用有代码搜索和故障定位。代码搜索，即用户输入自然语言的查询，通过各种检索算法在代码库中匹配到满足查询条件的代码片段。而故障定位则是指用户根据故障报告中对故障的自然语言描述，计算文本相似度找到相似的源码文件，从而确定故障最可能出现的地方。

4.1.1 代码搜索

软件工程领域解决代码搜索问题的经典方法主要来源于信息检索技术，通过建立源码与查询之间的索引来实现搜索功能。然而正如前文所述的那样，这种简单的关键词匹配方法并不能很好地解决代码和自然语言之间的语义鸿沟问题，因此始终不能很好地支持实际编程的需要。为了解决这个问题，Gu 等[35] 提出了 DeepAPI 方法，这种方法把代码检索看成是一个机器翻译问题，采用深度语言模型把自然语言的查询翻译成对应的 API 序列。与传统的想法相比，该模型具有两大优点：一是通过词嵌入技术学习了词语的语义信息，实现了基于语义的代码搜索；二是用 seq2seq（序列到序列）的形式替代传统方法所采用的词袋模型，因而可以考虑到查询的顺序。

4.1.2 故障定位

现代大型软件库采用多分支的开发模式，并且面临这频繁的版本更迭，会不可避免地收到大量的故障报告。如何根据这些故障报告定位到故障实际发生的源码文件，从而方便开发人员进行针对性的勘误是一个非常具有挑战性的工作。Zhou 等[16] 提出了一种改进的向量空间模型 rVSM，这种模型考虑到了历史上的故障修改信息，计算故障报告与源文件之间的文本相似度，从而判定最可能出现 bug 的源码文件。在上一篇的基础上，Lam 等[36] 提出了 HyLoc 模型。该模型将 rVSM 模型与深度学习结合起来，一方面借助向量空间模型来计算故障报告与源码文件之间的文本相似度，另一方面用人工神经网络来学习得到两者之间的相关性，最终的相似度又文本相似度与相关性的加权和表示。该模型在开源代码库上运行的性能超过了传统基于信息检索模型的方法，证明了深度学习可以强有力的处理故障定位问题。

4.2 代码到代码

满足这种映射关系的应用主要是代码推荐与代码生成。代码的推荐与生成像是一对孪生的概念，目前学界中彼此之间并没有明显的界限可以划分。为了保证叙述的清晰，我们在这里定义，代码推荐所推荐的是部分的代码元素，比如用‘.’符号推荐某个对象最可能会调用的方法，或者根据一段需求描述生成代码模板；而代码生成特指生成可以直接运行的源代码的方法，比如根据前 10 行代码生成接下来的 5 行代码。代码迁移指的将某种语言的代码迁移到另一种语言中，由于不同语言有着不同的 API 使用方法，代码迁移一直是一个极具挑战性的任务。

4.2.1 代码推荐与生成

在过去的十年内，软件工程研究者普遍的做法是通过形式化公理或者统计学知识来完成代码补全任务，无论是在任务完成的准确率或是泛化能力上都只是差强人意，而深度学习在自然语言处理领域的出色表现给了解决代码补全问题的新的希望。Alexandru 等[13] 认为在处理代码生成任务时，应该被考虑到的上下文不止是开发者已经编写的代码，还包括注释和需求描述信息等。基于这种思想，他们从代码的抽象语法树中抽取丰富的上下文信息，并设计了双向 RNN 网络来进行训练。实验数据说明这种考虑复合上下文信息的方法在代码生成任务上比以往的方法有着更加优越的性能。White 等[32] 认为提高软件工程任务的关键在于提高代码表示过程中的表征能力，而深度学习恰恰擅长赋予代码表征以丰富的语义信息。因此他们结合了 RNN 网络与统计语言模型，并用来处理代码推荐问题。与之相似的，Raychev 等[37] 也认为深度学习模型可以和统计语言模型结合起来，这种结合被描述为一种 RNN 网络和传统 n-gram 算法所预测结果的加权和。

4.2.2 代码迁移

在代码迁移方面，Nguyen 等[23] 借助了词嵌入技术，用空间分布的相似性来衡量 API 之间的相似度。Gu 等[27] 通过模仿机器翻译的方式，通过将一种编程语言“翻译”成另一种语言，在代码迁移任务重取得了很好的效果。

4.3　代码到自然语言

满足这种映射关系的应用主要属于代码摘要的范畴。这一类的任务可以归结为给代码生成简短的自然语言描述,而代码分类知识在这种描述的基础上增加了分类的工作。这一类工作的本质是从代码到自然语言的映射。如上文所述,在实现代码与自然语言之间的映射时,常常会面临“语义鸿沟”问题,因此这类工作一般会先采用词嵌入的方法来填补“语义鸿沟”,然后再借助深度学习模型进行得到相应的自然语言描述。Nguyen 等[17] 通过经验研究来验证人工神经网络是否有完成代码摘要任务的能力。他们设计了一个四层的神经网络,每一层分别代表了从源码级别到自然语言摘要级别的特征表述。该研究向人们证明了人工神经网络有能力从底层源码中捕捉高层概念,使用深度学习技术来实现代码摘要任务是切实可行的。Jiang 等[38] 则运用了深度机器翻译(NMT) 的技术来自动生成 commit 信息。作者从 github 的项目中抽取 diff 信息以及相应的 commit 信息作为训练集,通过 NMT 模型进行有监督地训练,最终实现了对 commit 信息的预测。

5　分析与讨论

深度学习技术在代码领域的应用总体上还处在初步发展的阶段: 研究者们在各个任务上尝试了深度学习技术,其中的一些工具取得了超越传统方法的性能; 一些经验研究论证了 RNN、CNN 等网络结构在代码领域的可行性,并证明了深度学习技术具有填补代码与自然语言间语义鸿沟的能力[16] 。与此同时,也有一些研究者认为与传统统计学方法相比,深度学习对训练数据的需求成量级地增加,其性能却没产生明显的优势[39-40] 。由于深度学习缺乏像人类智能那样“举一反三”的能力,只适合处理单一的任务目标,因此在实际应用中如何将现实生活中的需求抽象成适用于深度学习的任务也是一大难题。因此这些研究者对于深度学习在软件领域的应用持保留态度。通过对目前学界研究的总结,我们将代码领域的深度学习的优点与现有问题归纳如下。

5.1　优越性

5.1.1　表征能力

词嵌入技术和分布式假说赋予了深度学习工具强大的表征能力,代码的高层语义和结构信息以向量形式被编码进入神经网络中,这种强表征能力带给了深度学习至少在三个方面的优越性。首先,这种表征能力有利于填补代码与自然语言间的语义鸿沟,这使得深度学习尤其适合于代码分析任务。其次,表征能力使得深度学习可以以一种无监督的“黑盒”模式进行特征提取,从而省去了繁琐的人工特征工程,排除了人为观察带来的误差[10] 。最后,强大的表征能力使得深度学习不像传统统计学方法那样受限于“词表越界”问题(OOV) ,很多研究工作都记录了深度学习生成或合成了训练集中不存在的模式[11,14] 。

5.1.2　训练性能

深度学习在图像识别和自然语言处理中的蓬勃发展向我们展示了其巨大的应用潜力: 在大量优质训练数据的前提下,深度学习在某些任务中可以达到甚至超过人类智能的表现。因此我们也有理由相信这种技术在代码领域的潜力,更何况在已有的研究中,深度学习已经在一些特定任务或数据集上取得了超越传统方法的性能。随着软件工程的发展,人们越来越重视代码数据的收集与表征,一些新颖的代码特定的神经网络机构也被提出,我们有理由期待深度学习方法在未来代码分析任务中的表现。

5.2　问题与讨论

虽然深度学习具有巨大的优越性与应用潜力,在目前的代码分析研究中,该技术仍然一些方面具有相当的局限性。下面列举了深度学习技术现存的一些问题,并尝试对未来的解决方案进行了讨论。

5.2.1　抽象的任务描述

深度学习需要人工定义损失函数来描述任务执行的边界和目标,并指导隐层参数的更新。这就要求所执行任务是相对单一,可用数学语言描述的。然而正如前文所分析的,代码具有高度的抽象性和复杂性,其大部分性质是很难用简明的数学语言来定义的。比如在代码摘要任务中,我们很难用数学公式来定义摘要好坏的判断标准。一个合理、具有广泛适用性的代码体系衡量标准成为了应用深度学习技术的迫切需求。

Scalabrino 等[41] 通过一个经验研究指出,软件工程中现有的评判标准,无论是代码的可读性还是复杂度,都无法胜任衡量人对于代码信息的理解程度。一些研究工作借鉴了机器翻译的评判标准,采用诸如 BLEU 和 Perplexity 等体系[42] 来进行判断,但是也有研究指出这些评判标准只是对人理解文本方式的模拟,并不能真正的衡量人实际的理解程度。由于神经网络需要一个评判标准来设计损失函数,因此评判标准的合理与否直接决定了模型训练的性能,如何设计有效的代码分析评价体系成为了代码理解的关键问题。

5.2.2 标注数据集的匮乏

标注数据集是解决有监督学习问题的根本。在图像识别和自然语言领域,正是有了大量高质量的人工标注数据集,才迎来了深度学习应用的井喷式发展[11]。由于理解代码并进行标注需要大量的专业知识,因此代码标注的成本和难度远远高于图片和自然语言。如何得到高质量的标注数据集成为了深度学习应用发展的一大难题。

要更好地实现代码分析任务,研究者应致力于建设一个服务于该任务的基准数据集。这个数据集需要有关于具体任务的标注信息,比如人工地标注出某个类的摘要信息。这样的数据集可以是研究者不必再烦恼如何为任务提供先验知识,更加专注于算法设计,并且为不同的方法提供了通用的测试平台。关于特定任务的竞赛也可以在基准数据集上展开,充分动用大众的力量来解决现有问题。

除了采用人工标注的数据集之外,尝试使用新的算法也是可行的思路。人工智能领域近期的一些研究指出,通过采用无监督、半监督和增强学习等方法[43],可以大大地减少深度学习算法对于标注数据的依赖。在代码标注"寸土寸金"的软件工程领域,尝试使用这些新的深度学习训练思路不失为一条可行之道。

5.2.3 缺少领域知识的理解

要理解一段代码,除了代码本身的性质之外,学习这段代码所实现的业务逻辑也是必不可少的。现有的代码理解方法大多从代码本身的性质入手,通过分析 API 或词条的分布或者注释等自然语言信息来实现对代码的理解,因而很难获得源代码背后所蕴含的业务逻辑。要真正实现深度的代码理解,了解必要的业务逻辑是必不可少的。

与深度学习不同,知识图谱是一种自底向上、从细节到整体的人工智能方法,它通过一阶谓词逻辑来表述现实世界的知识,通过"实体"的链接来把不同领域的知识链接到一起[44]。正如上文所述的,深度学习的一大缺陷是难以考虑到代码背后的业务逻辑关系,而这种显示存在的知识恰恰是知识图谱技术最善于描述的。因此将深度学习技术与知识图谱技术的优点相结合可能是一个很好的代码分析的思路。

6 结 语

通过对现有工作的总结,可以看到深度学习已经在代码生成及理解领域取得了可喜的进步,展示了极具竞争力的性能与广阔的发展前景。与此同时,现有的工作也表现出了一定的局限性,比如可标注数据集的缺乏以及对抽象任务的处理等。

在未来的代码分析研究中,深度学习将会扮演越来越重要的角色。软件工程领域的深度学习技术将会朝着标准化的方向发展,比如建立基准数据集以及设计出若干标准化的代码评判体系。与此同时,无监督学习、增强学习以及知识图谱等新兴人工智能技术将会得到广泛的实践,这些技术将会与现有深度学习技术进行有效互补。在可预见的未来,代码分析与理解一定会取得更加令人振奋的进展。

参 考 文 献

[1] Brooks Jr F P. No Silver Bullet Essence and Accidents of Software Engineering [J]. Computer, 1987, 20(4): 10-19.

[2] Bessey Al, Block Ken, Chelf Ben, et al. A few billion lines of code later: using static analysis to find bugs in the real world [J]. Communications of the Acm, 2010, 53(2): 66-75.

[3] Clarke E, Kroening D, Yorav K. Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking [C]//Design Automation Conference. ACM, 2003: 368-371.

[4] Cousot P, Cousot R, Feret J, et al. The ASTREé Analyzer [J]. Lecture Notes in Computer Science, 2005, 3444: 140-140.

[5] Knuth D E. Literate Programming [J]. Computer Journal, 1984, 27(2): 97-111.

[6] Hindle A, Barr E T, Su Z, et al. On the naturalness of software [C] // International Conference on Software Engineering. IEEE, 2012: 837-847.

[7] Mou L, Li G, Zhang L, et al. Convolutional neural networks over tree structures for programming language processing [C] // Thirtieth AAAI Conference on Artificial Intelligence. AAAI Press, 2016: 1287-1293.

[8] Scott S, Matwin S. Feature Engineering for Text Classification [C] // Proceedings of the Sixteenth International Conference on Machine Learning, ICML'99. 1999: 379-388.

[9] Shwartz-Ziv R, Tishby N. Opening the Black Box of Deep Neural Networks via Information [DB]. eprint arXiv: 1703.00810, 2017.

[10] Tishby N, Zaslavsky N. Deep learning and the information bottleneck principle [C] // Information Theory Workshop. IEEE, 2015: 1-5.

[11] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks [C] // International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012: 1097-1105.

[12] Wu Y, Schuster M, Chen Z, et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation [DB]. arXiv: 1609.08144, 2016.

[13] Alexandru C V. Guided code synthesis using deep neural networks [C] // ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2016: 1068-1070.

[14] Mou L, Men R, Li G, et al. On End-to-End Program Generation from User Intention by Deep Neural Networks [DB]. arXiv: 1510.07211, 2015.

[15] Zhang Y, Jin R, Zhou Z H. Understanding bag-of-words model: a statistical framework [J]. International Journal of Machine Learning & Cybernetics, 2010, 1(1-4): 43-52.

[16] Zhou J, Zhang H, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports [C] // International Conference on Software Engineering. IEEE, 2012: 14-24.

[17] Nguyen A T, Nguyen T N. Automatic Categorization with Deep Neural Network for Open-Source Java Projects [C] // International Conference on Software Engineering Companion. IEEE Press, 2017: 164-166.

[18] Panichella A, Dit B, Oliveto R, et al. How to effectively use topic models for software engineering tasks? An approach based on Genetic Algorithms [C] // International Conference on Software Engineering. IEEE Computer Society, 2013: 522-531.

[19] Poshyvanyk D, Marcus A, Rajlich V, et al. Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification [C] // IEEE International Conference on Program Comprehension. IEEE, 2006: 137-148.

[20] Harris Z S. Distributional Structure [M] // Papers on Syntax. Springer Netherlands, 1981: 146-162.

[21] Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality [C] // International Conference on Neural Information Processing Systems. Curran Associates Inc. 2013: 3111-3119.

[22] Ye X, Shen H, Ma X, et al. From word embeddings to document similarities for improved information retrieval in software engineering [C] // Ieee/acm, International Conference on Software Engineering. IEEE, 2016: 404-415.

[23] Nguyen T D, Nguyen A T, Phan H D, et al. Exploring API Embedding for API Usages and Applications [C] // International Conference on Software Engineering. IEEE Press, 2017: 438-449.

[24] Chen C, Xing Z, Wang X. Unsupervised Software-Specific Morphological Forms Inference from Informal Discussions [C] // Ieee/acm, International Conference on Software Engineering. IEEE, 2017: 450-461.

[25] Peng H, Mou L, Li G, et al. Building Program Vector Representations for Deep Learning [C] // International Conference on Knowledge Science, Engineering and Management. Springer, Cham, 2015: 547-553.

[26] Nguyen T V, Nguyen A T, Phan H D, et al. Combining Word2Vec with Revised Vector Space Model for Better Code Retrieval [C] // Ieee/acm, International Conference on Software Engineering Companion. IEEE, 2017: 183-185.

[27] Gu X, Zhang H, Zhang D, et al. DeepAM: Migrate APIs with Multi-modal Sequence to Sequence Learning [C] // Twenty-Sixth International Joint Conference on Artificial Intelligence. 2017: 3675-3681.

[28] Simard P Y, Haffner P, Lecun Y. Boxlets: a fast convolution algorithm for signal processing and neural networks [C] // Conference on Advances in Neural Information Processing Systems II. MIT Press, 1999: 571-577.

[29] Mou L, Li G, Jin Z, et al. TBCNN: A Tree-Based Convolutional Neural Network for Programming Language Processing [DB]. arXiv: 1409.5718, 2014.

[30] Graves A. Supervised Sequence Labelling with Recurrent Neural Networks [M]. Springer Berlin Heidelberg, 2012.

[31] Sepp Hochreiter, Jürgen Schmidhuber. Long Short-Term Memory [J]. Neural Computation, 1997, 9(8): 1735-1780.

[32] White M, Vendome C, Linares-Vasquez M, et al. Toward Deep Learning Software Repositories [C] // Mining Software Repositories. IEEE, 2015: 334-345.

[33] Dam H K, Tran T, Grundy J, et al. DeepSoft: a vision for a deep model of software [C] // ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2016: 944-947.

[34] Murali V, Qi L, Chaudhuri S, et al. Bayesian Sketch Learning for Program Synthesis [DB]. arXiv: 1703.05698, 2017.

[35] Gu X, Zhang H, Zhang D, et al. Deep API learning [C] // ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2016: 631-642.

[36] An N L, Nguyen A T, Nguyen H A, et al. Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N) [C] // Ieee/acm International Conference on Automated Software Engineering. IEEE, 2016: 476-481.

[37] Raychev V, Vechev M, Yahav E. Code completion with statistical language models [C] // Acm Sigplan Symposium on Programming Language Design & Implementation. ACM, 2014: 419-428.

[38] Jiang S, Armaly A, Mcmillan C. Automatically generating commit messages from diffs using neural machine translation [C] // Ieee/acm International Conference on Automated Software Engineering. ACM, 2017: 135-146.

(下转第 22 页)

表 6 领域本体构建结果对照表

数据来源	类个数	对象属性	数据属性	实例	总数
ETP-tourism	194	41	46	19	300
travel-ontology-ontologies	35	6	4	14	59
TravelOntology	89	66	29	17	201
UyTravelOntology	278	107	71	50	506

4 结 语

本文通过英文本体重用的方法来构建了维吾尔语领域本体,并提出了基于跨语言本体重用的维语本体构建方法,扩充了维文本体构建领域,实现了对重用的本体三元组提取。使用 Jena 开源工程搭建了领域本体构建平台,并证明了该平台的可用性和高效性。在接下来的研究中,主要是完善理论知识和领域本体构建的模型框架。通过扩充重用的本体集合,进一步研究提高所构建领域本体的规模。

参 考 文 献

[1] Studer R, Benjamins V R, Fensel D. Knowledge engineering: principles and methods [J]. Data & Knowledge Engineering, 1998, 25(1-2): 161-197.

[2] Trinkunas J, Vasilecas O. Building ontologies from relational databases using reverse engineering methods [C] // International Conference on Computer Systems and Technologies. ACM, 2007: 13.

[3] Tham K D, Fox M S, Gruninger M. A cost ontology for enterprise modelling [C] // The Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises. IEEE, 1994: 197-210.

[4] Fernández-López M, Gómez-Pérez A, Juristo N. METHONTOLOGY: from ontological art towards ontological engineering [C] // Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series. 1997.

[5] 唐爱民, 真溱, 樊静. 基于叙词表的领域本体构建研究 [J]. 现代图书情报技术, 2005, 21(4): 1-5.

[6] 李景. 领域本体的构建方法与应用研究 [D]. 北京: 中国农业科学院农业信息研究所, 2009.

[7] Hankiz Y, Seyyare I, Askar H. A Mixed Method for Building the Uyghur and Chinese Domain Ontology [C] // China Conference on Knowledge Graph and Semantic Computing. Springer Singapore, 2016: 124-129.

[8] 朱昊天. 基于跨语本体转换的维吾尔文舆情本体构建研究 [D]. 新疆大学, 2015.

[9] Perez A G, Benjamins V R. Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods [C] //Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99) Workshop KRR5: Ontologies and Problem-Solving Methods: Lesson Learned and Future Trends. 1999.

[10] 吴丹, 王惠临. 本体在跨语言信息检索中的应用机制研究 [J]. 图书情报工作, 2006, 50(9): 10-13.

[11] 司莉, 陈雨雪, 曾粤亮. 基于多语言本体的中英跨语言信息检索模型及实现 [J]. 图书情报工作, 2017, 61(1): 100-108.

[12] Benafia A, Mazouzi S, Benafia S. Building Ontologies from Text Corpora [C] // The International Conference on Engineering & Mis. ACM, 2015: 28.

[13] Search travel ontology [EB/OL]. [2017-04-05]. http://swoogle.umbc.edu/2006/.

[14] Search tourism ontology [EB/OL]. [2017-04-05]. https://protegewiki.stanford.edu/wiki/Protege_Ontology_Library.

[15] 向阳, 王敏, 马强. 基于 Jena 的本体构建方法研究 [J]. 计算机工程, 2007, 33(14): 59-61.

[16] Kavalec M, Vojtech S V. A Study on Automated Relation Labelling in Ontology Learning [C] // Ontology Learning from Text: Methods, Evaluation and Applications. IOS. 2005: 44-58.

[17] 李连倍, 刘胜全, 刘艳, 等. 基于跨语本体重用的维语本体构建方法 [J]. 计算机工程与应用, 2015, 51(11): 104-108.

(上接第 17 页)

[39] Fu W, Menzies T. Easy over hard: a case study on deep learning [C] // Joint Meeting on Foundations of Software Engineering. ACM, 2017: 49-60.

[40] Hellendoorn V J, Devanbu P. Are deep neural networks the best choice for modeling source code? [C] // Joint Meeting on Foundations of Software Engineering. ACM, 2017: 763-773.

[41] Scalabrino S, Bavota G, Vendome C, et al. Automatically assessing code understandability: How far are we? [C] // Ieee/acm International Conference on Automated Software Engineering. IEEE Computer Society, 2017: 417-427.

[42] Papineni K, Roukos S, Ward T, et al. BLEU: a method for automatic evaluation of machine translation [C] // Meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2002: 311-318.

[43] Li Y. Deep Reinforcement Learning: An Overview [DB]. arXiv: 1701.07274, 2017.

[44] Wang Z, Zhang J, Feng J, et al. Knowledge Graph and Text Jointly Embedding [C] // Conference on Empirical Methods in Natural Language Processing. 2014: 1591-1601.

[45] Shu C, Zhang H. Neural Programming by Example [C] // AAAI Conference on Artificial Intelligence. 2017.