计算机科学
COMPUTER SCIENCE

一种基于神经网络的代码嵌入方法

孙雪凯, 蒋烈辉

引用本文

孙雪凯, 蒋烈辉. 一种基于神经网络的代码嵌入方法[J]. 计算机科学, 2023, 50(5): 64-71.

SUN Xuekai, JIANG Liehui. Code Embedding Method Based on Neural Network[J]. Computer Science, 2023, 50(5): 64-71.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220100094

一种基于神经网络的代码嵌入方法

孙雪凯　蒋烈辉
信息工程大学数学工程与先进计算国家重点实验室　郑州 450001
(sunxuekai1226@163.com)

摘　要　对代码进行分析研究具有很多的应用场景,例如代码抄袭检测、软件漏洞搜索等。随着人工智能的发展,神经网络技术被广泛应用于代码分析和研究。然而,现有的方法要么简单地将代码视为普通的自然语言处理,要么使用太过复杂的规则对代码进行采样,前者的处理方式容易造成代码关键信息的丢失,而后者会造成算法过于复杂,模型的训练需要花费较长的时间。Alon 等提出了一种名为 Code2vec 的算法,该算法采用了一种简单且有效的代码表示方法,相比之前的代码分析方法有着显著的优势,但 Code2vec 算法仍存在一些局限性。因此,在其基础上提出了一种基于神经网络的代码嵌入方法,该方法的主要思想是将代码函数表示为代码的嵌入向量。首先将一个代码函数分解为一系列抽象语法树路径,然后通过神经网络去学习如何表示每一条路径,最后将所有路径聚合成一个嵌入向量来表示当前的代码函数。文中实现了一个基于该方法的原型系统,实验结果表明,相比 Code2vec,所提算法的结构更加简单、训练速度更快。

关键词:神经网络;代码嵌入;代码分析;抽象语法树;代码分类
中图法分类号　TP311

Code Embedding Method Based on Neural Network

SUN Xuekai and JIANG Liehui
State Key Laboratory of Mathematical Engineering andAdvanced Computing,PLA Information Engineering University,Zhengzhou 450001,China

Abstract　There are many application scenarios for code analysis and research,such as code plagiarism detection and software vulnerability search. With the development of artificial intelligence,neural network technology has been widely used in code analysis and research. However,the existing methods either simply treat the code as ordinary natural language processing,or use much more complex rules to sample the code. The former processing method is easy to cause the loss of key information of the code,while the latter can make the algorithm to be too complicated,and the training of the model will take a lot of time. Alon proposed an algorithm named Code2vec,which has significant advantages compared with previous code analysis methods. But the Code2vec still has some limitations. Therefore,a code embedding method based on neural network is proposed. The main idea of this method is to express the code function as the code embedding vector. First,a code function is decomposed into a series of abstract syntax tree paths,then a neural network is used to learn how to represent each path,and finally all paths are aggregated into an embedding vector to represent the current code function. A prototype system based on this method is implemented in this paper. Experimental results show that compared with Code2vec,the new algorithm has the advantages of simpler structure and faster training speed.

Keywords　Neural network,Code embedding,Code analysis,Abstract syntax tree,Code classification

1　引言

随着互联网的不断发展,软件开发人员可以快速地在代码库中搜索到实现相关功能的源代码,通过复用已有代码,可以大大加快项目的开发进度。但这也导致了一些问题,如代码复用造成的版权纠纷问题以及软件漏洞造成的信息安全问题。因此,如何高效地对代码进行分析以满足不同的场景需要,已经成为当下的一个研究热点。

对代码的分析检测方法大致可以划分为 5 类[1]。

(1)基于文本的检测方法

基于文本的方法是最早的代码检测方法。首先预处理代码段,然后将代码段转换成字符,如果两个代码段的字符相同,则两段代码相同。该方法的优点在于实现过程简单;缺点是不能识别代码的语法、语义等信息。

(2)基于词法的检测方法

基于词法的检测方法,也被称为基于 Token 的检测方法。该方法借鉴了自然语言的处理方式,首先将代码段解析成一个字符串序列,然后分析不同代码段中的词法序列,如果存在相同的子序列,则说明存在代码克隆。此类方法的优点在于能使用轻量级工具,且相比基于语法、语义等检测算法

到稿日期:2022-01-11　返修日期:2022-09-19
通信作者:蒋烈辉(jiangliehui@163.com)

更加简单；缺点在于不能识别代码的语法语义等逻辑信息。

(3)基于语法的检测方法

基于语法的检测方法，也称为基于树的检测方法。首先通过对代码进行词法和语法分析，来构建源代码的抽象语法树(Abstract Syntax Tree，AST)，接着对抽象语法树进行分析比较，来确定代码是否存在克隆。此类方法的优点在于可以识别程序的语法信息，提高检测准确率；缺点是构造语法树以及匹配语法子树的算法代价很大，随着代码规模的扩大，最终检测方法的时空复杂度会很高。

(4)基于语义的检测方法

基于语义的检测方法，也称为基于图的检测方法。首先通过代码的语法结构和上下文环境等来构建代码的程序依赖图，然后通过代码切片和匹配算法得到相同或相似的子图，来确定是否有代码克隆。此方法的优点是可以识别程序的语义逻辑信息；缺点是构造程序依赖图的代价较大，且随着程序规模的扩大，方法的时空复杂度也会不断增加。

(5)基于度量值的检测方法

基于度量值的检测方法主要应用于对代码进行固定粒度检测的场景。首先将代码分成固定粒度的比较代码单元，然后从比较单元中抽取度量值，进而确定是否进行了代码克隆操作。该方法的优点是检测准确率高，便于代码重构；缺点是粒度不好把握，如果粒度太大，漏检率会很高。

上述 5 类检测方法中，前 2 种方法的实现较为简单，但是不能识别代码的语法、语义等信息；后 3 种方法的检测准确率较高，但也有各自的缺点。

随着机器学习和深度学习的迅速发展，许多学者致力于将这些技术应用于代码分析研究[2]。Alon 等[3]提出了一种名为 Code2vec 的方法，通过将抽象语法树转换为一个路径集合来表示代码，再使用神经网络学习每条路径的表示。该方法借助抽象语法树来提取代码的有效信息，提出了一个新的网络结构，可以进行代码嵌入，为代码学习到连续分布的向量表示。Code2vec 方法的代码表示规则更加简单，从抽象语法树到模型输入所需的步骤更少，算法速度更快；且根据文献[3]的表述，Code2vec 在嵌入的过程中能捕捉到名字之间的语义信息，方法名之间可以进行类比和推断，其实验结果也验证了模型的有效性，证明 Code2vec 相比其他的一些方法，在相同的数据集上性能提高了 75%。但 Code2vec 方法有两个局限性：1)针对中间的路径表示，算法仅对组成路径的字母进行一个哈希计算的叠加，并以此作为当前路径的标识，这会导致两条基本相同的路径得到的标识完全不同，不但丢失了关键信息，也大大增加了训练模型的时间花销；2)聚合路径时需要额外训练一个注意力参数，增加了模型复杂度。

本文构建了一种基于神经网络的代码嵌入方法(Code Embedding Method Based on Neural Network，CEMNN)。借鉴 Code2vec 将代码 AST 处理为路径集合的思想，模型也采取了将 AST 作为代码中间表示的方法，并通过采样路径的方式提取代码的有效信息，生成代码函数嵌入。为了验证 CEMNN 模型的有效性，还设计了一个代码分类任务作为模型的应用场景。实验结果表明，相比 Code2vec，CEMNN 模型的结构更加简单、训练速度更快。

2 相关工作

如何对代码进行处理与分析一直是软件工程领域的一个重要问题[4]。

早期的代码分析方法主要是利用文本或词法来处理代码。Kamiya 等[5]提出了一种基于词法的代码克隆检测方法，对代码进行规范化处理，将其转为标识符序列，然后输入到模型中。SourcererCC[6]在此基础上增强了对代码语义的提取，使用一种反向索引的方法对标识符序列的信息进行进一步挖掘。Jiang 等[7]提出的 Deckard 模型增强了对代码结构信息的挖掘，在模型的输入中加入了代码的语法结构信息。

一些学者借助抽象语法树来增强对代码信息的提取。Zhou 等[8]提出了一种名为 ContextCC 的方法，借助抽象语法树来扩充方法体内部元素的情景信息，以增强代码段的语义，提取代码的上下文信息来为代码段生成简明评论。Yan 等[9]将类似方法应用在代码推荐的场景中。除了借助抽象语法树扩充代码元素语义之外，更多学者选择对抽象语法树进行切分，得到子结构序列然后将其输入到模型中。Hu 等[10]提出了一种基于序列的模型，通过添加括号的方法来限定抽象语法树中节点的作用域，将抽象语法树转化为一个节点序列来生成代码注释。White 等[11]提出了一种基于深度学习的代码分析方法，借助代码的标识符序列以及抽象语法树节点序列，得到代码的语义向量和结构向量，并根据这些向量对代码克隆检测进行分析研究。

然而，这些方法将树形结构转化为序列结构，会不同程度地破坏元素之间的一些依赖关系，因此，有学者提出不降维而直接处理抽象语法树的树形深度学习模型。Wan 等[12]将代码分为两部分作为模型输入，使用 RNN 对代码的标识符序列进行编码得到语义向量，而在处理抽象语法树时，Wan 使用 Tree-LSTM 模型来得到代码的结构向量，而不是像 White 等那样直接将抽象语法树转化为节点序列。Mou 等[13]提出了一种基于树的卷积神经网络模型，该模型直接在抽象语法树上进行卷积运算，然后使用动态池化技术将不同规格的抽象语法树压缩成代码向量，以提取代码中的结构信息。

代码片段的控制结构也可以抽象成图来表示，将代码用图表示之后，可使用图嵌入技术对代码进行分析检测。Allamanis 等[14]提出了一种基于注意力机制的卷积神经网络方法，该方法通过相同的函数名以及变量名来静态地建立这些函数名以及变量之间存在的依赖关系。Iyer 等[15]提出了一种完全数据驱动的方法，来为源程序产生高质量的代码摘要，该方法相比 Allamanis 提出的卷积神经网络方法，训练速度有所加快，但精确率和召回率均有所降低。Gemini 模型[16]提出了一种基于图嵌入的相似性解决方案，利用统计方法得到基本块的向量表示，将二进制代码函数的控制流程图表示为具有向量值的属性控制流图，然后利用 Structure2vec 实现图嵌入，接着将两个图嵌入向量送入孪生神经网络，从而实现相似性比较。

也有学者提出基于度量值的代码检测思路，从代码中抽取一系列度量值来进行代码抄袭判定[17]。Donaldson 等[18]最早提出利用属性计数的基本方法来分析代码抄袭，充分

考虑程序的结构信息。ENGELS[19]将比较代码分为相似类和非相似类，提取代码对中的12个度量值作为特征向量，输入到神经网络中进行训练，反复调节学习参数来减小训练误差，从而使模型具有相似检测能力。

对比这些工作，本文的工作主要集中在借助抽象语法树对代码的信息进行提取，并利用神经网络模型来生成代码函数的嵌入。实验结果证明，所提模型的结构简单、训练速度快。

3 CEMNN模型介绍

3.1 代码表示

本节主要介绍CEMNN模型的采样方法，即如何表示代码函数。给定一段代码片段，首先需要将其转换为AST，然后根据AST生成不同的路径，这些路径将作为神经网络模型的输入。

首先对AST和对应的路径上下文做出定义说明。

(1)抽象语法树(AST)。给定代码片段C，其抽象语法树可以用一组元组$\langle N,T,X,s,\delta,\varepsilon\rangle$表示，其中$N$表示非终端节点，$T$表示终端节点，$X$表示一组值，$s$为根节点，$s\in N$，$\delta$表示非终端节点到它的孩子的映射函数，$\delta:N\rightarrow(N\cup T)$，$\varepsilon$为终端节点到其对应值的映射，$\varepsilon:T\rightarrow X$。

(2)抽象语法树路径上下文(AST path-context)。假定p为AST的一条路径，它同样可以被表示为一组元组$\langle t_s,t_e\rangle$，其中$t_s=\varepsilon(start(p))$，为当前路径$p$起始节点对应的值，$t_e=\varepsilon(end(p))$，为路径$p$末端节点对应的值。

如图1所示，给定代码函数，首先将其转化为AST，然后采样不同路径(图1中给出了4条路径)。路径1的上下文为起始节点elements和末端节点true所对应的值。

$$path\_context_1=\langle\varepsilon(elements),\varepsilon(true)\rangle \tag{1}$$

假定有代码函数C以及它的抽象语法树$\langle N,T,X,s,\delta,\varepsilon\rangle$，将AST所有的终端节点两两组队。

$$TPairs(C)=\{(term_i,term_j)\mid term_i,term_j\in termNodes(C)\wedge i\neq j\} \tag{2}$$

其中，$termNodes(C)$表示代码函数C所有终端节点的集合，$term$表示某一个终端节点，$TPairs(C)$表示函数C所有终端节点两两组队的集合。

代码函数C可以用抽象语法树的路径上下文来表示。

$$Repres(C)=\{(x_s,x_e)\mid\exists(term_s,term_e)\in TPairs(C):x_s=\varepsilon(term_s)\wedge x_e=\varepsilon(term_e)\} \tag{3}$$

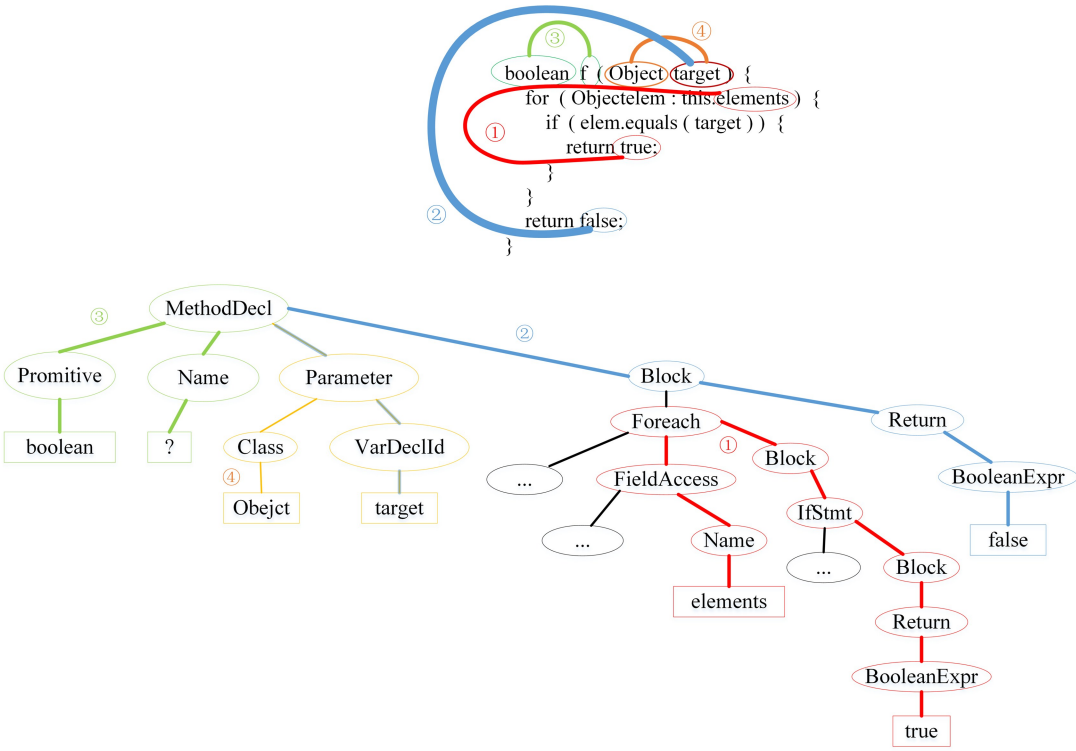

图1 代码函数表示

Fig. 1 Code function representation

3.2 模型设计

本节将详细介绍CEMNN模型的设计方法。3.1节介绍了路径上下文的概念，而一段代码片段就是由组成AST的这些路径上下文所表示的。每一个路径上下文所对应的值可组成一个向量，这个向量是模型的输入，同时也被模型训练学习。这些路径上下文向量中包含了当前路径的一些信息，最后会聚合成一个单独的代码嵌入。模型的总体架构如图2所示。

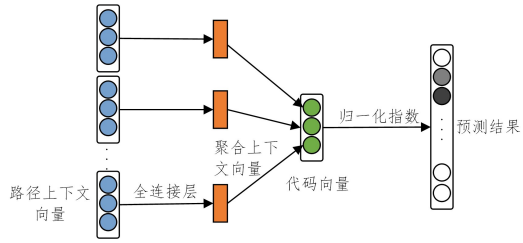

图2 模型总体架构

Fig. 2 Model overall architecture

CEMNN 模型需要训练以下几个部分：1）组成路径上下文的终端节点所对应的嵌入向量；2）全连接层的权重参数；3）函数所对应函数名的标签嵌入向量。

定义一个终端节点所对应的嵌入向量矩阵：

$$\boldsymbol{value\_vocab}\in\mathbb{R}^{|X|\times d} \tag{4}$$

其中，X 为数据集中所有终端节点的个数，d 为终端节点对应嵌入向量的维度。若要得到某一终端节点的嵌入，只需要在向量矩阵中查询矩阵的某一行即可。例如，在图 1 中，向量矩阵 ***value_vocab*** 每一行都表示一个终端节点的嵌入，如 elements，true，target，Object 等。矩阵中的值在模型训练开始前被随机初始化，然后通过神经网络训练学习。

对于全连接层的权重参数 $\boldsymbol{W}$，它的维度大小主要取决于训练时间、模型复杂度以及实验设备的性能。在实验中，为了方便，将其设置为 $2d$（与路径上下文的嵌入维度相同）。

将从代码片段中提取出的路径上下文 $\mathcal{B}=\{\beta_1,\cdots,\beta_n\}$ 作为神经网络模型的输入，其中 $\beta_i=\langle x_s,x_e\rangle$ 为其中一条路径上下文，组件 x 的对应嵌入可以通过 ***value_vocab*** 查询。一条路径上下文的嵌入由组成它的组件的嵌入组合表示：

$$\begin{aligned}c_i&=embedding(\langle x_s,x_e\rangle)\\&=[\boldsymbol{value\_vocab}_s;\boldsymbol{value\_vocab}_e]\in\mathbb{R}^{2d}\end{aligned} \tag{5}$$

当得到路径上下文的嵌入 c_i 后，将其作为模型的输入，经过一层全连接神经网络，得到输出 $\tilde{c}_i$：

$$\tilde{c}_i=\tanh(\boldsymbol{W}\cdot c_i) \tag{6}$$

其中，$\boldsymbol{W}\in\mathbb{R}^{2d\times 2d}$ 是全连接的权重参数矩阵，tanh 为常用的激活函数双曲正切函数，主要用于增加模型的表现力。下一步，需要将全连接层的输出 $\tilde{c}_i$ 聚合为一个单独的嵌入，这个嵌入表示整个代码函数：

$$code\ vector\ \boldsymbol{v}=\sum_{i=1}^{n}\tilde{c}_i \tag{7}$$

使用代码嵌入来预测标签，定义一个词汇标签矩阵：

$$\boldsymbol{tags\_vocab}\in\mathbb{R}^{|Y|\times d} \tag{8}$$

其中，Y 是训练集中所有训练标签的个数，tag_i 表示 ***tags_vocab*** 中第 i 行的元素，同时也是训练集中一个函数标签的嵌入。将代码嵌入 $\boldsymbol{v}$ 与每一个标签的嵌入进行运算后得到它的预测分布：

$$\text{for } y_i\in Y: q(y_i)=\frac{\exp(v^{\mathrm{T}}\cdot\boldsymbol{tags\_vocab}_i)}{\sum_{y_j\in Y}\exp(v^{\mathrm{T}}\cdot\boldsymbol{tags\_vocab}_j)} \tag{9}$$

模型采用的损失函数为交叉熵损失函数[20]。已知 q 为样本的预测分布，设 p 为样本的真实分布，则对应训练样本的真实标签时 p 的值应为 1，否则为 0，交叉熵损失函数的表达式如下：

$$\mathcal{H}(p\parallel q)=-\sum_{y\in Y}p(y)\log q(y)=-\log q(y_{\text{true}}) \tag{10}$$

其中，y_{true} 表示样本的真实标签，也就是说，损失是 $q(y_{\text{true}})$ 的负对数，$q(y_{\text{true}})$ 越倾向于 1，损失越接近 0。因此，最小化交叉熵损失就相当于最大化模型分配给真实标签 y_{true} 的可能性。

4 实验评估

CEMNN 模型能够将任意尺寸的代码片段转化为固定大小的代码嵌入，这些代码嵌入捕获了代码中的相关信息。本节将设计一个代码分类任务来评估本文模型。

评估的主要目标为：1）CEMNN 模型在代码分类任务上的表现；2）CEMNN 模型的表现对比 Code2vec 的优点和缺点；3）CEMNN 模型在聚合路径上下文时，如果加上注意力机制，则会对模型产生的影响。

4.1 准备工作

代码分类使用的数据集来源是 Alon 等[21]构建的 14M 样本集，从中随机选择了 224 类函数，每类包含 100 个样本，数据集共有 22400 条数据，按照比例随机分为训练集、验证集和测试集，详情如表 1 所列。

表 1 代码分类任务数据集

Table 1 Code classification task dataset

数据集	数量	数量（每类）
训练集	17920	80
验证集	2240	10
测试集	2240	10

实验使用的处理器为 Intel © Core™ i7-8700 CPU（6 个内核），内存为 16.0GB，显卡为英伟达 GTX 1660。代码分类模型使用 Adam 优化算法[22]，为了降低过拟合，在每轮训练时按照 0.25 的比例随机抛弃一些神经元[23]，所有可训练参数值的初始化使用 Bengio 等[24]提出的初始化方法。默认情况下，单个终端节点的维度设为 64，该参数即为 3.2 节中式(4)中的参数变量 d，由式(5)可知每个路径上下文的嵌入维度为 $2d=128$。维度参数设置得过小，会造成算法无法从代码中提取出足够的特征信息；设置得过大，会增加模型的训练时间，且在训练过程中可能会造成内存溢出。每个样本的路径上下文数量设为 200。图 1 中的样本列举出了 4 条不同的路径上下文，但在实际中每个代码函数会生成不同数量的路径上下文，虽然算法可以聚合任意数量的输入，但经过实验发现，从每个样本中随机采样 200 个路径上下文较为合适，增加该值并没有改善结果。模型训练 20 轮，每轮训练分批进行，每个批次由 512 个样本数据组成。每轮训练后，在验证集上测试损失及验证性能指标，以便对模型做一个初步的评估。在 20 轮训练结束后，保存在验证集上获得最佳性能的模型，以便进一步在测试集上评估模型的表现。

4.2 度量指标

在代码分类任务中，由以下几个广泛使用的度量指标来衡量模型性能。

(1) $top(k)$

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

其中，$|Q|$ 表示样本类别的数目，若前 k 个候选类别中有一个是当前样本的实际类别，则认为该次查询命中，$rel(k)$ 表示命中的查询样本的个数。

(2) $precision$

$$precision=\frac{TP}{TP+FP}$$

其中，$precision$ 表示精确率，TP 表示将正类预测为正类，FP 表示将负类预测为正类。在算法中，如果首个候选样本的类别就是当前样本的实际类别，则计入到 TP，否则计入到 FP。

(3)*recall*

$$recall=\frac{TP}{TP+FN}$$

其中，*recall* 表示召回率，*TP* 的定义同上，*FN* 表示将正类预测为负类。在算法中，如果样本的实际类别不在首个预测样本中，则计入到 *FN*。

(4)*f*1

$$f1=\frac{precision*recall*2}{precision+recall}$$

其中，*f*1 表示精确率 *precision* 和召回率 *recall* 的调和均值，这个指标同时考虑了精确率和召回率，当 *f*1 较高时，说明该方法比较有效。

4.3 实验结果

评估目标 1：CEMNN 模型在代码分类任务上的表现。

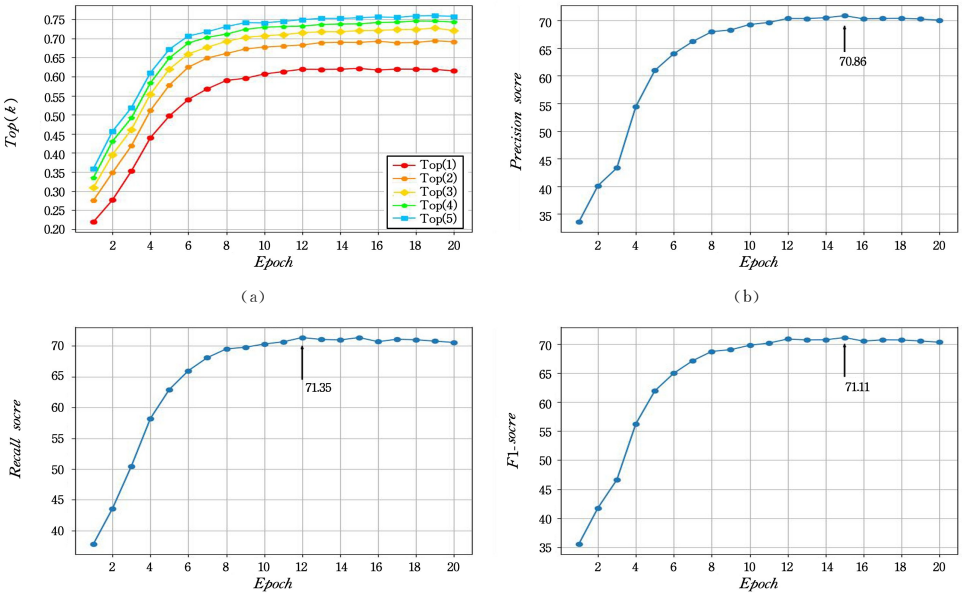

图 3 CEMNN 模型每训练一轮的验证结果

Fig. 3 Validation results of each training round of CEMNN

每轮训练后都会保存当前模型参数，经过 20 轮训练后，将表现最好的一组参数载入模型，并在测试集上测试该模型。模型在测试集上的表现如表 2 和表 3 所列。由表 2 可知，模型在测试集上的 *f*1 指标得分可以达到 71.43，由表 3 可知模型的 top(1)为 61.96%，top(5)为 76.16%。

表 2 CEMNN 模型在测试集上的表现结果

Table 2 Performance results of CEMNN on test set

指标	得分
精确率	71.17
召回率	71.69
f1	71.43

表 3 CEMNN 模型在测试集上的 top 指标

Table 3 Top indicators of CEMNN on test set

指标	结果/%
top(1)	61.96
top(2)	69.06
top(3)	72.41
top(4)	74.73
top(5)	76.16

评估目标 2：CEMNN 模型的表现对比 Code2vec 的优点和缺点。

图 3 给出了每经过一轮训练后，CEMNN 模型在验证集上不同度量指标的分布结果。由图 3 可知，大约经过 12 轮训练后，模型趋于收敛。

图 3(a)给出了模型 top(1)－top(5)的分布，其中当 *epoch*=15 时，top(1)，top(3)和 top(5)分别为 62.14%，72.01%和 75.40%。也就是说，模型经过 15 轮训练后，对验证集样本进行预测，排名最高的预测类别是样本真实类别的概率为62.14%，排名前 3 的预测类别中存在样本的真实类别的概率为 72.01%，排名前 5 的预测类别中存在样本的真实类别的概率为 75.40%。图 3(b)－图 3(d)分别给出了 CEMNN 模型在验证集上的精确率、召回率和 f1 调和均值的得分。由图 3 可知，模型在验证集上的 3 项指标得分会随着每轮训练而增加，直到第 12 轮训练后模型趋于收敛，经过 15 轮训练后，3 项指标的得分分别为 70.86，71.35 和 71.11。

本文分析了 Code2vec 模型的一些不足，且提出了 CEMNN 模型并改进了这些不足，因此将 Code2vec 模型作为对比模型。在实验过程中，两者使用相同的原始数据集，实验设备与参数配置也完全相同，终端节点的维度设为 64，每个样本的路径上下文数量设为 200，训练 20 轮，每轮训练分批进行，每个批次由 512 个样本数据组成。

图 4 分别给出了 CEMNN 与 Code2vec 模型在验证集上的精确率、召回率、f1 以及在训练集上训练累计时间的对比结果。由图 4(a)－图 4(c)可知，CEMNN 模型的精确率、召回率、f1 表现略低于 Code2vec 模型。其中 CEMNN 模型的精确率、召回率和 f1 的最高得分分别为 70.86，71.35 和 71.11，Code2vec 模型对应指标的最高得分分别为 72.63，73.01 和 72.79，CEMNN 模型比 Code2vec 的 3 项指标分别低了 2.4%，2.3%和 2.3%。图 4(d)给出了两种模型在训练集上训练的累计时间分布。由图 4 可知，CEMNN 模型的训练时间明显短于 Code2vec。训练 20 轮后，CEMNN 模型的累计训练时间相比 Code2vec 缩短了 59.9%。

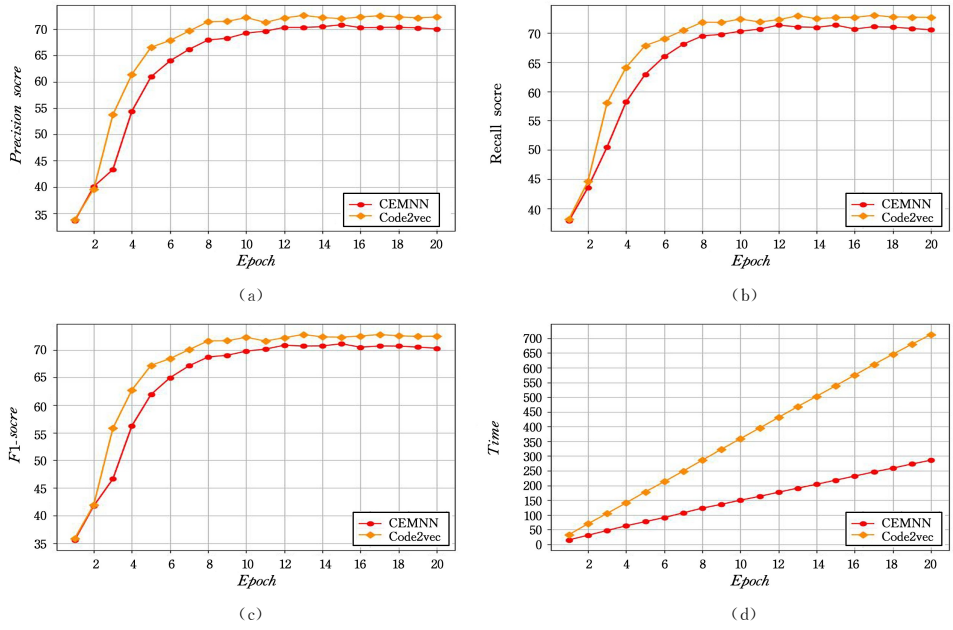

图 4 CEMNN 和 Code2vec 模型的对比结果

Fig. 4 Comparison results of CEMNN and Code2vec

图 5 分别给出了 CEMNN 与 Code2vec 模型在验证集上 top(1) 和 top(5) 的分布。由图 5 可知，CEMNN 模型的 top(1)表现略低于 Code2vec，其中 CEMNN 模型的 top1 最高为 62.14%，Code2vec 对应项的最高值为 64.15%；但 CEMNN 模型的 top5 表现最终要高于 Code2vec。由图 5 可知，在 14 轮训练之后，CEMNN 模型的表现超过了 Code2vec，CEMNN 模型在 19 轮训练时 top(5)=75.94%，而 Code2vec 模型在 19 轮训练时 top(5)=74.59%。

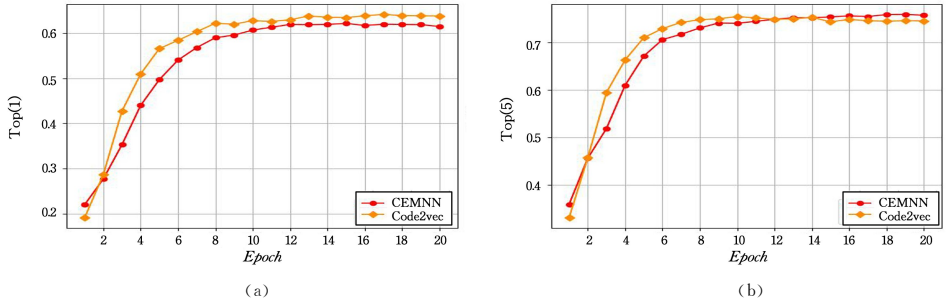

图 5 CEMNN 和 Code2vec 模型的 top 对比

Fig. 5 top comparison of CEMNN and Code2vec

综上所述，CEMNN 模型的缺点为其精确率、召回率和 f1 指标相比 Code2vec 分别低了 2.4%，2.3%和 2.3%，且top(1)的表现略低于 Code2vec；优点为其 top(5)的表现高于 Code2vec，且模型的训练时间比 Code2vec 模型缩短了约 59.9%。

评估目标 3：CEMNN 模型在聚合路径上下文时，如果加上注意力机制，则会对模型产生的影响。

Code2vec 在模型中引入了一种注意力机制，用于聚合路径，该机制的引入提高了算法的复杂度，延长了模型的训练时间。为了评估该机制会对 CEMNN 模型产生的影响，这里采用与 Code2vec 相同的注意力机制来聚合路径上下文。

3.2 节介绍了 CEMNN 模型聚合路径上下文的方法。在得到代码的路径上下文嵌入$\tilde{c}_i$后，引入一个注意力向量$\boldsymbol{\alpha}\in\mathbb{R}^{2d}$，该向量在训练前会先被随机初始化。给定一组路径上下文的嵌入$\{\tilde{c}_1,\cdots,\tilde{c}_n\}$，每一个路径上下文嵌入$\tilde{c}_i$对应一个注意力权重$\alpha_i$，权重$\alpha_i$的计算式如下：

$$attention\ weight\ \alpha_i=\frac{\exp(\tilde{c}_i^{\mathrm{T}}\times\boldsymbol{\alpha})}{\sum_{j=1}^{n}\exp(\tilde{c}_i^{\mathrm{T}}\times\boldsymbol{\alpha})} \tag{11}$$

经过加权计算可得到代码函数的嵌入：

$$code\ vector\ \boldsymbol{v}=\sum_{i=1}^{n}\alpha_i\tilde{c}_i \tag{12}$$

采用基于注意力机制的方法来聚合函数的代码嵌入，由于训练时参数 d 设为 64，因此新增参数$\alpha\in\mathbb{R}^{2d}$的维度为 128，在硬件配置和参数配置保持不变的情况下用相同的训练集训练模型(Attention-CEMNN)。

图 6 给出了加入注意力机制前后模型在验证集上的精确率、召回率、f1 以及在训练集上训练累计时间的对比结果。引入注意力机制后，每个路径上下文都对应一个权重，模型在训练过程中会自适应地给这些权重分配更加合适的值，因此算法会拟合得更快，模型的收敛速度也会加快。由图 6(a)—图 6(c)可知，加入注意力机制后模型的收敛速度会略微加快，但收敛后的模型性能并无明显变化。图 6(d)给出了算法在训练集上训练时间的累计结果对比，加入注意力机制后模型的训练时间延长了 28.7%。这是由于引入注意力

机制后，模型需要额外训练一个注意力参数，算法复杂度增加，因此训练时间也延长了。

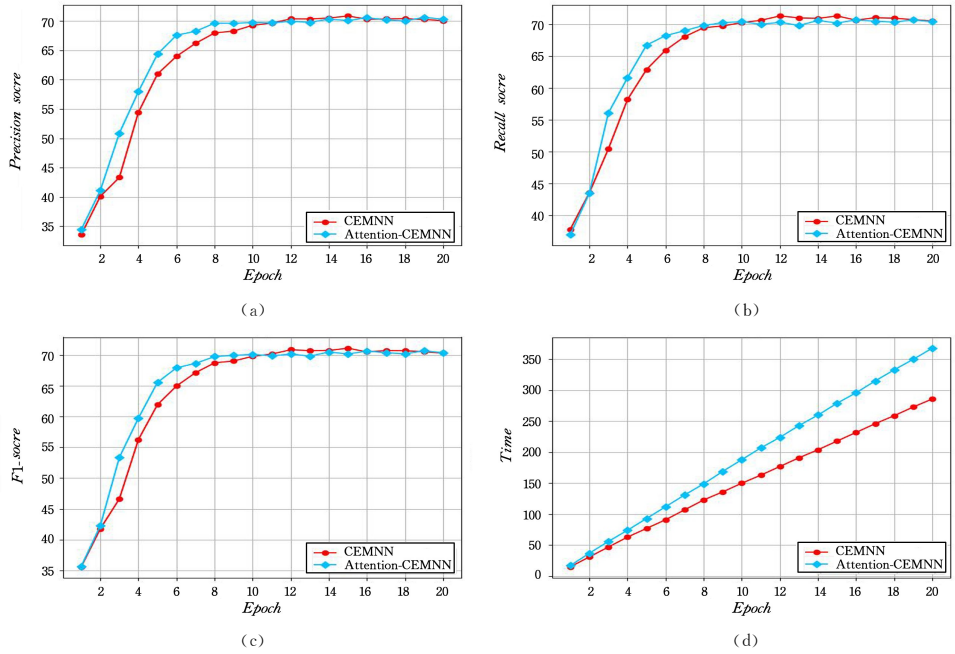

图 6 加入注意力机制前后模型的对比结果

Fig. 6 Comparison results before and after adding attention mechanism

图 7 分别给出了加入注意力机制前后模型在验证集上 top(1)和 top(5)的分布。

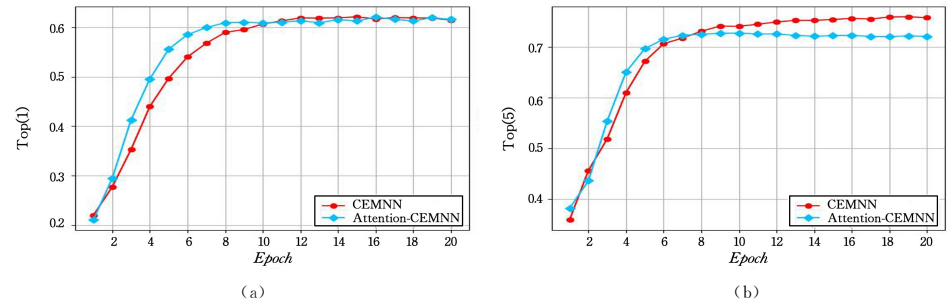

图 7 加入注意力机制前后模型的 top 对比

Fig. 7 top comparison before and after adding attention mechanism

由图 7 可知，加入注意力机制后模型 top(1)的最终表现无明显变化，但 top(5)的最终表现反而会变差，模型在第 19 轮训练后 top(5)达到了 75.94%，但是加入注意力机制后 top(5)降为了 72.19%。这说明引入注意力机制后，对模型的性能反而产生了消极影响，因此其 top(k)指标会随着 k 值的增加而下降。

综上所述，引入注意力机制后，算法拟合得更快，因此收敛速度会加快。但该方法并没有真正改善模型的算法机制，因此模型的最终性能并没有提高，且由于需要额外学习一个权重参数，因此模型的训练时间也延长了。

结束语 本文提出了一个基于神经网络的代码嵌入模型 CEMNN，并设计了一个代码分类任务对模型进行了评估。实验结果表明，与 Code2vec 相比，CEMNN 模型的结构更加简单、训练时间更短。

虽然实验结果证明 CEMNN 模型相比 Code2vec 具有一些优势，但该模型仍存在一些有待提高的地方。例如，对数据集中代码的书写质量要求较高，不规范的代码书写会影响模型的性能；当前的代码表示方法在采样时仍会造成一些代码信息的损失。未来会对模型进行进一步研究，寻找一个对代码进行预处理的方法，以降低不规范的代码书写对模型造成的影响；并寻求一种更加完善有效的方法来表示代码上下文中的路径，以减小代码信息的损失。

参 考 文 献

[1] ZHANG D, LUO P. Survey of code similarity detection methods and tools[J]. Computer Science, 2020, 47(3): 5-10.

[2] CHEN Q Y, LI S P, YAN M, et al. Code clone detection: A literature review[J]. Journal of Software, 2019, 30(4): 962-980.

[3] ALON U, ZILBERSTEIN M, LEVY O, et al. Code2vec: learning distributed representations of code[J]. Proceedings of the Programming Languages, 2019, 3(POPL): 1-29.

[4] SHI Z C, ZHOU Y. Method of Code Features Automated Extraction[J]. Journal of Frontiers of Computer Science and Technology, 2021, 15(3): 456-467.

[5] 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.

[6] 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.

[7] JIANG L，MISHERGHI G，SU Z，et al. DECKARD：scalable and accurate tree-based detection of code clones[C]// International Conference on Software Engineering. IEEE，2006.

[8] ZHOU Y，YAN X，YANG W，et al. Augmenting Java method comments generation with context information based on neural networks[J]. The Journal of Systems and Software，2019，156(Oct.)：328-340.

[9] YAN X，ZHOU Y，HUANG Z Q. Code snippets recommendation based on sequence to sequence model[J]. Journal of Frontiers of Computer Science and Technology，2020，14(5)：731-739.

[10] HU X，LI G，XIA X，et al. Deep code comment generation [C]// Proceedings of the 26th Conference on Program Comprehension. 2018：200-210.

[11] WHITE M，TUFANO M，VENDOME C，et al. Deep learning code fragments for code clone detection[C]// Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016：87-98.

[12] WAN Y，ZHAO Z，YANG M，et al. Improving automatic source code summarization via deep reinforcement learning [C]// Proceedings of the 33rd IEEE/ ACM International Conference on Automated Software Engineering. 2018：397-407.

[13] MOU L L，LI G，ZHANG L，et al. Convolutional neural networks over tree structures for programming language processing [C]// Proceedings of the 30th AAAI Conference on Artificial Intelligence. 2016：1287-1293.

[14] ALLAMANIS M，PENG H，SUTTON C. Aconvolutional attention network for extreme summarization of source code [C]// Proceedings of the 33nd International Conference on Machine Learning. 2016：2091-2100.

[15] IYER S，KONSTAS I，CHEUNG A，et al. Summarizing source code using a neural attention model [C]// Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. 2016.

[16] XU X J，LIU C，QIAN F，et al. Neural network-based graph embedding for cross-platform binary code similarity detection [C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017：363-376.

[17] XIONG H，YAN H H，GUO T，et al. Code similarity detection：a survey[J]. Computer Science，2010，37(8)：9-14，76.

[18] DONALDSON J L，LANCASTER A M，SPOSATO P H. A plagiarism detection system[C]// ACM SIGCSE Bulletin. 1981：21-25.

[19] ENGELS S，LAKSHMANAN V，CRAIG M. Plagiarism detection using feature-based neural networks[C]// ACM SIGCSE Bulletin. 2007：34-38.

[20] RUBINSTEIN R. The cross-entropy method for combinatorial and continuous optimization[J]. Methodology and Computing in Applied Probability，1999，1(2)：127-190.

[21] ALON U，ZILBERSTEIN M，LEVY O，et al. A general path-based representation forpredicting program properties [C]// Proceedings of the 39th ACM SIGPLAN Conference. ACM，2018.

[22] KINGMA D P，BA J. Adam：a method for stochastic optimization[J]. arXiv：1412.6980，2017.

[23] SRIVASTAVA N，HINTON G，KRIZHEVSKY A，et al. Dropout：a simple way to prevent neural networks from overfitting[J]. Journal of Machine Learning Research，2014，15(1)：1929-1958.

[24] BENGIO Y，GLOROT X. Understanding the difficulty of training deep feed forward neural networks[C]// Proceedings of the 13th International Conference on Artificial Intelligence and Statistics. 2010：249-256.

SUN Xuekai，born in 1991，Ph.D candidate. His main research interests include code similarity detection and code vulnerability mining.

JIANG Liehui，born in 1967，Ph.D，professor. His main research interests include computer architecture，reverse engineering，and cyberspace security.

（责任编辑：喻藜）