文章编号:1671-4598(2023)03-0056-07 DOI:10.16526/j.cnki.11—4762/tp.2023.03.009 中图分类号:TP3 文献标识码:A

基于混合注意力机制的软件缺陷预测方法

刁旭炀,吴 凯,陈 都,周俊峰,高 璞
(上海机电工程研究所,上海 201109)

摘要:软件缺陷预测技术用于定位软件中可能存在缺陷的代码模块,从而辅助开发人员进行测试与修复;传统的软件缺陷特征为基于软件规模、复杂度和语言特点等人工提取的静态度量元信息;然而,静态度量元特征无法直接捕捉程序上下文中的缺陷信息,从而影响了软件缺陷预测的性能;为了充分利用程序上下文中的语法语义信息,论文提出了一种基于混合注意力机制的软件缺陷预测方法 DP-MHA (defect prediction via mixed attention mechanism);DP-MHA 首先从程序模块中提取基于 AST 树的语法语义序列并进行词嵌入编码和位置编码,然后基于多头注意力机制自学习上下文语法语义信息,最后利用全局注意力机制提取关键的语法语义特征,用于构建软件缺陷预测模型并识别存在潜在缺陷的代码模块;为了验证 DP-MHA 的有效性,论文选取了6个 Apache 的开源 Java 数据集,与经典的基于 RF 的静态度量元方法、基于 RBM+RF、DBN+RF 无监督学习方法和基于 CNN 和 RNN 深度学习方法进行对比,实验结果表明,DP-MHA 在 F1 值分别提升了 16.6%、34.3%、26.4%、7.1%、4.9%。

关键词:软件缺陷预测;语法语义信息;静态度量元;多头注意力机制;全局注意力机制

Method of Software Defect Prediction Based on Mixed Attention Mechanism

DIAO Xuyang, WU Kai, CHEN Du, ZHOU Junfeng, GAO Pu
(Shanghai Electro-Mechanical Engineering Institute, Shanghai 201109, China)

Abstract: Software defect prediction technique is used to locate code module with possible defects in software, which helps developers test and fix bugs. Traditional defect prediction features are manual static code metrics based on software scale, complexity and language characteristic. However, these features cannot directly capture defect information from program context, resulting in the degradation of defect prediction performance. To take full advantage of syntactic and semantic features in program context, a method called Defect Prediction via Mixed Attention Mechanism (DP-MHA) is proposed in this paper. Firstly, DP-MHA first extracts the AST tree-based syntactic and semantic sequence from program modules and performs word embedding and positional encoding. Then the contextual syntax and semantic information is learned by the Multi-head attention mechanism. Finally, the global attention mechanism is used to extract the key syntactic and semantic features, which are used to build a software defect prediction model and identify code snippets with potential defects. In order to verify the effectiveness of DP-MHA, six Apache open-source Java projects are selected to be compared with the state-of-the-art methods including classical static code metric method based on RF, unsupervised learning method based on RBM+RF, DBN+RF and deep learning method based on CNN, RNN. The experimental results show that DP-MHA improves F1-Measure by 16.6%, 34.3%, 26.4%, 7.1% and 4.9%, respectively.

Keywords: software defect prediction; syntactic and semantic features; static code metrics; Mixed attention mechanism; global attention mechanism

0 引言

软件稳定性是衡量软件质量的重要标准,成千上万行复杂代码往往存在会导致软件失效的风险。为了帮助开发和测试人员更加高效地定位潜在缺陷,软件缺陷预测技术正广泛运用在软件的代码审查阶段,是软件工程[1]研究领域中的重要研究方向之一。

软件缺陷预测技术[2-3]利用软件代码库中的历史项目进行缺陷度量元的设计与提取,并将基于静态度量元的特征放入缺陷预测分类器中进行模型训练,最终得到的缺陷预测模型能够有效识别存在缺陷的代码模块。软件开发与测试人员可根据预测结果对相关模块进行复核校验,从而节省了查找定位缺陷的时间,提升了软件质量维护与保证的效率。软件缺陷预测主要分为同项目软件缺陷预测[4]与跨项目软件缺陷预测[5-6]两个研究方向。此外,依据提取的程序特征颗粒度,可将软件缺陷预测技术分为基于文件级、函数级、变更级的缺陷预测。

传统的缺陷预测方法通常是基于人工设计的静态度量元特征来构建软件缺陷预测模型。相关研究人员已经设计出了相关有辨别度的特征可以有效区分有缺陷和无缺陷的程序模块,主要包括基于运算符和操作数的 Halstead[7]特

收稿日期:2022-07-31; **修回日期**:2022-09-07。
作者简介:刁旭炀(1995-),男,上海奉贤人,工程师,硕士研究生,主要从事软件工程和机器学习方向的研究。
引用格式:刁旭炀,吴 凯,陈 都,等.基于混合注意力机制的软件缺陷预测方法[J].计算机测量与控制,2023,31(3):56-62,70.

征、基于依赖的 McCabe[8]特征、基于面向对象程序语言的 CK[9]特征以及基于多态、耦合的 MOOD[10]特征。

然而，由于静态度量元特征值是基于专家经验设计的统计值，存在有缺陷代码模块和无缺陷代码模块出现相同值导致无法区分的情况。ASTs[11]是一种基于源代码的特征树表示方式，蕴含着丰富的程序上下文信息。通过运用深度学习技术挖掘基于 ASTs 的语法语义特征，可以得到比静态度量元更具代表性的缺陷特征，从而构建出性能更好的缺陷预测模型。

为了充分运用程序上下文中潜在的语法语义信息，论文提出了一种基于混合注意力机制的软件缺陷预测方法。首先，使用词嵌入方法将特征序列表示为可被学习的多维向量，然后使用基于正余弦函数进行位置编码，接着运用多头注意力机制自学习每个位置在上下文中的语法语义信息，最后使用全局注意力机制提取整个程序模块的关键特征，从而构建出更加具有鉴别力的缺陷预测模型。论文选取了 7 个 Apache 开源项目作为数据集，与 5 种典型的基于静态度量元和程序语法语义学习的方法进行比较，实验结果表明论文提出的 DP-MHA 方法在 F1 上平均提升了 17.86%。

1　相关工作

在基于静态度量元的方法中，Chen[12]等人使用多目标决策优化算法对软件缺陷预测特征进行筛选；Huda[13]等人采用包裹和过滤式特征选择方法识别重要的缺陷特征；Okutan[14]等人使用贝叶斯网络来分配静态度量元与缺陷倾向性之间的影响概率；此外，Xu[15]等人提出了一种基于图的半监督缺陷预测方法来解决有标签数据不足和噪声问题。在基于程序语法语义学习的方法中，Wang[16]等人使用深度置信网络生产隐式的语法语义特征进行缺陷预测；Dam[17]等人建立了一种基于深度学习树的缺陷预测模型来尽可能多地保留 ASTs 的初始结构特征信息；Li [18]等人建立了嵌入静态度量元的卷积神经网络缺陷预测模型；此外，Phan[19]等人基于图神经网络从控制流图中挖掘软件的语法语义特征信息用来构建缺陷预测模型。

但是由于存在长期记忆依赖问题，基于 RNN 和 CNN 生产的语法语义信息可能存在信息的丢失。为了充分挖掘特征序列中每一位置的上下文信息，DP-MHA 采用多头注意力机制进行上下文自学习编码，然后将生产的隐式语法语义特征放入全局注意力机制网络中提取关键特征信息，用于缺陷预测模型的训练与预测。

2　DP-MHA 总体架构

本节对基于混合注意力机制的软件缺陷预测方法 DP-MHA 的总体架构进行了详细阐述，详见图 1。

首先，DP-MHA 将项目中的每个程序文件提取为 ASTs 树结构，然后从中选取关键节点并运用深度遍历方法获得序列向量，接着通过字典映射和词嵌入技术扩展为可学习的多维向量，其次运用正余弦函数进行位置编码，将编码后的向量放入多头注意力机制层，进一步挖掘每个位置的上下文语义，最后运用全局注意力机制提取程序文件中关键的语法语义特征，放入预测输出模型进行训练与预测。

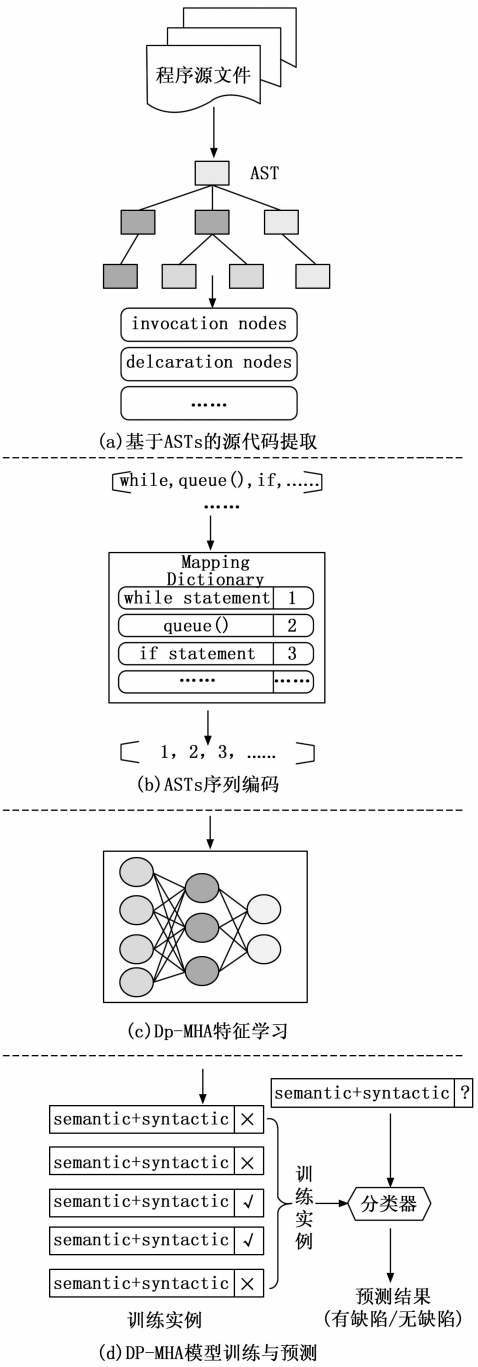

图 1　DP-MHA 总体架构

2.1　基于 ASTs 的源代码提取

为了将程序文件中的源代码用向量形式进行表征，需要选择合适颗粒度，如字符、单词或 ASTs 等形式对源程序进行特征提取。ASTs 是源代码语法结构的一种抽象表示，它以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的上下文语义信息。基于 ASTs 的表示形式能映射出源程序中的语法结构和语义信息，具有多维度

的特征信息可供缺陷预测模型进行学习，因此论文采用该形式对源代码程序进行抽象表征。

根据先前的研究，论文只提取 3 种类型节点作为 ASTs 树特征值。第一类为方法和类实例的创建节点，此类提取方法名称和类名称；第二类为声明节点，包括方法、类型、枚举等声明，此类提取它们的具体值；第三类为控制流节点，包括分支、循环、异常抛出等控制流节点用它们的类型值记录。所有被选择的 AST 节点值如图 2 所示。

MethodInvocation	ForStatement
SuperMethodInvocation	AssertStatement
PackageDeclaration	BreakStatement
InterfaceDeclaration	ContinueStatement
ClassDeclaration	ReturnStatement
MethodDeclaration	ThrowStatement
ConstructorDeclaration	SynchronizedStatement
VariableDeclarator	TryStatement
FormalParameter	SwitchStatement
BasicType	BlockStatement
CatchClauseParameter	StatementExpression
MemberReference	TryResource
SuperMemberReference	CatchClause
ReferenceType	CatchClauseParameter
IfStatement	SwitchStatementCase
WhileStatement	ForControl
DoStatement	EnhancedForControl

图 2 选取的 AST 节点类型

在本次实验中，论文采用基于 Python 的开源数据分析包 javalang 对 Java 源代码进行分析提取为 AST 抽象语法树，然后采用深度遍历的方法将提取的节点值转化为一个序列向量，详细描述见算法 1。

算法 1：将源程序文件提取为 ASTs 字符向量

输入：

F：源程序文件$\{f_1, f_2, \cdots, f_n\}$

R：选取的节点类型$\{r_1, r_2, \cdots, r_n\}$

输出：

S：字符向量$\{s_1, s_2, \cdots, s_n\}$

```
For i = 1→n do
    构建源程序 f_i 的 AST 抽象语法树 AST_i;
    For node in DFT(AST_i) then
        If node in R then
            Add node into ;
        End
    End
    Add s_i into S;
End
Return S;
```

2.2 ASTs 序列编码和类不平衡处理

ASTs 序列向量存储着大量程序模块的语法语义信息，两段代码模块可能具有相同的静态度量元特征值，但它们的 ASTs 结构存在差异，这使得基于 ASTs 序列可学习生成更具有辨别力的特征。为了让提取的 ASTs 序列向量具备可学习性，采用字典映射技术将序列中的节点映射为一个整型数字，假设节点的数量为 m，每个节点用一个整型数字表示，那么映射范围就是从 1 到 m。算法 2 为 ASTs 序列编码的详细描述，首先统计每个节点出现的频率，并按照频率从高到底排序，然后将 ASTs 字符向量映射成数字向量，接着需要将长短不一的数字向量统一为固定长度，少于固定长度的向量通过填充 0 来补齐长度，超出固定长度的向量将频率低的节点依次剔除直至长度与固定值一致，最终为了使得映射的数字向量具备可学习性，构建可被训练的高维词字典，采用词嵌入技术将每一个数字节点转化为一个多维向量。

算法 2：ASTs 序列编码

输入：

S：ASTs 字符向量$\{s_1, s_2, \cdots, s_n\}$

M：字符向量的长度

StrFreq：字符频率字典

StrtoInt：字符转整型字典

StrFreqList：字符频率列表

输出：

V：编码后的整型向量$\{v_1, v_2, \cdots, v_n\}$

```
1.  初始化 V、StrFreq、StrtoInt 和 StrFreqList
2.  For i = 1→n do
3.      For j=1→len(s_i) do
4.          if s_ij not in StrFreq.keys then
5.              StrFreq[s_ij]=0;
6.          End
7.          StrFreq[s_ij]+=1;
8.      End
9.  End
10. For key in StrFre.keys then
11.     StrFreqList.add((key, StrFreq[key]));
12. SortbyStrFreq(StrFreqList) ;//按频率降序排列
13. For i=1→len(SortbyStrFreq) do
14.     Str= SortbyStrFreq[i][0];
15.     StrtoInt[Str]=i;
16. End
17. For i = 1→n do
18.     For j=1→len() do
19.         v_ij = StrtoInt[s_ij];//将字符映射为整型,频率高的
            字符靠前
20.     End
21.     If len(v_i)<M then
22.         Add M-len(v_i) 0 s into v_i;
23.     End
24.     Else if len(v_i) > M then
25.         For k=1→len(v_i)-m do
26.             z=v_i.index(max(v_i));//找到频率最低的字符所
                对应的序号
27.             Del v_iz//删除该字符
28.         End
29.     End
30.     Add v_i into V;
```

31. End

32. Return V；

软件仓库中的缺陷数据通常是类不平衡的，其中有缺陷的程序文件往往只占据很小的一部分。直接采用原始数据集训练的到模型存在偏向于大多数（无缺陷）类的偏差。为了解决该问题，通常采用过采样或欠采样技术对数据集进行平衡处理。过采样通过复制少数类的实例来实现类间平衡，而欠采样则是将多数类中的实例进行剔除。考虑到欠采样或导致数据信息的丢失，从而导致欠拟合的情况，因此论文采用过采样技术将少数（有缺陷）类的实例进行复制，从而构造出两类平衡的数据集。

2.3 DP-MHA 神经网络构建

DP-MHA 神经网络架构如图 3 所示，主要包含节点嵌入层、位置编码层、多头注意力层、全局注意力层和预测输出层。

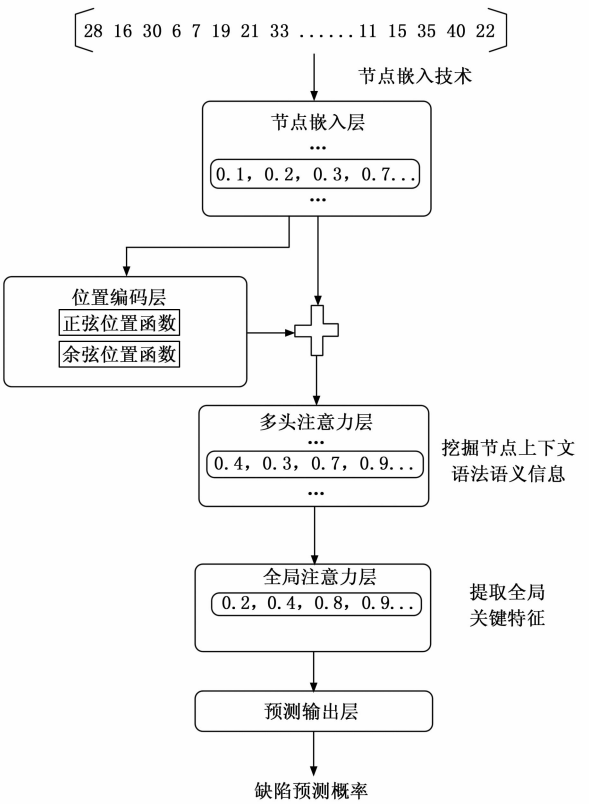

图 3　DP-MHA 神经网络架构

2.3.1　节点嵌入层

一维数字符号无法充分描述一个 AST 节点的上下文信息，论文采用词嵌入技术将每一个一维数字向量映射为一个高维向量，详细公式见式（1）所示。

$$F:M \rightarrow R^n \tag{1}$$

其中：M 代表一个一维 AST 节点向量，R^n 代表一个 n 维实数向量空间，F 代表一个参数化函数，能将 M 中的每个数字符号映射为一个 n 维向量。映射得到 n 维向量紧接着被放入位置编码层学习节点的位置信息。

运用词嵌入技术不仅可以将节点的语法语义表示形式限制在有限维度的空间内，节省网络训练的成本，避开维数灾难，而且节点之间的语法语义特征相似度也可以通过余弦距离公式来进行度量，极大地提升了网络学习语法语义特征的效率。

2.3.2　位置编码层

在 DP-MHA 模型中没有涉及到循环和卷积的神经网络，为了充分利用 AST 向量的序列特性，DP-MHA 将节点嵌入层输出的向量放入位置编码层进行相对位置关系的学习，位置编码后的向量维度与节点嵌入后的向量维度相同，以便两者相加得到最终包含相对位置关系的高维节点向量，作为多头注意力层的输入。

根据先前的研究，论文采用基于正弦和余弦函数进行位置编码，其计算公式如式（2）、式（3）所示：

$$PE_{(pos,2i)} = \sin(pos/10\ 000^{2i/d_{model}}) \tag{2}$$

$$PE_{(pos,2i+1)} = \cos(pos/10\ 000^{2i/d_{model}}) \tag{3}$$

其中：pos 代表第 pos 个向量，i 代表第 pos 个向量中的第 i 个维度，d_{model} 为节点嵌入的维度。即每个偶数位置的位置编码对应正弦函数，每个奇数位置的位置编码对应余弦函数，函数波长为 2π 到 $10\ 000 * 2\pi$ 的几何级数。对于任意固定偏移量 k，PE_{pos+k} 能用 PE_{pos} 的线性函数所表示，因此采用正余弦函数能较好的表示节点之间的相对位置关系。

2.3.3　多头注意力层

多头注意力层由 6（num _ blocks）个相同的神经网络层组成。每层由一个多头注意力机制和位置转化前向反馈网络组成，每个子层计算得到的输出与原始输入进行残差连接与正则化处理，其公式表达如式（4）所示：

$$LayerNormalization(x + SubLayerFun(x)) \tag{4}$$

其中：$SubLayerFun$ 为多头注意力机制和位置转换前向反馈网络两个子层的函数，将 $SubLayerFun$（x）与原始输入 x 相加得到残差连接并进行正则化处理，所有子层的输入输出向量维度与初始节点嵌入的维度相同。

多头注意力机制由 h 个基于点积的注意力函数组成，首先将节点嵌入的维度缩小 d_{model}/h 倍，使得神经网络的计算代价与带有整个节点维度的单个头注意力机制的计算代价相同，然后并行应用 h 个注意力函数，得到的输出进行全连接得到最终的上下文语法语义特征，具体机制架构详见图 4 所示。

每个点积注意力函数的输入为矩阵 $\boldsymbol{Q}$，$\boldsymbol{K}$，$\boldsymbol{V}$ 组成，计算公式如式（5）所示：

$$Attention(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{5}$$

其中：$\boldsymbol{Q}$ 为维度 batch _ size * vector _ len * d_k 的 queries 矩阵，$\boldsymbol{K}$ 为维度 batch _ size * vector _ len * d_k 的 keys 矩阵，$\boldsymbol{V}$ 为维度 batch _ size * vector _ len * d_v 的 values 矩阵。将 queries 和 keys 进行点积运算，计算每个位置节点在上下文 values 的权重，最终进行加权和计算得到每个位置节点经过上下文信息编码后的输出，具体函数架构详见图 5

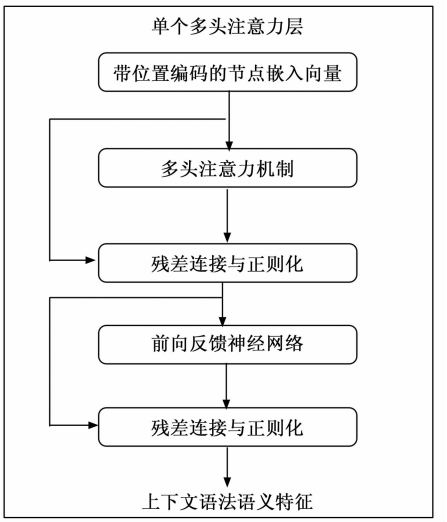

图 4 单个多头注意力层

所示。此外，由于当的值很大时，点积值往往会增长得很快，使得 softmax 函数出现梯度极端小的情况，为了解决梯度消失的问题，将点积结果值放缩$\frac{1}{\sqrt{d_k}}$倍。

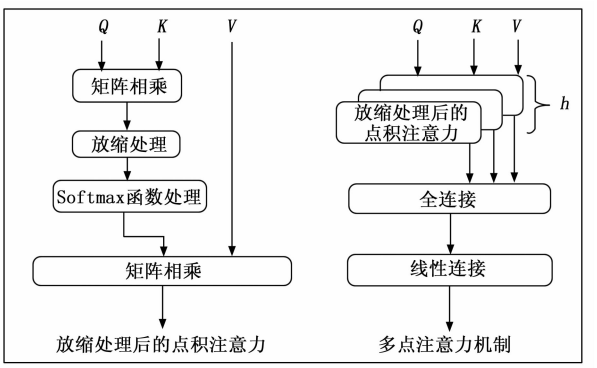

图 5 多头注意力机制的函数架构

在运用多头注意力机制后，DP-MHA 采用一个全连接的前向反馈神经网络进行位置转换，该网络层由两个线性函数和一个 ReLU 激活函数组成，其计算公式如式（6）所示：

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{6}$$

2.3.4 全局注意力层

从多头注意力层中可以得到一个序列所有位置节点的上下文语法语义信息，这些位置节点对整个序列的语法语义贡献是不同的。为了提取其中关键的语法语义特征，DP-MHA 采用了全局注意力函数，每个节点都会分配相应权重，进行加权和即可得到关键特征，其具体计算公式如式（7）～（9）所示：

$$u_{it} = \tanh(W_n h_{it} + b_n) \tag{7}$$

$$\alpha_{it} = \frac{\exp(u_{it}^T u_n)}{\sum_t \exp(u_{it}^T u_n)} \tag{8}$$

$$s_i = \sum_t \alpha_{it} h_{it} \tag{9}$$

其中，全局注意力机制首先将节点 h_{it} 放入一个多层感知器（MLP）来生成 u_{it} 作为节点的隐式特征表示，然后设定一个全局上下文向量 u_n，作为在每一个序列中搜寻关键节点信息的高阶表示向量，接着计算 u_{it} 与 u_n 的点积相似度并通过 softmax 函数进行归一化处理。最终计算所有节点的加权和作为学习得到的全局关键语法语义特征向量 s_i。

2.3.5 预测输出层

预测输出层由一层带有 sigmoid 激活函数的多层感知器组成，损失函数采用基于 softmax 的交叉信息熵，具体计算公式如式（10）、式（11）所示：

$$Y_i = \text{sigmoid}(s_i W_o + b_o) \tag{10}$$

$$Loss = -\sum_i \sum_a Y'_{ia} \log(softmax(Y_{ia})) \tag{11}$$

其中：Y_i 为维度为 num _ batch * 2 的预测值，Y'_{ia} 为第 i 个程序文件在 a 类上的真值，最终训练得到的预测模型可预测每个程序文件的缺陷概率。

3 实验设计

3.1 实验数据集

本文选取了 6 个公开 Java 项目作为实验数据集，所有数据集均可在 Github Apache 项目集中获得。每个项目选取两个版本，其中前置版本作为模型的训练集，后置版本作为衡量模型的测试集。在软件缺陷预测研究数据集中，静态度量元数据是一种公认的特征集，针对 Java 面向对象程序语言的特点，Jureczko[20] 等专家已进行相关研究并提取出了 20 个典型的静态度量元，其中包含了软件规模、代码复杂度以及与面向对象程序语言特性相关的特征。详细见表 1 所示。此外，程序的语法语义特征则由深度学习模型自动学习生成。

表 1 选取的 20 个静态度量元特征值

特征值	描述
WMC	类中方法的数量
DIT	类在继承树中的位置
NOC	类的直接后代数量
CBO	当一个类方法访问另一个类的服务时的增加值
RFC	响应对象消息而调用的方法数
LCOM	不能共享对实例变量的引用的方法数量
LCOM3	缺乏聚合的方法数量
NPM	一个类中所有声明为 public 的方法个数
DAM	私有和受保护属性数在属性总数的比率
MOA	类型为用户定义类的数据声明(类字段)的数量
MFA	类继承的方法数加上类成员方法可访问的方法数
CAM	每个方法中不同类型方法参数数量总和除以整个类中不同方法参数类型的数量与方法数量的乘积
IC	给定类耦合的父类数量
CBM	与所有继承方法耦合的新/重新定义方法的总数
AMC	JAVA 字节码个数
Ca	其他类使用特定类的数量
Ce	特定类使用其他类的数量
Max (CC)	同一类中方法的最大 McCabe 圈复杂度值
Avg (CC)	同一类中方法的平均 McCabe 圈复杂度值
LOC	代码容量大小

表 2 列举出了实验中项目数据集的相关信息，主要包括了项目名称、版本号、程序模块数量（以代码文件为单位）以及总体缺陷率。

表 2　Java 项目数据集信息

项目	版本	平均文件数	平均缺陷率/%
camel	1.4,1.6	918	18.1
jedit	4.0,4.1	309	25.0
poi	2.5,3.0	413	64.0
synapse	1.1,1.2	239	30.3
xalan	2.5,2.6	844	47.3
xerces	1.2,1.3	447	15.7

3.2　实验环境与评价指标

3.2.1　实验环境

本次实验的硬件配置为一台带有 Intel i7 酷睿处理器、8G 内存和 GTX1060 显卡的台式机，操作系统为 Windows10，使用到的开发语言为 Python 及其相关机器学习与深度学习模块 Scikit-learn 和 Tensorflow，开发工具为 pycharm。

DP-MHA 网络架构的输入向量长度（vector _ len）为 1 500，节点嵌入维度（d_{model}）为 32；多头注意力机制 query、key 与 value 向量的维度为 32，h 为 2，num _ blocks 为 6；全局注意力层的输出维度 $d_{\text{global\_attention}}$ 为 32，其余参数详见表 3。

表 3　DP-MHA 网络参数配置信息

参数	值
vector_len	1 500
d_{model}	32
d_q	32
d_k	32
d_v	32
h	2
num_blocks	6
$d_{\text{global\_attention}}$	32
batch_size	16
epoch	20
Loss function	softmax_cross_entropy
Optimizer	Adam

3.2.2　评价指标

本文从 F1 值角度对 DP-MHA 方法的软件缺陷预测性能进行衡量。通常情况下，当查全率（R）上升的同时，会降低查准率（P），F1 值综合了查准率（P）和查全率（R），对预测框架的综合性能进行了考量，越高代表着缺陷预测模型的稳定性越好，F1 值的范围为［0，1］，其计算公式如下：

$$P = \frac{N_{d\to d}}{N_{d\to d} + N_{c\to d}} \tag{12}$$

$$R = \frac{N_{d\to d}}{N_{d\to d} + N_{d\to c}} \tag{13}$$

$$F1 = \frac{2 * P * R}{P + R} \tag{14}$$

其中：c 为无缺陷程序文件，d 为有缺陷程序文件。$N_{d\to d}$ 表示预测和真实值都为有缺陷的数量；$N_{d\to d} + N_{c\to d}$ 为所有预测结果为有缺陷的数量；$N_{d\to d} + N_{d\to c}$ 为所有真实值为有缺陷的数量。

4　结果的分析和讨论

4.1　选取的经典缺陷预测方法

在实证研究阶段，论文选取了 5 个经典方法与 DP-MHA 方法进行比较，分别为基于静态度量元的 RF，基于无监督学习方法的 RBM＋RF 和 DBN＋RF 以及基于深度学习的 CNN 和 RNN，详见表 4 所示。

表 4　选取的 5 个经典缺陷预测方法

方法名称	说明
RF	基于 20 个静态度量元的随机森林方法
RBM＋RF	基于 RBM 提取程序语法语义特征的 RF 方法
DBN＋RF	基于 DBN 提取程序语法语义特征的 RF 方法
CNN	基于 textCNN 提取程序语法语义特征的方法
RNN	基于 Bi-LSTM 提取程序语法语义特征的方法

如表 5 所示，DP-MHA 方法相较于其他 5 种方法在 F1 值上平均提升了 17.86%。由此可见，基于混合注意力机制学习得到的上下文语法语义特征提升了缺陷预测模型的稳定性和准确度。

表 5　DP-MHA 相较于 5 种经典方法的 F1 值提升比率

	RF	RBM＋RF	DBN＋RF	CNN	RNN	平均
DP-MHA	16.6%	34.3%	26.4%	7.1%	4.9%	17.86%

4.2　与基于静态度量元方法的性能比较

为了说明 DP-MHA 方法相较于基于静态度量元方法的优越性，本文选取了基于 20 个静态度量元特征的 RF 方法进行对比分析。表 6 列出了 DP-MHA 与基于静态度量元特征方法 RF 之间的 F1 对比结果。相较于基于静态度量元的 RF 方法，DP-MHA 在 F1 的 W/T/L 上全部占优，在 F1 的平均值上提升了 16.6%。综合上述比较结果，可判得 DP-MHA 方法的软件缺陷预测模型稳定性能要优于基于静态度量元特征的缺陷预测方法。

表 6　DP-MHA 与基于静态度量元方法之间的 F1 值比较

项目	DP-MHA	RF
camel	**0.491**	0.396
jedit	**0.597**	0.555
poi	**0.730**	0.669
synapse	**0.597**	0.414
xalan	**0.654**	0.638
xerces	**0.263**	0.185
W/T/L		6/0/0
AVG	**0.555**	0.476

4.3 与基于无监督学习方法的性能比较

为了说明 DP-MHA 方法相较于基于无监督学习方法的优越性，本文选取了具有典型代表的两种无监督学习方法 RBM 和 DBN 来提取程序的语法语义特征，然后将两者学习到的特征分别放入 RF 分类器训练得到最终的缺陷预测模型。表 7 列出了 DP-MHA 与基于无监督学习方法 RBM＋RF 和 DBN＋RF 之间的 F1 值对比结果。DP-MHA 比 RBM＋RF 在 F1 值的 W/T/L 上赢了 6 次；相较于 DBN＋RF，也全部占优。此外从 F1 值的平均值来看，DP-MHA 相较于 RBM＋RF 和 DBN＋RF 方法分别提升了 34.3%、26.4%。综合以上比较结果，可判得在缺陷预测模型稳定性方面，DP-MHA 都要比基于无监督学习方法提取程序语法语义特征的方法来得更优。

表 7 DP-MHA 与基于无监督学习方法之间的 F1 值比较

项目	DP-MHA	RBM＋RF	DBN＋RF
camel	**0.491**	0.310	0.330
jedit	**0.597**	0.468	0.500
poi	**0.730**	0.639	0.652
synapse	**0.597**	0.303	0.360
xalan	**0.654**	0.628	0.623
xerces	**0.263**	0.128	0.167
W/T/L		6/0/0	6/0/0
AVG	**0.555**	0.413	0.439

4.4 与基于深度学习方法的性能比较

为了说明 DP-MHA 方法相较于基于其他深度学习方法的优越性，本文选取了具有典型代表的两种深度学习方法 CNN 和 RNN 来提取程序的语法语义特征并自动训练与预测。CNN 采用 1 维卷积核提取隐藏特征，RNN 采用 Bi-LSTM 的双向循环神经网络学习隐式语法语义特征。

表 8 列出了 DP-MHA 与基于深度学习方法 CNN 和 RNN 之间的 F1 值对比结果。DP-MHA 比 CNN 和 RNN 在 F1 值的 W/T/L 上分别都赢了 5 次。此外从 F1 值的平均值来看，DP-MHA 相较于 CNN 和 RNN 方法分别提升了 7.1%、4.9%。综合以上比较结果，可判得在缺陷预测模型稳定性方面，DP-MHA 都要比基于 CNN 和 RNN 的深度学习方法提取程序语法语义特征的方法来得更优。

表 8 DP-MHA 与基于深度学习方法之间的 F1 值比较

项目	DP-MHA	CNN	RNN
camel	0.491	0.473	**0.506**
jedit	**0.597**	0.596	0.595
poi	0.730	**0.734**	0.722
synapse	**0.597**	0.424	0.487
xalan	**0.654**	0.639	0.606
xerces	**0.263**	0.243	0.262
W/T/L		5/1/0	5/1/0
AVG	**0.555**	0.518	0.529

5 结束语

本文针对同项目软件缺陷预测问题，提出了一种基于混合注意力机制的软件缺陷预测方法 DP-MHA，运用节点嵌入与位置编码技术提取 AST 特征信息，放入多头注意力层中学习节点的上下文语法语义信息，然后使用全局注意力机制提取关键特征，放入预测输出层中进行训练与预测，该方法学习得到的程序语法语义特征有效提升了软件缺陷预测模型的性能。该方法仍有一些工作值得后续开展研究，具体而言为：(1) 考虑跨项目软件缺陷预测的应用场景，并验证该方法是否能提升预测性能；(2) 在实际工程项目[21-22]数据集中进一步验证该缺陷预测方法的效果与性能。

参考文献：

[1] RODRiGUEZ-PéREZ G, NADRI R, et al. Perceived diversity in software engineering: a systematic literature review [J]. Empirical Software Engineering, 2021, 26 (5): 1 - 38.

[2] KOTTE A, QYSER D, et al. A Survey of different machine learning models for software defect testing [J]. European Journal of Molecular and Clinical Medicine, 2021, 7 (4): 3256 - 3268.

[3] 陈 翔，顾 庆，刘望舒，等. 静态软件缺陷预测方法研究 [J]. 软件学报，2016，10 (1)：25.

[4] DALLA PALMA S, DI NUCCI D, PALOMBA F, et al. Within-project defect prediction of infrastructure-as-code using product and process metrics [J]. IEEE Transactions on Software Engineering, 2021, 48 (6): 2086 - 2104.

[5] ASANO T, TSUNODA M, TODA K, et al. Using bandit algorithms for project selection in cross-project defect prediction [C] //2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2021: 649 - 653.

[6] 陈 翔，王莉萍，顾 庆，等. 跨项目软件缺陷预测方法研究综述 [J]. 计算机学报，2018，41 (1)：254 - 274.

[7] ABDULKAREEM S A, ABBOUD A J. Evaluating Python, C++, JavaScript and Java Programming Languages Based on Software Complexity Calculator (Halstead Metrics) [C] // IOP Conference Series: Materials Science and Engineering, 2021: 012046.

[8] PEITEK N, APEL S, PARNIN C, et al. Program comprehension and code complexity metrics: An fmri study [C] //2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021: 524 - 536.

[9] HAMDI O, OUNI A, ALOMAR E A, et al. An empirical study on the impact of refactoring on quality metrics in android applications [C] //2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft), 2021: 28 - 39.

(下转第 70 页)