计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200900185

基于神经网络的二进制函数相似性检测技术

方 磊[1] 魏 强[1] 武泽慧[1] 杜 江[1] 张兴明[2]
1 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
2 之江实验室 杭州 310001
(nanbeiyouzi@live.com)

摘 要 二进制代码相似性检测在程序的追踪溯源和安全审计中都有着广泛而重要的应用。近年来,神经网络技术被应用于二进制代码相似性检测,突破了传统检测技术在大规模检测任务中遇到的性能瓶颈,因此基于神经网络嵌入的代码相似性检测技术逐渐成为热门研究。文中提出了一种基于神经网络的二进制函数相似性检测技术,该技术首先利用统一的中间表示来消除不同汇编代码在指令架构上的差异;其次在程序基本块级别,利用自然语言处理的词嵌入模型来学习中间表示代码,以获得基本块语义嵌入;然后在函数级别,利用改进的图神经网络模型来学习函数的控制流信息,同时兼顾基本块的语义,获得最终的函数嵌入;最后通过计算两个函数嵌入向量间的余弦距离来度量函数间的相似性。文中实现了一个基于该技术的原型系统,实验表明该技术的程序代码表征学习过程能够避免人为偏见的引入,改进的图神经网络更适合学习函数的控制流信息,系统的可扩展性和检测的准确率较现有方案都得到了提升。

关键词:二进制函数;相似性检测;表征学习;图神经网络
中图法分类号 TP313

Neural Network-based Binary Function Similarity Detection

FANG Lei[1],WEI Qiang[1],WU Ze-hui[1],DU Jiang[1] and ZHANG Xing-ming[2]
1 State Key Laboratory of Mathematical Engineering and Advanced Computing,PLA Information Engineering University,Zhengzhou 450001,China
2 Zhejiang Lab,Hangzhou 310001,China

Abstract Binary code similarity detection has extensive and important applications in program traceability and security audit. In recent years,the application of neural network technology in binary code similarity detection has broken through the performance bottleneck encountered by traditional detection techno-logy in large-scale detection tasks,making code similarity detection technology based on neural network embedding gradually become a research hotspot. This paper proposes a neural network-based binary function similarity detection technology. This paper first uses a uniform intermediate representation to eliminate the differences in instruction architecture of assembly code. Secondly,at the basic block level,it uses a word embedding model in natural language processing to learn the intermediate representation code and obtain the basic block semantic embedding. Then,at the function level,it uses an improved graph neural network model to learn the control flow information of the function,taking consideration of the basic block semantics at the same time,and to obtain the final function embedding. Finally,the similarity between two functions is measured by calculating the cosine distance between the two function embeddingvectors. This paper also implements a prototype system based on this technology. Experiments show that the program code representation learning process of this technology can avoid the introduction of human bias,the improved graph neural network is more suitable for learning the control flow information of functions,and the scalability and detection accuracy of our system are both improved,compared with the existing schemes.

Keywords Binary function,Similarity detection,Representational learning,Graph neural network

1 引言

目前学术界对二进制代码相似性还没有标准的定义。通常认为,由同一或相似的源代码编译得到的二进制代码是相似的。二进制代码相似性检测技术常被应用于代码复用和脆弱性检测。在现实情况中,出于对软件知识产权保护的考虑

到稿日期:2020-09-25 返修日期:2020-12-14 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2016QY07X1404,2017YFB0802901);之江实验室"先进工业互联网安全平台"项目(2018FD0ZX01)
This work was supported by the National Key Research and Development of China (2016QY07X1404,2017YFB0802901) and Advanced Industrial Internet Security Platform Project(2018FD0ZX01).
通信作者:魏强(prof_weiqiang@163.com)

和其他原因，安全人员常常无法获得程序完整的源代码。可执行程序或组件是由源代码编译而来的，那么对源代码的复用和修改在对应的二进制代码中也能得到一定的体现。因此，针对二进制代码的相似性检测技术具有更强的通用性，逐渐成为了研究热点之一。

同一源代码经不同的编译器，应用不同的编译优化配置，针对不同的指令架构（Instruction Set Architecture，ISA），编译得到的二进制代码不尽相同。因此，二进制代码相似性检测会遇到跨编译器、跨编译优化配置和跨指令架构检测等难题。传统的二进制代码相似性检测技术的主要思路是，首先为二进制代码片段抽象出与编译器、编译优化配置和指令架构无关的中间表示，如标识符序列、抽象语法树和控制流图（Control Flow Graph，CFG）[1]，然后通过分析和比较中间表示的异同来度量代码间的相似性。特别地，CFG 作为程序代码的一种高度抽象表示，具有跨语言的特性，因此无论是源代码还是二进制代码的相似性检测技术大多依赖 CFG 作为中间表示。但是，由于图匹配算法的时间复杂度缺少多项式解，且多是两两匹配算法，在处理大规模任务时，计算量会随代码库规模的增加呈几何倍数式增长[2]。因此，在很长一段时间里，这类技术的研究主要围绕优化 CFG 结构和图匹配算法展开。

直到最近几年，神经网络技术为该领域开启了新的研究方向，即利用神经网络为二进制代码生成嵌入向量，用嵌入向量间的距离来度量代码间的相似度。这类方法虽然突破了传统技术遇到的性能瓶颈，但最新的研究成果仍然存在以下 3 点不足。

（1）检测方法的可扩展性仍需加强。虽然现有的许多方法都具有跨指令架构的特性，但在实际应用中，一方面，需要针对每一种架构做单独的分析和处理；另一方面，神经网络的训练样本通常是由不同架构的组合所构造的一对数据，随着系统支持架构种类的增加，适配的工作量也会剧增，这在一定程度上会限制方法的可扩展性。

（2）针对代码高级中间表示的分析和比较难度大、效率低。传统的 CFG 图匹配算法存在开销大的问题，在大规模检测任务中尤为突出，而简化 CFG 节点、优化图结构、提升算法效率等方法对系统整体效率的提升不仅十分有限，而且会提高检测的误报率和漏报率，降低准确度。

（3）相似度模型的构建容易引入人为偏见。现有研究多采用人工定义和筛选的代码特征作为相似特征，尤其是基于代码统计特征构建特征向量的方法，特征向量中的每一维度代表一种代码属性，这类方法默认不同属性之间互不相关且对相似性的影响大小相同，然而实际情况往往并非如此，会引入人为偏见。

基于相似函数的二进制代码具有相似的语义和相似的执行逻辑这一假设，本文提出了一种新的基于神经网络的二进制函数相似性检测技术。该技术的主要原理是：首先将二进制函数转换为 CFG 中间表示，并将 CFG 的节点，即程序基本块字节码，翻译为 VEX[3] 中间表示（VEX 是动态分析框架 Valgrind 的中间表示，具有二地址的形式，支持多种指令架构[4-5]），以消除二进制代码在指令架构上的差异；其次，用指令序列的语义相似性建模基本块相似性，即利用自然语言处理（Natural Language Processing，NLP）的 PV-DM[6]（Distributed Memory Model of Paragraph Vectors）模型来学习基本块的 VEX IR（Intermediate Representation）代码，以获得基本块语义嵌入；然后，用基本块语义和 CFG 结构相似性建模函数相似性，即利用改进的 Structure2vec[7] 图神经网络（Graph Neural Network，GNN）模型来学习函数 CFG，以获得函数嵌入；最后，用函数嵌入向量间的余弦距离来表示函数间的相似度。该技术可以应用于基于函数相似度的下游任务，如函数级的代码复用和脆弱性检测任务。

与以往方法不同，本文受动态分析技术利用中间表示来跨架构地分析二进制代码的启发，提出利用 VEX IR 来消除二进制代码在指令架构上的差异，从而轻松实现对任意指令架构的支持和扩展，弥补了不足（1）；利用两种不同的神经网络模型，分别学习程序基本块语义和函数控制流信息，弥补了不足（2），且函数的相似性特征完全由神经网络学习获得，避免使用人工定义的特征，弥补了不足（3）；利用消息传递网络（Message Passing Neural Network，MPNN）[8] 的概念改进了 Structure2vec 模型，较现有方法[9] 更适合函数 CFG 嵌入任务。

本文第 2 节介绍了基于神经网络嵌入的二进制代码相似性检测的研究现状；第 3 节对本文所研究的问题进行了定义和规范；第 4 节介绍了本文提出的基于神经网络的二进制函数相似性检测技术的原理和所用模型的结构；第 5 节围绕该技术的创新点对原型系统的各项指标进行了评估；最后总结全文。

2　相关研究

Xu 等[9] 于 2017 年提出了第一个基于图神经网络的跨架构二进制代码相似性检测技术，并实现了一个名为 Gemini 的原型系统。其提出了 ACFG（Attributed CFG）的概念，即将 CFG 的节点用特征向量来替代（特征向量是由 6～8 种与指令架构无关的代码统计特征所构成，每种特征对应向量中的一个维度）。其利用改进的 Structure2vec 模型将函数 ACFG 嵌入高维向量，并利用孪生架构来加快参数的训练过程，最后采用函数嵌入向量间的余弦距离来度量函数间的相似性。

Liu 等[10] 在 αDiff 检测系统中，利用二维卷积神经网络（Convolutional Neural Network，CNN）直接对程序的二进制字节流进行嵌入，在最终的代码相似度中还考虑了函数的出入度和调用表的相似度。但该方法无法解释为什么常用于图像数据处理的 CNN 模型，在处理二进制程序时也能取得不错的效果，尽管二进制程序的字节流明显有别于图像数据。

Ding 等[11] 提出的 Asm2vec 是一种基于 PV-DM 模型（PV-DM 属于自然语言表征模型，是为文本数据而设计的）所改进的汇编代码表征学习模型，它利用自定义的函数内联规则和随机漫步机制将函数的 CFG 转换为汇编指令的线性序列。该模型以汇编文本为输入，将函数的语义嵌入向量作为输出。Asm2vec 是第一个采用 NLP 的表征学习模型来为汇

编代码构建语义嵌入的方法，具有优秀的抗混淆和抗编译优化特性，遗憾的是它不能用于跨架构的代码相似性比较。Zuo 等[12]实现的二进制代码相似性检测系统 INNEREYE 分别从基本块、CFG 路径和程序组件 3 个层次来研究程序的语义相似性。该系统的基本块语义嵌入过程是利用 NLP 的 Word2vec[13]（Word2vec 是谷歌于 2013 年推出的一款 NLP 工具，它能基于上下文将单词向量化，从而定量度量单词之间的语义关系[14]）和属于循环神经网络（Recurrent Neural Network，RNN）的 LSTM（Long Short-term Memory）[15]模型来实现的。Luca 等[16]在提出的 SAFE 模型中也利用 Word2vec 来实现汇编语言的指令嵌入，不同的是他们利用自关注的 RNN 模型来学习指令序列的上下文关系[17]，从而实现函数语义嵌入。SAFE 摒弃了以 CFG 为中间表示的做法，取而代之的是利用神经网络直接将汇编代码的语义信息嵌入到高维向量中，这样做既省去了耗时的 CFG 提取过程，又避免了引入人为偏见。但在跨指令架构的检测任务中，理论上，随着系统支持架构种类的增加，训练样本库的规模也需要根据不同架构的组合成倍扩大，这在一定程度上限制了该模型的可扩展性。Luo 等[18]提出的 GeneDiff 系统是第一个使用语义表征模型来学习二进制代码的中间语言的克隆检测的研究。该技术借助 VEX IR 消除不同指令架构的二进制代码之间的差异，再利用 PV-DM 模型为函数的 VEX IR 代码生成语义嵌入向量。上述这些将函数转化为指令序列的方法[16-18]都存在一个共同的问题，即当遇到执行流复杂的函数时，很难完全覆盖所有有效的执行路径。

Yu 等[19]在最新的研究中用语义感知、GNN 和 CNN 这 3 种不同的神经网络分别学习二进制代码的基本块的语义、CFG 的结构和节点的顺序信息，并对相同类型的不同网络模型的性能做了横向比较，对后续的研究具有较大的指导意义。

3 问题定义

本文认为，由同一源代码，经不同编译器，应用不同编译优化配置，针对不同指令架构，编译得到的不同二进制代码是相似的。本文主要关注的是跨架构的二进制函数的静态相似性模型的构建。

给定两个二进制函数 f_n，f_m（它们是由同一编译器编译而来的，但可能应用了不同的编译优化配置，采用了不同的指令架构），假设存在未知形式的判别式 π，$\pi(f_n,f_m)=1$ 代表 f_n 和 f_m 是相似的，而 $\pi(f_n,f_m)=-1$ 代表它们是相异的。我们的目的是，通过对有限数量的 $\langle f_n,f_m,\pi(f_n,f_m)\rangle$ 实例的观察，寻找到一个映射 ϕ，ϕ 可以将任意函数 f 映射为向量 $\boldsymbol{\mu}_f$，记作 $\boldsymbol{\mu}_f=\phi(f)$，使得如果 $\pi(f_n,f_m)=1$，则 $Sim(\boldsymbol{\mu}_{f_n},\boldsymbol{\mu}_{f_m})\to 1$，而如果 $\pi(f_n,f_m)=-1$，则 $Sim(\boldsymbol{\mu}_{f_n},\boldsymbol{\mu}_{f_m})\to -1$。这里 $Sim(,)$ 代表函数间的相似度，可以用向量间的距离来表示。

4 基于神经网络的二进制函数相似性检测技术

4.1 技术概述

本文采用 CFG 作为二进制函数的中间表示，将所有函数的集合记作 F，F 中任意函数的 CFG 记作 $g=\langle V,E\rangle$，$g\in F$，其中 V，E 分别代表 g 中节点和边的集合。初始时，图中节点 V 为函数基本块的字节码，边 E 用于表示基本块之间的执行流跳转。基于相似函数具有相似 CFG 的假设，即基本块的指令序列具有语义相似性，则可以将任意节点 $v\in V$ 映射为向量 $\boldsymbol{\mu}_v\in P_v$，使得语义相似的节点在向量空间 P_v 中的距离也相近，本文将该映射过程称为基本块语义嵌入；而函数的 CFG 也具有结构相似性，则可以将任意函数的 CFG g 映射为向量 $\boldsymbol{\mu}_g\in P_F$，使得结构相似的 CFG 在向量空间 P_F 中的距离也相近，将该映射过程称为函数 CFG 嵌入。

本文提出的技术的函数嵌入过程主要分为 3 部分，包括代码预处理、基本块语义嵌入和函数 CFG 嵌入，如图 1 所示。基于该技术的 FuncSim 原型系统的架构如图 2 所示。代码预处理模块会为二进制函数构建 CFG 并对其结构进行优化，再将基本块的字节码翻译为 VEX IR 代码；基本块语义嵌入模块会对基本块 VEX IR 代码做进一步抽象，再利用 PV-DM 模型来为 VEX IR 代码生成语义嵌入向量；函数 CFG 嵌入模块利用改进后的 Structure2vec 模型计算整个函数 CFG 的嵌入向量；最后，系统会通过计算函数嵌入向量间的余弦距离来度量函数间的相似度；此外，系统还构建和维护了一个代码数据库，用于存储代码的相关信息和系统产生的中间数据。

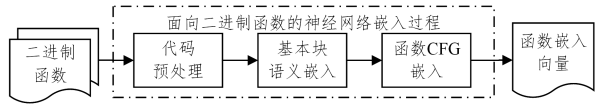

图 1 面向二进制函数的神经网络嵌入过程

Fig. 1 Process of neural network embedding for binary function

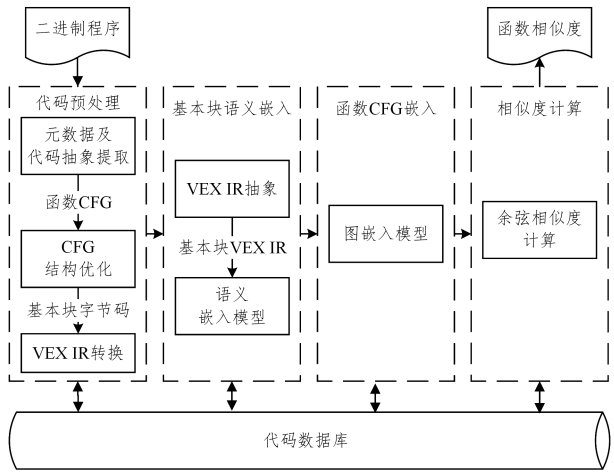

图 2 FuncSim 系统架构

Fig. 2 Architecture of FuncSim system

4.2 代码预处理

代码预处理的目的是分析输入的可执行文件，从中提取有价值的信息并获得基本块的 VEX IR 代码和函数 CFG。

4.2.1 元数据及代码抽象提取

程序的可执行文件通常包含许多可直接获取的重要信息，系统首先提取这些元数据，然后利用程序解析器提取代码的高级信息和抽象表示，最后将这些信息关联起来并有序地存储到代码数据库中，以便后续的查找和使用。提取的信息如表 1 所列。

表 1　提取信息列表

Table 1　List of extracted information

级别	种类	类型
二进制	指令架构	X86-64/AArch64
	基地址	整型
	哈希值	16 进制
	字节序	MSB/LSB
	编译器	GCC/GLANG
	优化级别	O0/O1/O2/O3
函数级	函数名	List
	父/子函数	List，CG(Call Graph)
	基本块	CFG
	字节码	Bytes

4.2.2　CFG 结构优化

CFG 能够忠实还原代码的分布式内存和非线性执行的特性，其节点为程序基本块，函数原始 CFG 的结构往往可以被进一步优化。优化的目标是最大限度地恢复代码的空间连续性，为后续的分析和比较做铺垫。优化方法主要有基本块合并和选择性内联。

(1)基本块合并。某些情况下代码在内存中虽然是非线性分布的，但某些连续的基本块在逻辑上可以被合并为一个基本块。基本块合并的算法描述为：遍历函数 f 的基本块，找到只有一个出口的基本块 b，如果 b 的后续基本块 b' 只有一个入口，则将 b 和 b' 合并，同时如果 b 的结尾是跳转指令则一并删除。

(2)选择性内联。函数是代码的封装，有助于程序的编写和阅读，但会破坏代码在内存空间中的连续性，这也是程序代码有别于自然语言的原因之一。函数名在汇编代码中以地址的形式存在，程序解析器可以从可执行文件的导入导出表中识别出标准库函数和系统调用，并用具有全局唯一性的符号来区分这类函数。表征学习模型能够通过代码的上下文来预测这类函数的功能和语义，而不必关心函数的具体实现方法。但在采用静态调用的发行版程序中，更多的函数是很难被解析器准确识别的，因为这类函数的入口地址具有随机性，表征学习模型也难以通过有限的上下文来预测其功能和语义，此时就需要对函数体做进一步的分析。对函数采用内联展开，虽然可以恢复基本块内部的空间连续性，但有时也会产生副作用：一方面，当子函数的代码量超过父函数时，扩展后父函数的语义将由子函数所主导，这可能导致父函数与其他函数的相似性不再由父函数自身所决定，反而由子函数所决定；另一方面，如果子函数的结构过于复杂，则内联扩展反而会破坏当前基本块内部的空间连续性，同时也改变了父函数的 CFG 的整体结构，从而影响相似性判定。综上所述，本文设计了一种选择性内联机制，其算法描述为：假设 f_S 是函数 f 的一个子函数，$|f|$ 表示 f 的长度，即 f 中包含全部指令的条数。当 f_S 同时满足以下两个条件时，对 f_S 做内联扩展：1) f_S 无已识别的全局函数名，且在函数嵌入库中无嵌入向量；2) $|f_S|$ 与 $|f|$ 的比值小于 0.5，且 f_S 经基本块合并后只由一个基本块构成。

4.2.3　基本块 VEX IR 翻译

许多程序动态分析工具会将程序的汇编代码转换成某种中间表示，这些中间表示虽然各异，但都有一个共同的目的，就是消除代码在指令架构上的差异，而不改变程序的原始功能和语义，从而实现跨架构的代码分析，本文就借鉴了这一做法。此外，根据 Poeplau 等[20]的研究，目前各种动态执行框架所使用的不同中间表示在表示能力和执行速度方面没有明显的差异，本文最终决定采用 VEX IR 作为基本块字节码的中间表示，因为其对多种指令架构的支持最好。

每条汇编指令会被翻译为若干条 VEX IR 指令并以 IMarK 标记分隔。图 3 给出了由同一 C 源码采用 GCC 编译器和 O0 配置编译得到的不同指令架构的汇编代码及其对应的 VEX IR 代码。从图中可以看出，两者的 VEX IR 代码的主要操作基本相同，其主要差异在于临时寄存器的选择。

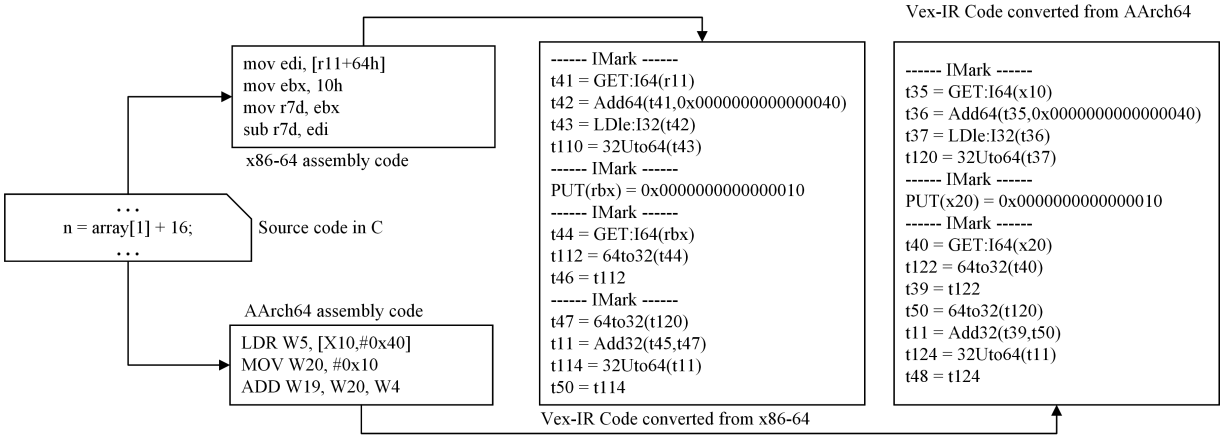

图 3　同一源码不同指令架构的汇编代码及其 VEX IR

Fig. 3　Assembly code and its VEX IR in different ISA from a same source code

4.3　基本块语义嵌入

在基本块级别，确定源代码与二进制代码之间的对应关系具有一定的难度。PV-DM 模型采用无监督的学习方法，非常适合基本块级的语义嵌入任务。本文中的基本块嵌入过程为：首先对基本块的 VEX IR 代码做进一步抽象，然后利用训练好的 PV-DM 模型预测基本块 VEX IR 代码的语义嵌入向量。

4.3.1　自定义 VEX IR 抽象规则

经预处理得到的基本块 VEX IR 代码还不适合 NLP 模型学习，因为其中包含许多简单运算符(例如“=”“( )”“,”

等)以及低频词汇(即数值常量、内存地址和特殊字符串)。运算符的主要作用是提高代码的可读性,本文对其采取删除操作,并用空格分隔前后的标识符。低频词汇的数值具有随机性,与程序的功能和语义关系较小,但会扩大语料库,影响模型的学习效果。系统使用标签来替换相同类型的低频词汇,抽象规则中低频词汇类型与标签的对应关系如表 2 所列。图 4 给出了对VEXIR代码应用自定义抽象规则后的代码示例。

表 2 低频词汇抽象对照表

Table 2 List of low-frequency word labels

低频词汇类型	标签
数值常量	Num
数据地址	Mem
字符串	Str
未识别函数地址	Func

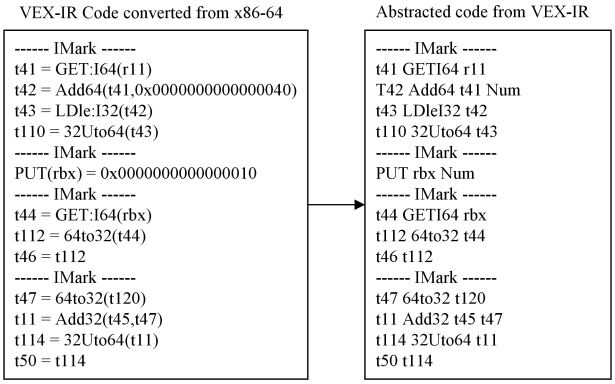

图 4 VEX IR 代码抽象

Fig. 4 Abstraction for VEX IR code

4.3.2 基本块语义嵌入模型

本文将 NLP 的 PV-DM 模型用于基本块语义嵌入,该模型是著名的词嵌入模型 CBOW(Continuous Bag-of-Words Mode)[13]的扩展,CBOW 模型能够通过文本的上下文来预测中心词的语义嵌入向量,而 PV-DM 模型则是在此基础上引入了段落向量的概念。本文将 VEX IR 代码视为一种特殊的语言,将 VEX IR 的指令和操作对象视为“单词”,将由同一条汇编指令翻译而来的一组 VEX IR 指令视为一个“句子”,将基本块中全部的 VEX IR 指令序列视为“段落”,从而预测基本块的语义嵌入向量。

在初始化阶段,PV-DM 模型会将语料库中的每个基本块映射为一个随机且唯一的 p 维段落向量 $\boldsymbol{d}$,对应 n 维段落向量矩阵 $\boldsymbol{D}$ 中的一个列向量,同时代码中的每个标识符(token)将会被初始化为一个随机且唯一的 q 维的词向量 $\boldsymbol{w}$,对应 m 维词向量矩阵 $\boldsymbol{W}$ 中的一个列向量。PV-DM 模型会设置一个大小为 k 的滑动窗口,每输入一条文本序列 $\boldsymbol{w}_1,\boldsymbol{w}_2,\boldsymbol{w}_3,\cdots,\boldsymbol{w}_T$,窗口就会从序列的头部开始向尾部逐词滑动,每次选取 k 个 token 并将一个固定位置的 token 选作中心词。

在训练和预测阶段,PV-DM 模型首先对段落向量 $\boldsymbol{d}$ 和除中心词以外的其他词向量 $\boldsymbol{w}_{t-k},\cdots,\boldsymbol{w}_{t+k}$ 进行连接(Concatenate)操作;然后将中心词的预测视为一种多分类问题,分类过程通过一个二进制哈夫曼树(Binary Huffman Tree)结构的层次 Softmax(Hierarchical Softmax)网络来实现[21-23]。PV-DM 模型结构及嵌入过程如图 5 所示。

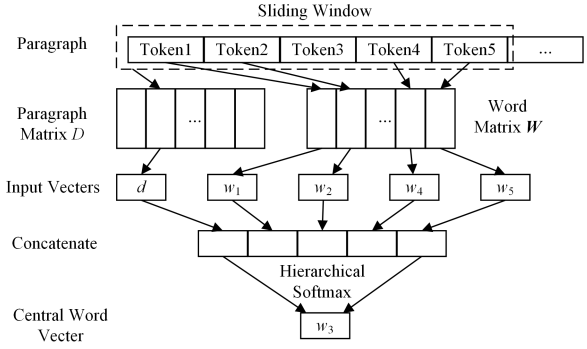

图 5 PV-DM 模型的语义嵌入过程

Fig. 5 Process of PV-DM model semantic embedding

PV-DM 模型的目标函数是最大化如下形式的平均对数似然概率:

$$\frac{1}{T}\sum_{t=k}^{T-k}\log P(\boldsymbol{w}_t\mid\boldsymbol{w}_{t-k},\cdots,\boldsymbol{w}_{t+k}) \tag{1}$$

其中,$P(\boldsymbol{w}_t\mid Context(\boldsymbol{w}_t))$表示在当前上下文中预测结果恰好为中心词的概率,其计算方法如下:

$$P(\boldsymbol{w}_t\mid\boldsymbol{w}_{t-k},\cdots,\boldsymbol{w}_{t+k})=\frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \tag{2}$$

其中,y_i表示输出的第 i 个词的非规范化的对数概率,计算方法如下:

$$y=\boldsymbol{U}\cdot h(d,\boldsymbol{w}_{t-k},\cdots,\boldsymbol{w}_{t+k};\boldsymbol{D},\boldsymbol{W})+b \tag{3}$$

其中,$\boldsymbol{U},\boldsymbol{b}$ 是层次 Softmax 函数的参数矩阵,函数 $h(\cdot)$表示对当前的段落向量和上下文的词向量做连接操作。

4.3.3 基本块语义嵌入模型的优化

为了节省时间,本文直接将 x86-64 和 AArch64 架构的二进制程序翻译为 VEX IR 代码,并用于构建 VEX IR 语料库。

在训练过程中,基本块语义嵌入模型通过随机梯度上升的方法来更新参数和语义嵌入向量,梯度通过反向传播[24]来获得。在预测阶段,该模型的参数 $\boldsymbol{U},\boldsymbol{b}$ 和词汇表 $\boldsymbol{W}$ 固定不变,只是迭代更新段落向量矩阵 $\boldsymbol{D}$。

4.4 函数 CFG 嵌入

在基本块语义嵌入过程之后,将语义嵌入向量同对应的基本块建立索引关系,然后便可以进一步对函数 CFG 进行嵌入。原始的 Structure2vec 模型的输入为网络图,图中节点可以具有初始特征,模型通过考虑一定范围内的邻居节点特征和图的拓扑结构来更新图中每个节点的特征向量。为得到函数 CFG 嵌入向量,FuncSim 系统在 Structure2vec 模型的基础上对顶点向量做进一步的聚合操作,同时本文还借鉴了 MPNN 架构,对 Structure2vec 模型进行了改进。

4.4.1 MPNN 架构

MPNN 是由 Gilmer 等[8]提出的一种图神经网络通用框架,用于帮助研究人员理解图神经网络的结构和原理,从而改进模型。MPNN 将现有图神经网络抽象为两个阶段,分别是消息传递(Message Passing)和读出(Readout)阶段。本文假设给定任意 CFG,记作 $g=\langle V,E\rangle$,节点 $v\in V$ 的原始特征为 p 维段落向量$\boldsymbol{\mu}_v$,节点 v 的邻接节点的集合记作 $N(v)$(由于

CFG 是有向图，因此 $N(v)$只包含 v 的下一跳节点）。在消息传递阶段，MPNN 定义了消息函数M_t和更新函数U_t，分别对应神经网络的输入层和隐藏层，其中 t 为模型运行的时间步。函数的形式如下：

$$\boldsymbol{m}_v^t=\sum_{w\in N(v)}M_t(\boldsymbol{h}_w^{t-1}) \tag{4}$$

$$\boldsymbol{h}_v^t=U_t(\boldsymbol{h}_v^{t-1},\boldsymbol{m}_v^t) \tag{5}$$

其中，$\boldsymbol{m}_v^t$是节点 v 在第 t 时间步接收到的消息，$\boldsymbol{h}_v^t$是节点 v 在第 t 时间步的状态。在每一轮的迭代中，节点信息会通过邻接节点逐步向远端传播。

经历 T 轮迭代以后，在读出阶段 MPNN 通过读出函数 R 来计算 g 的图嵌入向量$\boldsymbol{\mu}_g$，函数的形式如下：

$$\boldsymbol{\mu}_g=R(h_v^{\mathrm{T}}\mid v\in V) \tag{6}$$

最后用合适的表达式替换框架中的 M,U,R 函数，即可实现图嵌入网络的构建和改进。

4.4.2　函数 CFG 嵌入模型

依据 MPNN 架构，本文在 Structure2vec 模型的基础上进行了改进，用一个 n 层的全连接神经网络来实现消息函数 M，其定义如下：

$$m_v^t=\boldsymbol{P}_1\times ReLU(\boldsymbol{P}_2\times\cdots ReLU(\boldsymbol{P}_n\times\sum_{w\in N(v)}\boldsymbol{h}_w^{t-1})) \tag{7}$$

其中，$\boldsymbol{P}_i(i=1,\cdots,n)$为 $r\times r$ 维的参数矩阵，r 为最终 CFG 嵌入向量的维度，$\boldsymbol{h}_v^0:=\boldsymbol{W}_1\times\boldsymbol{\mu}_v$，$\boldsymbol{W}_1$为 $r\times p$ 维的参数矩阵。在更新函数 U 中，本文用 GRU(Gate Recurrent Unit)网络替换 Structure2vec 模型中的 RNN 网络，定义如下：

$$\boldsymbol{h}_v^t=\mathrm{GRU}(\boldsymbol{h}_v^{t-1},\boldsymbol{m}_v^t) \tag{8}$$

GRU[25-26]是 LSTM 网络的改进，其本质也是一种 RNN，在解决传统 RNN 存在的长期记忆和梯度消失的问题的同时，还控制了参数的数量。在每一轮计算前，我们将图中所有节点及其邻接节点的状态向量输入模型。由式(7)、式(8)可知，当经历 T 轮迭代后，节点 v 的状态$\boldsymbol{h}_v^T$将包含距离为 T(即深度)的子节点信息。

Xu 等于 2018 年发表的关于 GNN 的研究[27]已经证明，将 *sum* 函数用作读出函数 R 的效果最好，本文采纳这一方法，具体定义如下：

$$\boldsymbol{\mu}_g=\boldsymbol{W}_2\sum_{v\in V}\boldsymbol{h}_v^{\mathrm{T}} \tag{9}$$

其中，$\boldsymbol{W}_2$是 $r\times r$ 维的参数矩阵。图 6 给出了函数 CFG 嵌入模型的整体架构。

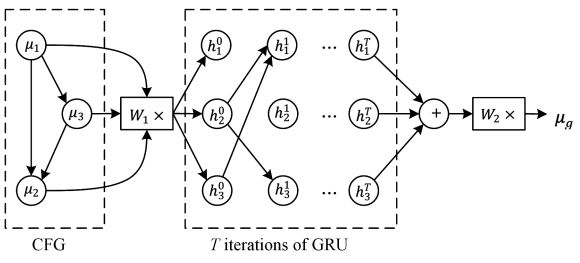

图 6　CFG 嵌入网络的架构

Fig. 6　Architecture of CFG embedding network

4.4.3　函数 CFG 嵌入模型的优化

本文用两个完全相同的 CFG 嵌入网络组成一个孪生架构，两个神经网络使用相同的超参并共享参数。孪生网络的架构如图 7 所示。

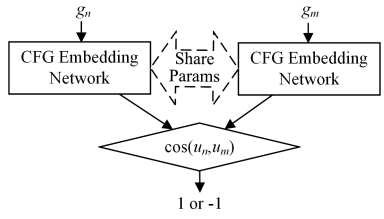

图 7　孪生架构

Fig. 7　Siamese architecture

函数 CFG 嵌入模型采用有监督的训练方法，训练样本为带有相似性标签的样本对，即从网上下载源代码，我们采用不同编译优化配置，针对不同指令架构，编译得到二进制代码库，选择两个由同一源码编译而来的不同二进制函数来构建相似函数对$\langle f_n,f_m,1\rangle$；在不同目录下，随机选择两个名称不同的函数的二进制代码来构建相异函数对$\langle f_n,f_m,-1\rangle$。

函数相似度的定义如下：

$$Sim(f_n,f_m)=\cos(\boldsymbol{\mu}_n,\boldsymbol{\mu}_m) \tag{10}$$

给定一个拥有 K 个样本对的训练集 F，$f_n,f_m\in F$，通过反向传播和随机梯度下降的方法来更新函数 CFG 嵌入模型的参数，使得目标函数(即交叉熵损失函数)的值最小化，具体形式如下：

$$Loss=\sum_{i=1}^{K}(Sim(f_n,f_m)-\pi(f_n,f_m))^2 \tag{11}$$

5　实验

5.1　设计与准备

本文提出的基于神经网络的二进制函数相似性检测技术的目标是弥补当前研究中存在的 3 点不足。实验分别对基本块语义嵌入模型、函数 CFG 嵌入模型和 FuncSim 系统整体的准确率进行了评估，从而回答以下 3 个问题。

问题 1　检测系统能否进行跨架构检测，系统的可扩展性如何？

问题 2　改进的函数 CFG 嵌入模型的学习效果较原有模型是否有提高？

问题 3　表征学习模型所获得的基本块语义嵌入向量是否优于人工设计的特征向量？

在现有二进制函数相似性检测研究中，Xu 等于 2018 年提出的 Gemini 方案[9]是一个比较优秀的研究成果，其公开了系统的源代码和部分测试集。后来许多方案都受到了 Gemini 的启发，并将其作为参考对象，本文也以 Gemini 为基准进行对比测试。Gemini 系统有两个主要的功能模块，分别是 ACFG 提取器和图嵌入模型。本文通过构建跨架构的测试集来测试 FuncSim 系统检测的准确性，从而验证系统的跨架构特性和可扩展性；对于 FuncSim 系统的函数 CFG 嵌入模型，本文以 Gemini 的 ACFG 嵌入模型为对照，验证模型改进后的效果；FuncSim 系统的基本块语义嵌入模型由于采用了无监督的训练方法，将其嵌入向量直接用于基本块级的相似度计算的效果并不理想，而 Gemini 的节点特征向量是由人工定义的基本块级的属性特征所构建的，因此本文用 Gemini 的节点特征向量来替换 FuncSim 系统的基本块语义嵌入向量，对比系统前后的性能变化，从而验证表征学习在处理基本块嵌

入任务的优越性。

FuncSim 系统的函数 CFG 提取方法参考了 Gemini 的 ACFG 提取器；第三方工具包 PyVEX 提供了 VEX IR 翻译的 Python 接口；第三方工具包 Gensim 提供了 PV-DM 模型的实现，即 Doc2vec 模型；函数 CFG 嵌入模型是利用 TensorFlow 1.14 工具包来实现的。实验在一台拥有两颗 Intel XeonSliver 4210 处理器、两条 32GB DDR4 内存的服务器上完成。

基本块语义嵌入模型的 VEX IR 语料库的构建方法是，从 x86-64 和 AArch64 版本的 Debian 10 系统的“/bin”“/sbin”和“/lib”目录下分别提取了共计 4Gbs 的可执行二进制文件作为训练样本。而函数 CFG 嵌入模型的样本库的构建方法则是从网上下载开源项目 OpenSSL 1.1.1h-dev 和 C 标准函数库的源代码，采用 GCC 9.3 编译器、3 种编译优化配置(O0，O1，O2)，针对 x86-64 和 AArch64 两种指令架构，编译得到 11053 个函数的 60978 个不同版本的二进制代码，将它们划分为没有交集的训练集和测试集，每从中选择一个函数，便可以随机选择一个与之相似的函数来组成正样本对和一个与之不相似的函数来组成负样本对，最终正负样本各占一半。二进制函数代码分类和数量的统计如表 3 所列。需要说明的是，GCC 编译器的编译优化配置除了 O0，O1，O2 之外还有很多，为简化实验，本文只针对 3 种常用的优化选项进行了研究。其中，O0 表示不做优化，是编译器的默认配置；O1 表示只做部分优化，编译后的二进制文件体积最小；O2 表示较 O1 配置做更多的寄存器和指令级的优化，但编译过程需要更多的时间和资源。

表 3 二进制函数代码统计

Table 3 Number of binary function code

(单位：个)

ISA	Optimization	Training set	Test set	Total
x86-64	O0	30096	3063	33159
	O1			
	O2			
AArch64	O0	25380	2439	27819
	O1			
	O2			

通过训练，我们将 FuncSim 模型的超参数确定为：基本块语义嵌入向量的维度 p 为 50，窗口大小为 8，最小词频为 8，迭代更新次数为 20，CFG 嵌入向量的维度 r 为 64，全连接网络的深度 n 为 2，迭代更新的次数 T 为 5。

函数相似性检测的目标任务是判断输入的两个函数是否相似，此任务可以被看作一个二分类问题，本文采用准确率作为评价指标。

5.2 测试与结论

实验的测试集有单一指令架构和多指令架构之分，单一指令架构有 x86-64 测试集和 AArch64 测试集，但对于 AArch64 测试集，本文还特别增加了模型训练集不包含 AArch64 样本的测试条件，即 AArch64(Untrained)测试；多指令架构为 x86-64 和 AArch64 的交叉测试集。测试对象除了原始的 Gemini 和 FuncSim 系统，还增加了后者的一个变种 FuncSim2，即用 Gemini 的节点特征向量替代 FuncSim 的基本块嵌入向量。不同模型在测试集下的准确率如表 4 所列。通过 Gemini 和 FuncSim 系统的测试数据的对比可以看到，FuncSim 系统在各项测试中的准确率都高于 Gemini 系统。

表 4 模型在不同测试集下的准确率

Table 4 Accuracy of models on different test set

(单位：%)

Model	x86-64	AArch64 (Untrained)	AArch64	x86-64 and AArch64
Gemini	77.6	63.5	79.8	76.3
FuncSim	84.7	81.4	87.6	86.4
FuncSim2	79.2	—	82.7	78.8

特别是在 AArch64(Untrained)测试集中，FuncSim 系统的表现依然十分稳定，当增加了相应的训练样本时，模型效果有所提升，这说明 VEX IR 能够有效屏蔽不同指令架构之间的差异，使得系统支持不同指令架构的代价较小，从而得出结论，FuncSim 较 Gemini 系统具有更强的跨架构特性和可扩展性，回答了问题 1。

Gemini 和 FuncSim2 系统都属于 ACFG 嵌入模型，两者之间的主要差异在于后者对图嵌入模型进行了改进。从数据中可以看到改进后的系统的准确率有所提升，从而得出结论，改进后的图嵌入模型能够更好地利用 CFG 的节点和结构信息，回答了问题 2。

FuncSim 和 FuncSim2 系统之间的主要差异在于节点特征向量的定义和提取方式，前者使用表征学习，而后者使用人工定义的特征。从数据中可以看到 FuncSim 系统的准确率明显高于 FuncSim2 系统，从而得出结论，基于表征学习的基本块嵌入过程能够有效地避免引入人为偏见，其嵌入向量在相似性检测中的表现优于人工设计的特征向量，回答了问题 3。

结束语 随着代码复用在软件开发中的流行和代码规模的不断增长，代码相似性检测无论是在软件知识产权保护领域还是安全领域必将发挥越来越大的作用。本文提出的基于神经网络的二进制函数相似性检测技术利用了统一的中间表示来消除二进制代码在指令架构上的差异，其基于 NLP 表征学习模型的基本块语义嵌入过程有效地避免了引入人为偏见，同时改进的图神经网络更适合函数语义和逻辑信息的嵌入任务。实验结果表明，在跨架构二进制代码的相似性检测任务中，FuncSim 原型系统较现有方案具有更强的可扩展性且准确率更高。

下一步工作将研究不同编译器对二进制代码的影响，并测试系统的跨编译器特性；针对面向大规模代码库的相似函数搜索任务，研究提高 FuncSim 系统检测效率和降低误报率的方法。

参 考 文 献

[1] ALLEN F E. Control Flow Analysis[J]. ACM Sigplan Notices, 1970, 5(7): 1-19.

[2] QIAN F, ZHOU R, XU C, et al. Scalable Graph-based Bug Search for Firmware Images[C]// Proceedings of AcmSigsac Conference on Computer & Communications Security. New York: ACM, 2016: 480-491.

[3] Valgrind. Python Bindings for Valgrind's VEX IR[EB/OL].

(2020-07-28)[2020-09-08]. https://github.com/angr/pyvex.
[4] Valgrind. ValgrindHome[EB/OL]. (2020-07-13)[2020-07-13]. https://www.valgrind.org/.
[5] NETHERCOTE N, SEWARD J. Valgrind: AFramework for Heavyweight Dynamic Binary Instrumentation[C]// Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2007: 89-100.
[6] LE Q, MIKOLOV T. Distributed Representations of Sentences and Documents [C]// Proceedings of International Conference on Machine Learning. Beijing: JMLR, 2014: 1188-1196.
[7] DAI H, DAI B, SONG L. Discriminative Embeddings of Latent Variable Models for Structured Data [C]// Proceedings of International Conference on Machine Learning. New York: JMLR, 2016: 2702-2711.
[8] GILMER J, SCHOENHOLZ S S, RILEY P, et al. Neural Message Passing for Quantum Chemistry[C]// Proceedings of International Conference on Machine Learning. Sydney: JMLR, 2017: 1263-1272.
[9] XU X, LIU C, FENG Q, et al. Neural Network-based Graph Embedding for Cross-platform Binary Code Similarity Detection [C]// Proceedings of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2017: 363-376.
[10] LIU B, HUO W, ZHANG C, et al. αDiff: Cross-version Binary Code Similarity Detection with DNN[C]// Proceedings of ACM/IEEE International Conference on Automated Software Engineering. New York: ACM, 2018: 667-678.
[11] DING S H H, FUNG B C M, CHARLAND P. Asm2vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization[C]// Proceedings of IEEE Symposium on Security and Privacy. San Francisco: IEEE, 2019: 472-489.
[12] ZUO F, LI X, YOUNG P, et al. Neural Machine Translation Inspired Binary Code Similarity Comparison beyond Function Pairs [C]// Network and Distributed Systems Security Symposium. 2019.
[13] MIKOLOV T, CHEN K, CORRADO G, et al. Efficient Estimation of Word Representations in Vector Space [C]// International Conference on Learning Representations. 2013.
[14] Google. Tool for Computing Continuous Distributed Representations of Words[EB/OL]. (2013-07-30)[2020-03-07]. https://code.google.com/archive/p/word2vec/.
[15] HOCHREITER S, SCHMIDHUBER J. Long Short-term Memory[J]. Neural Computation, 1997, 9(8): 1735-1780.
[16] MASSARELLI L, DI LUNA G A, PETRONI F, et al. SAFE: Self-attentive Function Embeddings for Binary Similarity[C]// Proceedings of International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Gothenburg: Springer, 2019: 309-329.
[17] LIN Z, FENG M, SANTOS C N, et al. A Structured Self-attentive Sentence Embedding [C]// International Conference on Learning Representations. arXiv: 1703.03130, 2017.
[18] LUO Z, WANG B, TANG Y, et al. Semantic-Based Representation Binary Clone Detection for Cross-Architectures in the Internet of Things[J]. Applied Sciences, 2019, 9(16): 3283-3304.
[19] YU Z, CAO R, TANG Q, et al. Order Matters: Semantic-Aware Neural Networks for Binary Code Similarity Detection[C]// Proceedings of the AAAI Conference on Artificial Intelligence. Palo Alto: AAAI, 2020: 1145-1152.
[20] POEPLAU S, FRANCILLON A. Systematic Comparison of Symbolic Execution Systems: Intermediate Representation and Its Generation[C]// Proceedings of Annual Computer Security Applications Conference. New York: ACM, 2019: 163-176.
[21] MORIN F, BENGIO Y. Hierarchical Probabilistic Neural Network Language Model[C]// Proceedings of International Conference on Artificial Intelligence and Statistics. AISTATS, 2005: 246-252.
[22] MNIH A, HINTON G E. A Scalable Hierarchical Distributed Language Model[C]// Proceedings of International Conference on Neural Information Processing Systems. Red Hook: Curran Associates, 2008: 1081-1088.
[23] MIKOLOV T, SUTSKEVER I, CHEN K, et al. Distributed Representations of Words and Phrases and their Compositionality[C]// Proceedings of International Conference on Neural Information Processing Systems. Red Hook: Curran Associates, 2013: 3111-3119.
[24] RUMELHART D E, HINTON G E, WILLIAMS R J. Learning Representations by Back-propagating Errors[J]. Nature, 1986, 323(6088): 533-536.
[25] GREFF K, SRIVASTAVA R K, KOUTNIK J, et al. LSTM: A Search Space Odyssey[J]. IEEE Transactions on Neural Networks, 2017, 28(10): 2222-2232.
[26] CHO K, VAN MERRIENBOER B, BAHDANAU D, et al. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches[C]// Proceedings of Workshop on Syntax, Semantics and Structure in Statistical Translationar. Doha: Association for Computational Linguistics, 2014: 103-111.
[27] XU K, HU W, LESKOVEC J, et al. How Powerful are Graph Neural Networks? [C]// Proceedings of the 7th International Conference on Learning Representations. 2019: 1-17.

FANG Lei, born in 1989, postgraduate, assistant engineer. His main research interests include security of network information and so on.

WEI Qiang, born in 1979, Ph.D, professor, Ph.D supervisor. His main research interests include security of network information and so on.