计算机科学与探索
Journal of Frontiers of Computer Science and Technology doi: 10.3778/j.issn.1673-9418.2005048

代码特征自动提取方法

史志成[1,2]，周　宇[1,2,3+]
1. 南京航空航天大学 计算机科学与技术学院，南京 210016
2. 南京航空航天大学 高安全系统的软件开发与验证技术工信部重点实验室，南京 210016
3. 南京大学 软件新技术国家重点实验室，南京 210023
+ 通信作者 E-mail: zhouyu@nuaa.edu.cn

摘　要：神经网络在软件工程中的应用极大程度上缓解了传统的人工提取代码特征的压力。已有的研究往往将代码简化为自然语言或者依赖专家的领域知识来提取代码特征，简化为自然语言的处理方法过于简单容易造成信息丢失而引入专家制定启发式规则的模型往往过于复杂可拓展性以及普适性不强。鉴于以上问题，本文提出了一种基于卷积和循环神经网络的自动代码特征提取模型，该模型借助代码的抽象语法树(abstract syntax tree，AST)来提取代码特征，为了缓解因 AST 过于庞大而带来的梯度消失问题，还将 AST 进行切割，转换成一个 AST 序列再作为模型的输入。该模型利用卷积网络提取代码中的结构信息，利用双向循环神经网络提取代码中的序列信息。整个流程不需要专家的领域知识来指导模型的训练，只需要将标注类别的代码作为模型的输入就可以让模型自动地学习如何提取代码特征。应用训练好的分类编码器，在相似代码搜索任务上进行测试，Top1、NDCG、MRR 的值分别能达到 0.56、0.679 和 0.638，对比当下前沿的用于代码特征提取的深度学习模型以及业界常用的代码相似检测工具有显著的优势。

关键词：代码特征提取；代码分类；程序理解；相似代码搜索

文献标志码：A　中图分类号：TP391

史志成，周宇. 代码特征自动提取方法[J]. 计算机科学与探索
SHI Z C, ZHOU Y. The method of code features automated extraction[J]. Journal of Frontiers of Computer Science and Technology

The method of code features automated extraction

SHI Zhicheng[1,2], ZHOU Yu[1,2,3+]
1. College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
2. Key Laboratory for Safety-Critical Software Development and Verification, Ministry of Industry and Information Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
3. State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210023, China

Abstract: The application of neural networks in software engineering has greatly eased the pressure of traditional method of extracting code features manually. Previous code feature extraction models usually regard code as natural language or heavily depend on the domain knowledge of experts. The method of transferring code into natural language is too simple and can easily cause information loss. However, the model with heuristic rules designed by experts is

* The National Key Research and Development Program of China under Grant No. 2018YFB1003902 (国家重点研发计划); the National Natural Science Foundation of China under Grant No. 61972197 (国家自然科学基金); the Fundamental Research Funds for the Central Universities of China under Grant No. NS2019055 (中央高校基本科研业务费专项资金); the Qing-Lan Project of Jiangsu Province (江苏高校“青蓝工程”).

usually too complicated and lacks of expansibility and generalization. In regard of the problems above, this paper proposes a model based on convolutional neural network and recurrent neural network to extract code features through abstract syntax tree (AST). To solve the problem of gradient vanishing problem caused by the huge size of AST, we split the AST into a sequence of small ASTs and then feed these trees into the model. The model uses convolutional neural network and recurrent neural network to extract structure information and sequence information respectively. The whole procedure doesn't need to introduce the domain knowledge of experts to guide the model training and the model will automatically learn how to extract features through the codes which have been labeled classification. We use the task of similar code search to test the performance of the trained encoder, the metric of Top1,NDCG and MRR is 0.56,0.679 and 0.638 respectively. Compared with recent state-of-the-art feature extraction deep learning models and common similar code detection tools, our model has significant advantages.
Key words: code feature extraction; algorithm classification; program comprehension; similar code search

1 引言

在“大代码”背景的驱动下，如何高效地提取代码特征并对其进行编码以快速地处理海量数据的需求已日益迫切。如今，深度学习在自然语言处理的很多领域已经取得了突破性的进展，人工智能也逐渐从“感知智能”迈向“认知智能”，机器不仅能够感知客观世界所释放的信息，更能够对这些信息像人一样进行理解和分析。Hindle 等人[1]已经论证了程序语言和自然语言类似，具备众多可供分析的统计属性，代码语言可以和自然语言一样能够被机器理解和分析。因此，许多学者简单地将代码作为自然语言来处理。例如，代码被表示为一个字符串序列应用在克隆检测[2, 3]，漏洞定位[4]以及代码作者分类(code authorship classification)[5]的任务当中。尽管代码和自然语言有很多共性的特征，都是由一系列单词组成且都能表示成语法树的形式，然而代码有许多自己专有的特性，代码具有更强的逻辑结构，自定义的标志符，且标志符之间存在长距离依赖等等。简单地将代码视作自然语言进行处理难免会造成信息丢失。为了使模型更适合处理代码语言，部分学者借助软件工程的领域知识，制定了一些启发式规则来静态地提取代码特征，例如在应用程序接口（Application Programming Interface，API）推荐领域，Zhou 等人[6]就将用户查询后得到的反馈信息作为构建 API 向量的一维特征来提高查询效果，然而这些静态提取代码特征的方式有以下三个弊端：

（1）完全依赖研究人员的先验知识来提取特征，所提取的特征数目有限。

（2）当代码库过于庞大和复杂，特征规则的制定也会相应变的复杂，难以适用于对海量且结构复杂的代码数据进行处理。

（3）规则的制定往往是面向特定任务的，可迁移性差。

因此，近几年许多研究使用深度学习模型来自动提取代码的特征[7-10]，以摆脱对人工提取特征的依赖。这些模型很多借助代码的 AST 来提取特征，通过对 AST 内部节点之间的依赖关系进行提取，来获得代码的结构信息，而不是简单地根据代码的标志符序列提取语义信息。然而这些方法有一个明显的弊端，就是都将树的二维结构转化成一维的序列结构进行预处理，如 Alon 等人[10]将 AST 处理成一个路径集合，Hu 等人[8]以及 White 等人[9]直接使用先序遍历的方式获得 AST 节点的展开序列作为模型的输入，这些降维的处理方式一定程度上会造成节点之间某些依赖关系的丢失。

因此，为了更充分地提取代码的结构信息，部分学者提出了一些不降维而直接处理 AST 的树形深度学习模型。如 Wan 等人[11]使用 Tree-LSTM 进行注释生成，Mou 等人

[12]使用 TBCNN 模型进行代码分类。然而这些树形深度学习模型仍然有两个局限性：第一，与自然语言的长文本类似，在 AST 规模十分庞大的情境下，很容易在模型的训练过程中出现梯度消失的情况[13-15]，因此无论是 Wan 等人使用 Tree-LSTM 或者 Mou 等人使用的 TBCNN 都会很容易丢失那些存在长期依赖关系节点之间的部分信息。第二，Wan 等人使用的 Tree-LSTM 要将 AST 先转化为二叉树再进行编码，该预处理操作破坏了 AST 原始的结构且转化后树的深度将会大幅增加，加剧梯度消失问题。

在这篇论文中，我们结合 TBCNN 以及 Zhang 等人[16]提出的 ASTNN 模型构建了一个基于卷积和循环神经网络的自动代码特征提取模型 CVRNN。模型的输入是代码的 AST，为了提高 AST 初始词向量的质量，还针对 AST 设计了一个专门的词向量预训练模型，预训练词向量不仅能提升后面实验部分代码分类以及相似代码搜索的实验结果，还能在模型的训练过程中加速模型的收敛。借鉴 ASTNN 切割 AST 的思想，模型并未直接对整棵 AST 进行处理，而是根据“if”、“while”、“for”以及“function”这 4 个代表程序控制块的 AST 节点将 AST 切割成一系列子树，再利用 TBCNN 模型对这些子树进行卷积运算。子树的规模远小于原先的 AST，因此，可以有效地解决节点长期依赖导致的梯度消失问题。之后，采用双向循环神经网络[17]，内部神经元采用 LSTM[18]，来提取代码块之间的序列信息，将每个时间步生成的代码向量存放到一个向量矩阵中，最终采用最大池化得到代码向量。为了验证 CVRNN 模型生成的代码向量的质量，还在 CVRNN 模型基础之上设计了两个具体的应用模型，代码分类以及相似代码搜索模型，实验结果表明，CVRNN 模型不仅能够高效地自动提取代码的特征，且所提取的特征具有很强的迁移能力，不仅适用于代码分类任务，其分类训练后得到的编码器在相似代码搜索任务上也有很好的效果。

该论文的主要贡献是：

（1）提出了一个基于 AST 的词向量训练模型。

（2）提出了一个基于卷积和循环神经网络的自动代码特征提取的深度学习模型。

（3）构建了一个高效的用于搜索相似代码的系统，且该项目所使用的数据集以及代码已在 github 上开源(https://github.com/zhichengshi/CVRNN)。

2 代码特征提取模型

这部分主要分为两个子模型进行介绍：

（1）基于树的词向量训练模型。

（2）基于卷积和循环神经网络的特征提取模型。

2.1 基于树的词向量训练模型

词嵌入技术是将自然语言文本中的单词嵌入到低维空间的一种向量表示方法[19]。近几年，许多学者也将该方法成功应用在处理代码的任务中，如 API 推荐[20, 21]，漏洞检测[22]等等。然而这些词嵌入技术简单地将代码视为序列化的文本进行处理，往往采用类似 Word2vec[23]滑动窗口的方式建立特定单词的上下文情境，并以此为依据来生成相应的词向量，如 Zhang 等人[16]就通过先序遍历的方式获得 AST 的节点序列，然后使用 Word2vec 生成词向量，而代码元素之间的依赖关系其实是一种二维的树形结构，应用这种方法获取词的情境难免会导致代码的部分结构信息难以编码进最终生成的词向量中。Mou 等人[12]在对 AST 中节点进行编码时只将当前节点的孩子节点作为情境，这种情境信息的提取方式仍然不完备。因此本文借鉴 Perez 等人[24]构造树中节点上下文情境的方法，结合当前节点的祖先节点、兄弟节点以及子孙节点作为情境，在提取祖先节点以及子孙节点作为情境的过程中分别设置一个向上以及向下探测的参数来控制提取情境的深度，并设计了一个

词向量训练模型根据提取的情境来生成AST节点的词向量。为了提高相似代码搜索的准确率以及提高模型的训练速度，将不考虑AST叶子节点的信息即代码中的标志符信息，因此该模型更侧重于对代码结构信息的提取。相似代码搜索任务使用的数据集来源于leetcode编程网站，代码分类任务所使用的数据集是Mou等人[12]提供的104数据集，该数据集共包含104个类别的代码，每个类别下包含500个样本。在分类训练的过程中并没有使用AST叶子节点的信息，原因有二：（1）由于代码中的标志符是程序员自定义的，而据观察104数据集中的词汇与leetcode上的词汇并不重叠，使用叶子节点反而会引入噪音。（2）leetcode上的代码所使用的变量名仅仅起一个命名的作用并没有语义表示的功能，更多的是采用a、b、temp等不携带语义信息的单词作为变量名。AST的生成工具使用的是srcml(https://www.srcml.org/)，该工具不检查代码的正确性，因此即使编译不能通过的代码也能生成AST，提高了模型的容错性。在代码分类以及相似代码搜索的任务中所使用的代码都是C/C++编写的，而srcml用来表示这两种语言所定义的节点只有60个，加入叶子节点后词汇表的大小将扩充至上万，因此不使用叶子节点将大大提高模型的收敛速度。

图1展示了整个词向量的训练过程。选取节点b作为当前节点，向上探测寻找祖先节点的深度设置为1，向下探测子孙节点的深度设置为2，因此得到祖先节点情境信息a，兄弟节点情境信息c，以及子孙节点情境信息(d,e,f,g,h)，将这三类情境节点合并得到(a,c,d,e,f,g,h)作为节点b的情境信息。在模型的训练过程中，首先使用正态分布对词汇表矩阵进行初始化，得到矩阵$\mathcal{D}\in\mathbb{R}^{n\times f}$，其中$n$表示词汇表的大小，$f$表示词向量的维度。模型的输入分为两部分，情境节点以及当前节点在词汇表中的索引，通过查找词汇表可以得到情境节点以及当前节点的词向量表示，分别对应图1中的context embedding以及current vector，之后context embedding经过两层全连接神经网络得到变换后的情境向量context vector，使用交叉熵作为损失函数，Adam作为优化器进行梯度下降，为了使生成的context vector能够与currentvector进行交叉熵运算，输出层输出向量的维度与词汇表中向量的维度相同。整个计算过程的数学公式如下：

$$h = tanh(v \bullet W_1 + b_n)$$
$$y' = soft\max(h \bullet W_2)$$
$$H_{y'}(y) = -\sum_i y_i' \bullet lb(y_i)$$

其中$v\in 1\times f$表示情境向量，h表示经过隐层后得到的输出向量，$W_1\in\mathbb{R}^{f\times k}$是隐层的权值矩阵，用来将维度是$f$的向量映射成$k$维向量，$b\in 1\times k$是隐层网络的偏置项。$W_2\in\mathbb{R}^{k\times v}$是输出层的权值矩阵，$y'\in 1\times v$是输出层的输出向量，$y\in 1\times v$是当前节点的词向量，$y_i'$表示向量$y'$在第$i$个维度上的值，$y_i$同理。

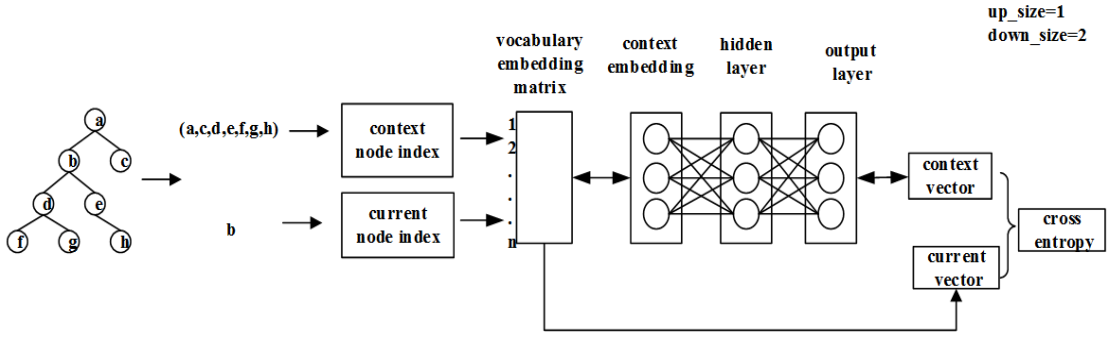

Fig.1 Node embedding training model

图1 词向量训练模型

2.2 基于卷积和循环神经网络的 CVRNN 模型

2.2.1 AST 切割

为了防止 AST 规模过大，带来梯度消失问题。并未直接将 AST 作为模型的输入，而是先将 AST 进行切割，得到 AST 的子树序列输入到模型之中，详细步骤如下。

首先，定义一个用于存储 AST 分裂节点的集合 S={“if”，“while”，“for”，“function”，“unit”}。其中“if”用来提取条件子树，“while”、“for”用于提取循环子树，“function”用来提取方法体子树，“unit”是未切割前 AST 的根节点，AST 剩余的节点都在“unit”子树上。图 2 展示了具体的切割算法。该算法以 AST 的根节点作为输入。首先用根节点初始化一个数组 T，然后通过深度优先遍历获得节点序列 N，对 N 中的节点进行遍历，若节点属于集合{“function”，“if”，“while”，“for”}，该节点连同其子节点将会被切割取出存储到 T 中。值得注意的是，当出现嵌套的情况，例如“if”子树中还包含“for”子树，“for”子树将会从“if”子树上提前切割下来，因为深度优先遍历将先访问“for”再访问“if”节点，两个独立的子树根节点将按序存储到 T 中。

Algorithm 1 split AST

```
input: the root of the AST
output: a sequence of small ASTs
function SPLIT(root)
    T=[root]
    N=[]
    N=dfs(root,nodes)
    for node in N do
        if node ∈ {function,if,while,for} then
            T.append(node)
            if node.parent ≠ None then
                parent=node.parent
                parent.remove(node)
            end if
        end if
    end for
    return T
end function

function DFS(root, nodes)
    nodes.append(root)
    for child in root do
        dfs(child, nodes)
    end for
    return nodes
end function
```

Fig.2 Algorithmof splitting AST
图 2 切割 AST 算法

2.2.2 卷积以及循环神经网络模型

在介绍 CVRNN 模型之前，先详细地阐述一下 Mou 等人[12]提出的基于树的卷积神经网络 TBCNN。图 3 展示了该模型执行的具体流程。首先应用词嵌入技术将 AST 中的节点转化成向量表示，不同的是，该方法只使用了被表示节点的孩子节点作为情境，没有考虑到兄弟节点以及祖先节点的信息，具体的词嵌入公式如下：

$$vec(p) = tanh(\sum_i l_i W_i \cdot vec(c_i) + b)$$

其中 p 表示双亲节点，c_i 表示 p 的第 i 个孩子节点，W_i 是 c_i 的权值矩阵，$l_i = \frac{\#node\ under\ c_i}{\#node\ under\ p}$ (孙子节点的数目除以孩子节点的数目)是系数，b 是偏置项。然后通设置一个固定深度的滑动窗口遍历整个 AST 再引入最大池化得到一个形状与原先一样的 AST，接着使用动态池化[25]以及两层全连接神经网络得到最终的代码向量。

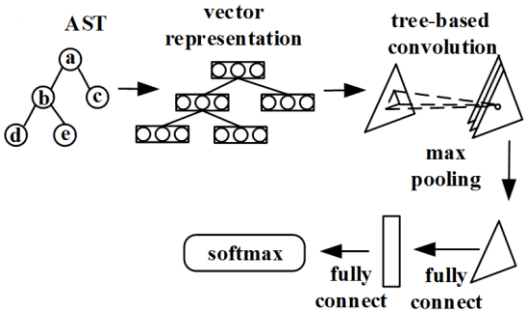

Fig.3 TBCNN model
图 3 TBCNN 模型

图 4 展示了 CVRNN 的模型架构，图 5 以伪代码的形式展示了该模型具体的执行流程。首先根据 2.1 得到的词向量表将 AST 中的节点表示为向量，再使用 TBCNN 模型对 AST 进行卷积得到编码后的向量，然后使用双向循环神经网络并以 LSTM 作为神经元对得到的向量序列进行编码以提取代码的序列信息。标准的 RNN 是单向的，其编码受限于过去的信息，采用双向 RNN 则能够同时使用过去和未来两个方向的信息。在后面的实验中，将会对这两种不同的策略进行对比。给定一棵代码的 AST，假设该 AST 被切割成 n 棵子树，这些子树经过 TBCNN 编码后将得到一个向量序列 $TV \in \mathbb{R}^{n \times k}$，$k$ 表示向量的维

度，n 表示子树的数目。在某个时间步 t，LSTM 的计算公式如下：

$$i_t=\sigma(W_i[h_{t-1},e_t]+b_t)$$
$$f_t=\sigma(W_f[h_{t-1},e_t]+b_f)$$
$$o_t=\sigma(W_o)[h_{t-1},e_t]+b_o$$
$$\widetilde{c_t}=tanh(W_c[h_{t-1},e_t]+b_c)$$
$$c_t=f_t*c_{t-1}+i_t*\widetilde{c_t}$$
$$h_t=o_t*tanh(c_t)$$

其中 σ 是 sigmoid 激活函数，c_t 表示新状态 $h_t\in\mathbb{R}^{1\times m}$ 表示输出向量，由当前时间步两个方向的向量拼接而成的向量，W_i,W_f,W_o 是权值矩阵，b_t,b_f,b_o 是偏置项。i_t 表示输入门，决定 e_t 中哪一部分将被保留添加到 c_t 中，f_t 表示遗忘门，决定 e_t 的哪一部分将被遗忘，o_t 是输出门用来计算输出向量。整个双向 RNN 的计算可以形式化如下：

$$\overrightarrow{h_t}=\overrightarrow{LSTM}(e_t),t\in[1,n]$$
$$\overleftarrow{h_t}=\overleftarrow{LSTM}(e_t),t\in[1,n]$$
$$h_t=[\overrightarrow{h_t},\overleftarrow{h_t}],t\in[1,n]$$

至此，每棵子树将会被转化成一个向量 $b\in\mathbb{R}^{1\times 2m}$。这些向量将会被存放到一个矩阵 $BV\in\mathbb{R}^{n\times 2m}$ 中，考虑到不同的子树的重要程度可能不同，例如“if”代码块中的语句要多于“for”代码块中的语句，直观上该“if”子树所携带的信息将高于“for”子树，采用均值池化则两个代码块所赋予的权值则相同，因此该模型使用最大池化得到代码向量 $r\in\mathbb{R}^{1\times 2m}$，而没有选择均值池化。

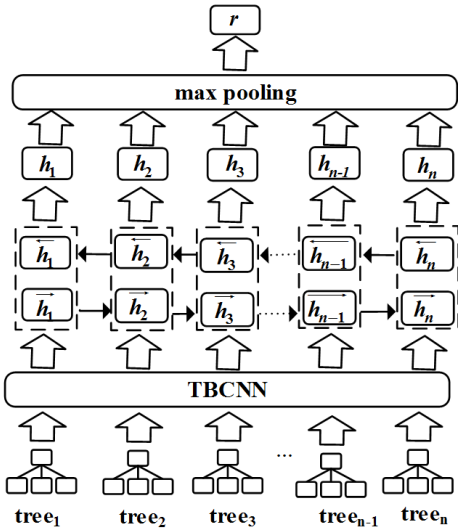

Fig.4 CVRNN model

图 4 CVRNN 模型

Algorithm 2 CVRNN

input: the small tree root node sequence, R

output: the code vector, r

```
function CVRNN(R)
    //get the length of small tree sequence
    n = len(R)
    // get the output of TBCNN, TV ∈ R^{n×k}
    TV = TBCNN(R)
    // get the output of biRNN, BV ∈ R^{n×2m}
    BV = biRNN(TV)
    // get code vector, r ∈ R^{1×2m}
    r = [max(BV_1), ..., max(BV_2m)]
    return r
end function
```

Fig.5 Algorithm of CVRNN

图 5 CVRNN 模型算法

3、验证模型

这部分主要介绍两个基于CVRNN的具体的应用模型，代码分类模型以及相似代码搜索模型，其中相似代码搜索模型又是基于代码分类模型训练好的编码器的。

3.1 代码分类模型

代码分类所使用的数据集是 Mou 等人[12]构建的 104 数据集。原始数据集共包含 52000 条数据。训练数据是随机从中抽取的 51000，剩余的 1000 条作为验证数据，来监控训练过程中是否发生过拟合。

代码分类模型如图 6 所示，经过 CVRNN 的编码，任意代码段都能转换成一个固定长度的向量表示 r，为了使模型适配 104 代码分类任务，在该模型的基础之上再添加了一层全连接网络，将 r 映射成维度是 104 的向量 r'。使用 one-hot 方法生成每个代码的标记向量 l，l 的维度 104，若代码属于第 5 类，则 l 位置 5 上的值为 1，其余位置为 0。选取交叉熵作为损失函数，使用 Adam 作为优化器。

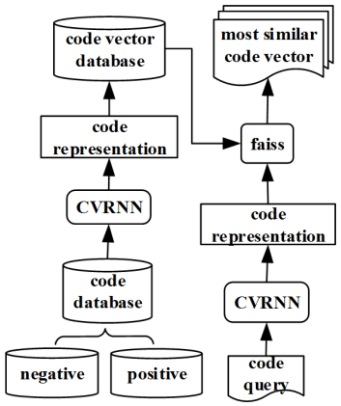

Fig.6 Code classification

图 6 代码分类

3.2 相似代码搜索模型

相似代码搜索使用的数据集来源于leetcode，没有在实际生产环境中去寻找功能相似的代码的原因是这种人为的筛选方式具有很强的主观性，功能划分的粒度很难把握。比如同样是一个排序代码，一个使用快速排序实现，另一个使用归并排序实现，很难判定两个是否是同一个类别。而使用leetcode 编程网站上同一个题目的不同题解作为判断功能是否相似是一个很客观的标准。为了保证功能绝对一致，每个题解都输入到在线的 IDE 中进行了测试，该网站会提供几百到上千个测试用例，所筛选得到的题解能通过全部的测试用例。选择其中一个题解用作查询，另一个用作匹配对象放入数据库中，该数据库对应图 7 中的正样本。在真实环境下，数据库中会包含更多的代码，因此，为了使相似代码搜索任务更贴近真实环境，该实验对代码的搜索空间进行了扩充。使用了 104 数据集中的 52000 条数据作为负样本加入到待搜索的数据库中。

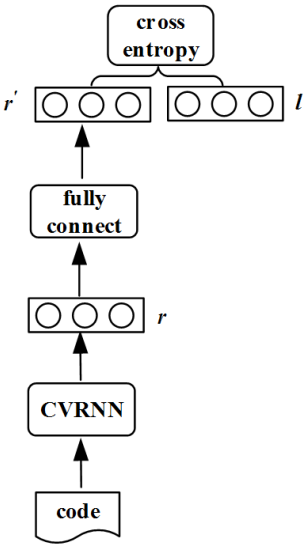

Fig.7 Similar code search

图 7 相似代码搜索

图 7 左边是代码向量的生成过程，应用训练好的CVRNN模型对数据库中的代码进行编码，最后将代码向量存放到一个数据库中。图 7 右边是代码查询的过程，模型的输入是一个用作查询的题解，经过 CVRNN 编码后得到代码向量。借助 Facebook 所提供的 faiss[26]工具将数据库中的代码向量基于与查询向量之间的欧式距离进行排序，选择前 10 个搜索结果作为结果统计样本。因为在搜索时，数据库中的代码都已经是向量的形式，所以搜索时间很快，时间可以忽略不计，在普通笔记本上的搜索时间小于 2ms。

4、实验评估

在这一部分，将基于 4 个研究问题对CVRNN 模型的性能进行评估。

（1）CVRNN 模型在分类任务上的效果如何?

（2）CVRNN 生成的代码向量是否满足相似度越高，彼此之间的几何距离就越短的性质?

（3）CVRNN 在相似代码搜索任务上的实验结果如何?

（4）CVRNN 预训练词向量以及双向 RNN 模块对模型产生怎样的影响?

4.1 对比模型

TBCNN 是第一个用于处理 104 代码分类任务的深度学习模型。前文已经详细介绍过 TBCNN 模型的执行流程。因为 TBCNN 模型内嵌在 CVRNN 模型之中，为了对比的公平性，在实验过程中，TBCNN 与 CVRNN 中使用的 TBCNN 在参数配置上完全相同，卷积层的数目、卷积维度都相同。在相似代码搜索任务中，TBCNN 模型生成的代码向量对应图 3 中倒数第二个全连接神经网络的输出向量，维度是 400。

ASTNN 模型据我们所知是当下在 104 分类任务上取得最好结果的模型。该模型与 CVRNN 模型的主要区别就是对 AST 序列进行编码的方法不同。该模型主要采用如下公式对 AST 中的节点进行自下而上的编码：

$$h = \sigma(W_n^T v_n + \sum_{i \in [1,C]} h_i + b_n)$$

其中 h 表示当前节点更新后的状态，W_n 是权值矩阵，v_n 表示当前节点的初始化词向量，h_i 表示当前节点第 i 个孩子节点的状态，C 表示当前节点孩子节点的数目，b_n 是偏置项。

然后对 AST 中的所有节点使用最大池化获得 AST 的向量表示。接着将得到的 AST 向量序列输入到双向 RNN 中，再经过最大池化得到整棵 AST 的向量表示。因为 CVRNN 也使用双向 RNN 来提取代码的序列信息，为了对比的公平，ASTNN、CVRNN 均使用 LSTM 作为神经元，且隐层单元都设置为 200，因此两个时间方向拼接之后得到的代码向量维度也都是 400。

为了对比公平，CVRNN、TBCNN、ASTNN 共享同一张预训练词向量表。除了深度学习模型，CVRNN 模型还与 3 个业界常用的相似代码检测工具进行了对比，分别是 moss(https://theory.stanford.edu/~aiken/moss/)[27](measure of software similarity)、jplag (https://jplag.ipd.kit.edu/)以及 sim(https://dickgrune.com/Programs/similarity_tester/)。moss 是斯坦福开发的利用文件指纹技术来确定程序相似性的系统。jplag 以程序覆盖率作为代码相似的度量指标，在覆盖的过程中，代码被分为多个相似的片段，通过计算平均覆盖率和最大覆盖率作为最终代码相似度的值。sim 主要根据代码所使用的词汇来计算代码之间的相似度，该工具广泛应用于检查大型软件项目中的重复代码。此外数据还被部署在一个开源的代码搜索引擎 searchcode(https://github.com/boyter/searchcode-server)上进行测试，该搜索引擎也是以代码作为查询语句在数据库中搜索相似的代码，然而所有反馈结果都是 0，因此没有在后面的实验结果统计表中列出。

4.2 度量指标

在代码分类任务中，仅选取准确率作为度量指标。在相似代码搜索中，应用以下三个在搜索领域广泛使用的度量指标来衡量搜索结果：

(1) top@k

$$\mathbf{top@}k = \frac{rel(k)}{|Q|}$$

Q 表示查询样本，$|Q|$ 表示查询样本的数目，若前 k 个候选样本中有一个与查询匹配，则认为该查询命中结果，$\mathrm{rel}(k)$ 表示命中的查询样本的数目。

(2) MRR

$$\mathbf{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{FRank_i}$$

$FRank_i$ 表示第 i 个样本第一个命中的位置，Q 以及 $|Q|$ 的定义同上。

(3) NDCG

$$\mathbf{DCG} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\sum_{j=1}^{n}\frac{2^{rel_{ij}}-1}{lb(i+1)}$$

$$\mathbf{NDCG} = \frac{\mathbf{DCG}}{\mathbf{IDCG}}$$

n 表示为限定反馈数目所设置的阈值，默认为 10。rel_{ij} 表示第 j 个候选样本与查询样本 i 的相关程度，在相似代码搜索任务中，若候选样本与查询样本匹配则值为 1，反之为 0。Q 以及 $|Q|$ 的定义同上。**IDCG** 表示 **DCG** 的最优计算结果，假设 n 为 5，前 5 个搜索结果与查询样本的相关程度分别为(0,0,1,0,0)，则使用(1,0,0,0,0)计算 **IDCG** 的值，(0,0,1,0,0)计算 **DCG** 的值，以此得到 **NDCG** 的值。

4.3 实验结果

研究问题 1：CVRNN 模型在分类任务上的效果如何？

图 8 展示了经过每一轮训练之后，三个深度学习模型在验证集上的分类准确率。可以看出 CVRNN 模型相对于其它两个模型收敛速度更快，经过 1 轮训练之后，CVRNN 模型在分类的精度就达到了 76.5%，而 ASTNN 以及 TBCNN 则分别是 45.2%和 65%(对应图 7 纵轴的截距)。经过 30 轮训练之后，CVRNN 模型的分类精度达到 94.4%，而 ASTNN 以及 TBCNN 则分别是 90.7%和 80.9%。在这 30 轮的训练过程中，可以看出，CVRNN 的每一轮的验证精度均高于另外两个深度学习模型。因此，CVRNN 在代码分类任务上对比这两个深度学习模型有明显的优势。

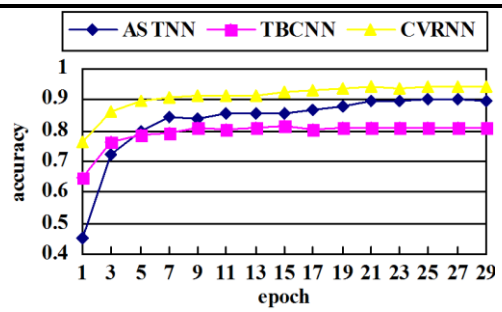

Fig.8　Valid accuracy of three deep learning models in each epoch

图 8　3 个深度学习模型每训练一轮的验证精度

研究问题 2：生成的代码向量是否满足相似度越高，彼此之间的几何距离就越短的性质？

图 9 展示了 leetcode 上 100 个问答对生成的代码向量使用 TSNE（t-Distributed Stochastic Neighbor Embedding）方法降维打印后的图片，可以看出很多问答对生成的代码向量之间的距离很近，因此也回答了研究问题 2，使用 CVRNN 模型可以将功能相似的代码向量聚集在一起，相似度越高，彼此之间的几何距离就越短。

Fig.9　Leetcode code vector

图 9　Leetcode 代码向量

研究问题 3：CVRNN 在相似代码搜索任务上的实验结果如何？

表 1 展示了各个模型在相似代码搜索上的实验精度，3 个深度学习均采用第 30 轮训练结束之后的编码器。在使用 jplag 模型进行测试时，我们统计了该模型由平均相似度以及最大相似度分别得到的实验结果。实验结果表明：深度学习模型的实验效果相对于 3 个传统的相似度检测工具有明显的优势，CVRNN 在各项度量指标上均高于其它任何一个模型。尽管在分类精度上 TBCNN 要劣于 ASTNN，ASTNN 要高出 9.8 个百分点，然而除了 Top1 二者的精度相等之外，TBCNN 其他几项的精度均高于 ASTNN。这说明，尽管可以借助代码分类任务训练编码器，然而模型在两个任务上的性能并不是正相关的，即在代码分类任务上精度越高并不代表相似代码搜索的效果越好，然而从表 1 以及图 7 可以看出，CVRNN 模型在两个任务上的效果都是最优的。

表 1　相似代码搜索准确率

Table 1　similar code search accuracy

model	Top1 /%	Top3 /%	Top10 /%	NDCG /%	MRR /%
jplag(max)	25	27	31	27.8	26.8
jplag(avg)	25	29	31	28.2	27.4
moss	32	33	33	32.6	32.5
sim	44	46	46	45.2	45.0
ASTNN	50	58	62	56.0	54.0
TBCNN	50	62	75	62.2	58.1
CVRNN	**56**	**69**	**81**	**67.9**	**63.8**

研究问题 4：CVRNN 预训练词向量以及双向 RNN 模块对模型产生怎样的影响？

定义如下 4 个模型：

模型 1(without biRNN)：将双向 RNN 直接从模型移除。

模型 2(RNN)：将双向 RNN 替换成双层单向 RNN。

模型 3(random)：使用正态分布随机初始化词向量。

模型 4(CVRNN)：默认配置，使用双向 RNN 以及预训练的词向量。

模型 1 直接将双向 RNN 移除，表示整个 CVRNN 模型将不提取代码的序列信息，

而模型 2 将双向 RNN 替换成双层单向 RNN 则表示削弱模型提取序列信息的能力。表 2 展示了各个模型在代码分类上取得的精度，可以看出模型 1 在失去提取代码序列信息模块之后，分类精度直接从 94.4%降到了 73.8%，对比模型 2 的 92.5%有很大的差距。表 3 展示了各个模型在相似代码搜索任务上的结果，可以发现，模型 1 虽然在失去提取序列信息的模块之后在代码分类任务上要明显逊色于模型 2，在相似代码搜索各项度量指标上的值却都高于模型 2，该结果也与问题 3 得到的结论相符合，模型在代码分类以及相似代码搜索上的性能并不是成正相关的。从表 2 以及表 3 可以看出，模型 4 在两个任务上的的性能都要高于模型 1 和模型 2。图 10 展示了模型 3 以及模型 4 在训练过程中每一轮在验证集上的分类精度。可以看出使用预训练词向量的模型 4 将收敛的更快，经过第一轮训练之后，模型 3 的分类精度为 72%，而模型 4 精度能达到 76.5%。并且在每一个训练轮次之中，使用预训练向量的模型都有更高的精度，使用预训练词向量的模型 4 最终达到 94.4%的分类精度，而随机初始化词向量的模型最终的分类精度是 91.4%。从表 3 也可以看出，预训练词向量对提升相似代码搜索的性能也有贡献。

表 2　CVRNN 代码分类消融实验

Table 2　CVRNN ablation research in code classification

model id	accuracy/%
1	73.8
2	92.5
3	91.4
4	**94.4**

表 3　CVRNN 相似代码搜索消融实验

Table 3　CVRNN ablation research in similar code search

model id	Top1 /%	Top3 /%	Top10 /%	NDCG /%	MRR /%
1	49	63	75	61.2	56.9
2	46	58	68	56.8	53.3
3	55	68	79	66.6	62.7
4	**56**	**69**	**81**	**67.9**	**63.8**

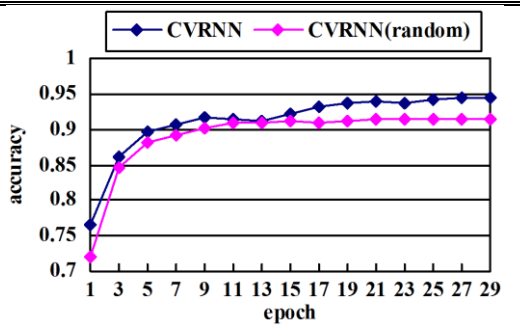

图 10　随机初始化词向量以及使用预训练词向量每一轮验证精度对比

Fig.10　the valid accuracy compare between random initializing node embedding and pre-training embedding

5、相关工作

如何对代码进行表示一直是软件工程领域研究的一个重要的问题。

传统的信息检索以及机器学习方法主要将代码作为纯文本进行处理。输入到模型之中的是一个被规范化处理好的标志符序列[2]，SourcererCC[3]在此基础上增强了对代码语义的挖掘，使用一种反向索引(inverted index)技术对标志符的序列信息进行挖掘。与 SourcererCC 不同的是，Deckard[28]模型选择增强对代码结构信息的挖掘，在输入的数据中加入了代码的语法结构信息。

不少研究人员借助 AST 所提供的代码元素之间的依赖关系来增强对代码信息的提取。例如：Zhou 等人[7]借助 AST 来扩充方法体内部元素的情境信息以增强代码段的语义，提高注释生成的效果。Yan 等人[29]应用类似方法进行代码推荐工作。除了借助 AST 扩充代码元素语义之外，更多的模型选择将 AST 整体或切分后得到的子结构序列输入到模型之中。例如：Hu 等人[8]就通过添加括号的方式来限定 AST 中节点的作用域，将 AST 转化成一个节点序列来生成代码对应的注释；Alon 等人[10]借助函数体的 AST 路径来生成相应的函数名；White 等人[9]则分别根据代码的标志符序列以及 AST 节点序列得到代码的语义向量和结构向量，

根据这两个向量对代码克隆检测展开研究。

然而这些借助 AST 的方法都是将树形结构转化为一维的序列结构，破坏了元素之间的依赖关系。因此，部分学者提出了不降维而直接处理 AST 的树形深度学习模型。与 White 的方法类似，Wan 等人[11]也将代码分为两部分作为输入，使用循环神经网络(RNN，recurrentneuralnetwork)对代码的标志符序列进行编码得到语义向量，不同的是，Wan 等人使用 Tree-LSTM 模型[30]去处理 AST，而并没有简单地通过将 AST 转化成节点序列的方式得到代码的结构向量。Wei 等人[31]同样使用Tree-LSTM对克隆检测展开研究。Mou 等人[12]提出了一种基于树的卷积神经网络模型 TBCNN，该模型直接在 AST 上进行卷积计算，然后使用动态池化技术将不同规格的 AST 压缩成代码向量，该向量能够很好地提取代码中的结构信息，在 104 类代码分类任务上能够达到 94%的精度。为了解决上述树神经网络的问题，一种方法就是获得代码的控制流图以及数据依赖图，静态地建立节点与节点之间的联系，将代码表示成一种图的数据结构，最终使用图嵌入技术[32]对代码进行表示。例如 Allamanis 等人[33]就通过相同的函数名以及变量名来静态地建立这些函数以及变量之间存在的依赖关系，Tufano 等人[34]则直接构建程序的控制流图来补充节点之间的控制关系。然而程序中元素的依赖关系往往要借助编译后代码的中间表示或者字节码才能得到[35]。在现实环境中，很多代码不能够被编译，因此这些方法所适用的范围将会受到很大的限制，而且代码图结构的定义以及对于图特征的提取也十分依赖专家的领域知识，模型往往很复杂，设计代价较高。

对比这些工作，本文的工作集中于借助 AST 对代码的结构以及序列信息进行提取。此外本文提出了一种新的获得代码向量的方法，该方法不使用完整的训练好的代码分类模型，仅截取模型中的一部分来生成代码向量，得到的代码向量可满足功能越相似，几何距离就越短的性质。

6、结束语

本文提出了一个基于卷积以及循环神经网络的自动代码特征提取模型 CVRNN。通过对已经标记类别的代码训练编码器，编码器将自动地学会如何提取代码特征，对比当下两个前沿的代码分类模型有显著的优势，在此过程中嵌入的预训练词向量不仅能提高分类的精度，更能够加快整个模型的拟合速度。在相似代码搜索任务上，CVRNN 无论是对比近几年提出的前沿的深度学习模型还是广泛应用于业界的代码相似度检测工具都有显著的优势。

References:

[1]Hindle A, Barr E T, Su Z et al., On the naturalness of software[C]//2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012: 837-847.

[2]Kamiya T, Kusumoto S.Inoue K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Transactions on Software Engineering,2002, 28(7). 654-670.

[3]Sajnani H, Saini V, Svajlenko J et al., SourcererCC: Scaling code clone detection to big-code[C]// Proceedings of the 38th International Conference on Software Engineering, 2016: 1157-1168.

[4]Zhou J, Zhang H.Lo D, Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports[C]//2012 34th International Conference on Software Engineering (ICSE), IEEE, 2012: 14-24.

[5]Frantzeskou G, MacDonell S, Stamatatos E et al. Examining the significance of high-level programming features in source code author classification[J]. Journal of Systems Software,2008, 81(3). 447-460.

[6]Zhou Y, Yang X, Chen T et al. Boosting API Recommendation with Implicit Feedback[J]. arXiv:.01264,2020.

[7]Zhou Y, Yan X, Yang W et al. Augmenting Java method comments generation with context information based on neural networks[J]. Journal of Systems Software,2019, 156(328-340.

[8]Hu X, Li G, Xia X et al., Deep code comment generation[C]//Proceedings of the 26th Conference on Program Comprehension, 2018: 200-210.

[9]White M, Tufano M, Vendome C et al. Deep learning code fragments for code clone detection[C]//

2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2016: 87-98.
[10]Alon U, Brody S, Levy O et al. code2seq: Generating sequences from structured representations of code[J]. arXiv:.01400,2018.
[11]Wan Y, Zhao Z, Yang M et al., Improving automatic source code summarization via deep reinforcement learning[C]//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018: 397-407.
[12]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, 2016.
[13]Bengio Y, Simard P.Frasconi P. Learning long-term dependencies with gradient descent is difficult[J]. IEEE transactions on neural networks, 1994, 5(2). 157-166.
[14]Hochreiter S. The vanishing gradient problem during learning recurrent neural nets and problem solutions[J]. International Journal of Uncertainty, Fuzziness Knowledge-Based Systems,1998, 6(02). 107-116.
[15]Le P.Zuidema W. Quantifying the vanishing gradient and long distance dependency problem in recursive neural networks and recursive LSTMs[J]. arXiv:.00423,2016.
[16]Zhang J, Wang X, Zhang H et al., A novel neural source code representation based on abstract syntax tree[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019: 783-794.
[17]Schuster M.Paliwal K K. Bidirectional recurrent neural networks[J]. IEEE transactions on Signal Processing,1997, 45(11). 2673-2681.
[18]Gers F A, Schmidhuber J.Cummins F. Learning to forget: Continual prediction with LSTM[C]//presented at the 9th International Conference on Artificial Neural Networks: ICANN '99, 1999.
[19]Henkel J, Lahiri S K, Liblit B et al., Code vectors: Understanding programs through embedded abstracted symbolic traces[C]//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018: 163-174.
[20]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.
[21]Nguyen T D, Nguyen A T, Phan H D et al., Exploring API embedding for API usages and applications[C]//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, 2017: 438-449.
[22]Pradel M.Sen K J T D. Department of Computer Science. Deep learning to find bugs[J]. 2017.
[23]Mikolov T, Chen K, Corrado G et al. Efficient estimation of word representations in vector space[J]. arXiv:1301.3781,2013.
[24]Perez D.Chiba S, Cross-language clone detection by learning over abstract syntax trees[C]//2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), IEEE, 2019: 518-528.
[25]Socher R, Huang E H, Pennin J et al., Dynamic pooling and unfolding recursive autoencoders for paraphrase detection[C]//Advances in neural information processing systems, 2011: 801-809.
[26]Johnson J, Douze M.Jégou H. Billion-scale similarity search with GPUs[J]. IEEE Transactions on Big Data,2019.
[27]Schleimer S, Wilkerson D S.Aiken A, Winnowing: local algorithms for document fingerprinting[C]// Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003: 76-85.
[28]Jiang L, Misherghi G, Su Z et al., Deckard: Scalable and accurate tree-based detection of code clones[C]//29th International Conference on Software Engineering (ICSE'07), IEEE, 2007: 96-105.
[29]Yan X, Zhou Y.Huang Z. Code snippets recommendation based on sequence to sequence model[J]. journal of frontiers of computer science and technology,2020,05, 731-739.
[30]Tai K S, Socher R.Manning C D. Improved semantic representations from tree-structured long short-term memory networks[J]. arXiv:.00075,2015.
[31]Wei H.Li M, Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code[C]//IJCAI, 2017: 3034-3040.
[32]Ou M, Cui P, Pei J et al., Asymmetric transitivity preserving graph embedding[C]//Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016: 1105-1114.
[33]Allamanis M, Brockschmidt M.Khademi M. Learning to represent programs with graphs[J]. arXiv:.00740,2017.
[34]Tufano M, Watson C, Bavota G et al., Deep learning similarities from different representations of source code[C]//2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), IEEE, 2018: 542-553.
[35]Myers E M, A precise inter-procedural data flow algorithm[C]//Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1981: 219-230.

附中文参考文献：

[29] 闫鑫;周宇;黄志球;.基于序列到序列模型的代码片段推荐[J].计算机科学与探索,2020,05:731-739.

SHI Zhicheng was born in 1995. He is an M.S. candidate at College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. His research interests include software evolution analysis and mining software repositories.

史志成（1995-），男，南京航空航天大学计算机科学与技术学院硕士研究生，主要研究领域为软件演化分析，软件库挖掘。

ZHOU Yu was born in 1981. He received the Ph.D. degree from Nanjing University in 2009. Now he is a professor at College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, and the senior member of CCF. His research interests include software evolution analysis, mining software repositories, software architecture and software reliability analysis.

周宇（1981-），男，2009 年于南京大学获得博士学位，现为南京航空航天大学计算机科学与技术学院教授，CCF 高级会员，主要研究领域为软件演化分析，软件库挖掘，软件架构，软件可靠性分析。