软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(12):5382−5396 [doi: 10.13328/j.cnki.jos.007067] [CSTR: 32375.14.jos.007067]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于生成对抗策略的代码搜索*

张祥平[1,2], 刘建勋[1,2], 扈海泽[1,2], 刘 益[1,2]

[1](服务计算与软件服务新技术湖南省重点实验室 (湖南科技大学), 湖南 湘潭 411201)

[2](湖南科技大学 计算机科学与工程学院, 湖南 湘潭 411201)

通信作者: 刘建勋, E-mail: liujx@hnust.cn

摘 要: 基于深度学习的代码搜索方法通过计算代码与描述语句各自表征的相似程度, 实现代码搜索任务. 然而此类方法并未考虑代码和描述语句之间真实存在的相关性概率分布. 针对此问题, 将经典概率模型中代码和描述语句的相关性概率分布与向量空间模型中特征提取相结合, 提出基于生成对抗策略的代码搜索方法. 所提方法首先设计代码和描述语句的特征编码器用于特征提取. 接着采用生成对抗策略, 将代码和描述语句之间的概率分布应用于生成器和判别器的交替训练, 同时实现对代码编码器和描述语句编码器的优化, 生成高质量的代码表征和描述语句表征用于代码搜索任务. 最后在公开的数据集上进行实验验证, 结果表明所提出的方法相比于 DeepCS 方法在 *Recall*@10, *MRR*@10 和 *NDCG*@10 指标上分别提升 8.4%、32.5% 和 24.3%.

关键词: 代码搜索; 生成对抗策略; 代码表征; 近似采样

中图法分类号: TP311

中文引用格式: 张祥平, 刘建勋, 扈海泽, 刘益. 基于生成对抗策略的代码搜索. 软件学报, 2024, 35(12): 5382–5396. http://www.jos.org.cn/1000-9825/7067.htm

英文引用格式: Zhang XP, Liu JX, Hu HZ, Liu Y. Code Search with Generative Adversarial Game. Ruan Jian Xue Bao/Journal of Software, 2024, 35(12): 5382–5396 (in Chinese). http://www.jos.org.cn/1000-9825/7067.htm

Code Search with Generative Adversarial Game

ZHANG Xiang-Ping[1,2], LIU Jian-Xun[1,2], HU Hai-Ze[1,2], LIU Yi[1,2]

[1](Key Lab for Services Computing and Novel Software Technology (Hunan University of Science and Technology), Xiangtan 411201, China)

[2](School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China)

Abstract: The code search method based on deep learning realizes the code search task by calculating the similarity of the corresponding representation of the code and the description statement. However, this manner does not consider the real probability distribution of relevance between the code and the description. To solve this problem, this study proposes a code search method based on a generative adversarial game that combines the correlation between the code and the description in the classical probability model with the feature extraction in the vector space model. Then the generative adversarial game is adopted to apply the probability distribution between the code and the description to the alternate training of the generator and discriminator. Meanwhile, the code encoder and the description encoder are optimized, and high-quality code representation and description statement representation are generated for the code search task. Finally, experimental verification is carried out on the public dataset, and the results show that the proposed method improves the *Recall*@10, *MRR*@10, and *NDCG*@10 metrics by 8.4%, 32.5%, and 24.3% respectively compared to the DeepCS method.

Key words: code search; generative adversarial game; code representation; approximate sampling

代码搜索作为软件开发人员日常开发过程中最常见的活动之一, 在提高软件开发效率上有着至关重要的作用[1]. 研究表明, 有 59% 的开发人员每天都进行代码搜索[2], 开发人员通过在搜索引擎中键入所需要实现的软件功能,

* 基金项目: 国家自然科学基金 (61872139)
收稿时间: 2023-01-25; 修改时间: 2023-04-17; 采用时间: 2023-10-10; jos 在线出版时间: 2024-02-05
CNKI 网络首发时间: 2024-02-09

从代码搜索引擎所返回的搜索结果中选择满足其开发需求的代码内容进行使用, 在提高代码复用性的同时, 显著提升开发效率.

现有的代码搜索研究可以分为两类, 一类是基于信息检索技术的代码搜索. 这类方法将代码数据视为普通文本, 对比用户给出的查询语句与代码库中代码片段两者之间的关键字匹配程度返回搜索结果. 如, Hill 等人[3]将源代码的上下文信息与查询描述相结合, 计算两者之间的相关性来提高搜索性能. Lv 等人[4]提出了 CodeHow, 它把用户输入的查询语句与代码片段 API 进行关键字匹配, 并使用布尔模型进行拓展来进行代码搜索. 这类基于信息检索技术的代码搜索工作的局限性在于仅考虑了查询语句与代码片段的表面符号信息, 缺乏对查询语句和代码片段深层次的语义内容分析, 尤其是查询语句与代码片段之间存在的语义鸿沟问题难以解决.

另一类是基于深度学习技术的代码搜索. 这类工作的通常做法是对查询语句和代码片段建模获得对应语义向量, 在相同的代码语义空间中比较查询语句与代码片段两者之间的相似度, 尝试解决两者之间存在的语义鸿沟问题, 这种特征提取模型也被称为向量空间模型[5–8]. 此类方法的基本流程如图 1 所示, 首先从数据集中抽取代码及正描述语句和负描述语句, 分别使用不同的编码器转换为特征向量之后, 在代码语义空间中衡量代码与描述语句之间的距离, 通过不断训练获得高质量的代码表征和描述语句表征用于代码搜索任务. Gu 等人[9]提出 DeepCS 模型, 首次将深度学习技术应用到代码搜索任务中. 该方法将代码片段所包含的 3 个部分: 方法名、API 调用序列和词汇单元, 与该代码对应的描述进行联合嵌入表征, 使代码片段和描述文档映射到相同的代码-描述语句空间中, 通过计算代码片段与其正样本和负样本之间的三元损失函数进行模型训练. 之后在测试过程中, 直接在所有代码表征向量池中选择与查询语句最匹配的结果作为搜索结果返回. Shuai 等人[10]在 DeepCS 模型的基础上使用卷积神经网络和共同关注机制进一步考虑了代码片段和查询语句之间的相互依赖关系, 该方法同样是通过使用三元损失函数实现对代码和查询语句的区分. Fang 等人[11]进一步使用自注意力机制综合考虑代码片段中 3 个部分之间的关联性, 该方法采用最先进的 Transformer 编码器构建代码片段与查询语句之间的语义联系, 同样采用三元损失函数进行模型的训练.

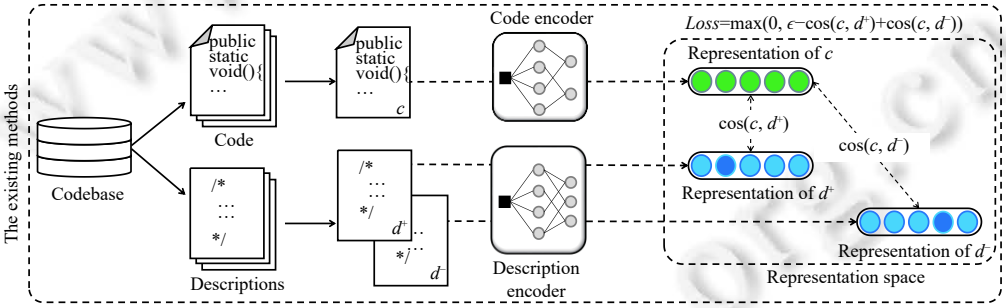

图 1 现有基于深度学习的代码搜索任务训练方式

上述方法在提升代码搜索精度方面取得了不错的效果, 然而它们仍然存在一些不足. 上述方法通过计算代码片段与正负查询语句之间的相似度来获得代码搜索结果. 这种搜索方式高度依赖神经网络的特征提取能力, 忽视了对查询语句和代码片段之间真实存在的相关性概率分布的考虑[12]. 实际上“代码-描述语句”数据集中真实的数据分布情况也应作为数据特征考虑进代码搜索任务中. 为此, 本文聚焦于代码搜索问题, 将概率检索模型中代码和描述语句的相关性概率分布与向量空间模型中特征提取相结合, 提出了基于生成对抗策略的代码搜索方法. 该方法首先设计代码和描述语句的特征编码器用于特征提取, 接着利用生成对抗策略, 从数据集中获得描述语句和代码之间的概率分布, 在生成器和判别器的交替训练过程中同时实现对代码编码器和描述语句编码器的训练, 使两者能够生成高质量的代码表征和描述语句表征用于代码搜索任务. 最后在公开的数据集上进行实验验证, 实验结果表明, 本文所提出的方法能够有效提升代码搜索任务的效果.

概括来说, 本文工作主要有以下几点贡献.

(1) 本文首次提出了基于生成对抗策略的代码搜索方法. 该方法将概率检索模型中代码和描述语句的相关性与向量空间模型中特征提取相结合, 在使用极大极小策略对生成器和判别器完成交互优化的同时实现对代码和查询语句之间细粒度特征的提取. 实验结果表明本文所提出方法相比于 DeepCS 方法在 *Recall*@10, *MRR*@10 和 *NDCG*@10 指标上分别提升了 8.4%、32.5% 和 24.3%.

(2) 本文首次提出使用近似采样策略模拟生成器生成代码片段的过程, 该方式能够获得数据集中真实存在的代码-描述语句的统计分布信息并应用于判别器的训练. 同时本文首次提出温度调节机制, 用于调节生成器对所生成代码样本的关注程度. 通过实验验证了该机制的有效性, 实现结果表明: 温度越低, 生成器越关注最接近真实样本的负样本, 能够显著提高代码搜索的效果.

(3) 本文考察了在代码特征提取过程中不同的特征对代码搜索任务效果的影响, 实验结果表明: 不同的代码结构对代码搜索任务均有影响, 不同特征的组合特征能够获得比单一特征更好的搜索结果.

本文第 1 节介绍代码搜索任务的相关工作. 第 2 节详细阐述基于生成对抗策略的代码搜索方法. 第 3 节详细介绍实验所使用数据集与实验设置. 第 4 节对实验进行分析与讨论. 最后一节总结全文并展望.

1 代码搜索相关工作

早期的代码搜索研究是以信息检索技术为基础, 常见做法是以关键字匹配方法完成对查询语句和代码片段之间的匹配. 这些方法主要关注代码文本层面的相似性, 而忽略了代码的语义或者功能层面上的相似性. 因此,搜索所返回的结果往往难以达到预期的准确性和相关性[13]. Hill 等人[3]将源代码的上下文信息与查询语句相结合, 计算两者之间的相关性来提高搜索性能, 但在特定数据集上的召回率仅有 38%. 导致这一现象的最主要原因在于此类基于关键字匹配的方法无法解决代码和查询语句之间的语义鸿沟问题.

随着深度学习技术的快速发展, 研究者们开始探索并将其应用于代码搜索任务, 以解决代码与描述语句之间存在的语义鸿沟问题. 通过对源代码进行统计分析, 发现源代码与自然语言有着相似的统计学规律. 将自然语言处理领域中的文本处理技术与代码搜索任务相结合, 常见的方法是将描述语句和代码片段映射为同一个向量空间中的连续向量, 进而评估两者之间的相似程度. 这种方法的效果依赖于向量所蕴含的上下文隐含语义信息的丰富程度[14–16]. Gu 等人[9]首次将神经网络技术引入到代码搜索任务中, 提出了 DeepCS 方法.该方法将代码片段和代码描述语句映射到同一个高维度的向量空间中, 使得描述语句和代码具有相似的向量表示. 通过计算描述语句向量和代码向量之间的余弦相似度来搜索与描述语句相匹配的代码片段. Shuai 等人[10]在 DeepCS 方法的基础上使用卷积神经网络和共同关注机制进一步考虑了代码片段和查询语句之间的相互依赖关系, 该方法同样是通过使用三元损失函数实现对代码和查询语句的区分. Fang 等人[11]与文献 [9] 采取了相似的代码预处理策略, 将代码片段解析为方法名、API 以及词汇单元 3 个部分, 将这 3 个部分送入嵌入层中转换为对应的特征向量, 并在特征向量上使用自注意力机制, 获得各个部分内部元素的注意力分数, 最后将注意力分数附加在特征向量中. 对于代码的描述文本也用同样的方式转换为特征向量, 通过计算代码特征向量与代码描述文本特征向量之间的余弦相似度, 返回代码搜索结果. Chen 等人[17]提出了基于语义的 Java 方法搜索方法, 该方法增加对代码其他特性的支持, 使得其搜索能力得到提升.

有许多代码搜索工作也关注于代码所具有的独特属性, 如可以将代码转换为对应的抽象语法树, 或者程序依赖图, 并在其上运用深度学习技术获得这些结构所包含的代码信息. 如基于程序依赖图的代码搜索方法通常将项目中的代码转换为代码图, 使用图神经网络技术提取代码图中的节点特征信息. 之后根据代码图解析查询语句, 将查询语句的关键词与代码图中的候选节点进行匹配, 生成子图并推荐搜索结果. Gu 等人[18]首先设计了一种新的树型结构, 用于化简代码对应的抽象语法树, 之后在化简后的抽象语法树上引入了两种树序列化方法, 将抽象语法树转换为线性的词汇单元序列, 最后采用了伪孪生网络架构,同时处理代码序列和树序列, 实现多模态的代码搜索任务. Meng 等人[19]提出了一种抽象语法树解析方法, 能够将代码的词法特征和结构特征融入到代码表征向量中, 同时该工作使用自注意力机制, 突出了代码中关键词的作用, 进而提高了代码搜索的效果. Xu 等人[20]提出了基于注

意力机制的代码搜索模型 TabCS, 通过结合代码的文本特征、代码对应的抽象语法树结构特征来提高代码搜索任务的效率和准确性. 在该工作中使用两个阶段的注意力网络结构来提取代码以及查询语句的语义信息. 第 1 阶段利用注意力机制从代码和查询中提取语义, 第 2 阶段通过共注意力机制捕捉两者的语义相关性. 该模型在两个大规模数据集上进行评估, TabCS 模型在性能上显著优于现有的深度学习模型. Zou 等人[21]提出了一种基于图嵌入技术的代码搜索方法, 该方法首先软件项目源代码转换为代码图, 之后使用图嵌入技术表示代码图中的每个代码元素. 接着使用自然语言搜索代码图, 返回由相关代码元素及其关联关系组成的子图作为搜索结果. 这种方法能够有效地表示软件代码图的深层结构信息.

以上这些方法的研究对象均为代码或者描述语句本身, 依赖于所使用模型的特征提取能力, 尽可能多地获得两者的特征来提高代码搜索任务的效果[22]. 然而这些方法都缺乏对代码数据集中潜在数据分布信息的利用. 在本文中, 将概率检索模型中代码和描述语句的相关性概率分布与向量空间模型中特征提取相结合, 提出了基于生成对抗策略的代码搜索方法, 用于提高代码搜索的效果.

2 方法介绍

本文以向量空间模型为基础, 结合概率检索模型中查询语句与代码片段的相关性概率分布, 提出了一种基于生成对抗策略的代码搜索方法, 该方法由 3 部分组成.

首先, 构建代码编码器和代码描述编码器分别用于生成代码特征的表征向量和代码对应的描述语句的表征向量. 对于代码而言, 考虑到其本身所具有的性质, 将代码划分为 3 个部分: 方法名、API 调用序列和代码中所包含的词汇单元 (Token) 符号, 并对这 3 个部分分别构建特征编码器[9]. 其中, 对于方法名和 API 调用序列, 使用循环神经网络 (RNN) 进行特征提取[23]; 对代码中所包含的词汇单元符号使用多层感知机进行特征提取. 在获得的特征向量上使用最大池化操作, 获得最显著的特征信息用于三者信息的特征融合, 最后生成整个代码的表征向量 c. 对于代码描述语言, 采用循环神经网络进行特征提取, 并且同样在其获得的特征向量上使用最大池化操作获得最显著的特征信息作为其整体的表征向量 d.

其次, 引入生成对抗策略实现对代码编码器和代码描述编码器的训练. 其核心思想是构造一个生成器和一个判别器, 在两者的交替训练过程中完成对代码和描述语句的特征提取. 生成器的目的是生成最接近真实数据的虚假数据用于欺骗判别器. 判别器是一个简单的二分类器, 其目的是尽可能正确地区分真实数据和虚假数据. 在两者的训练过程中引入了温度调节机制和奖励机制, 迫使生成器关注于更接近真实样本的负样本, 促使生成器生成更加真实的数据欺骗判别器.

最后, 当生成器和判别器的训练趋于稳定的同时, 代码编码器和代码描述编码器朝着全面提取代码和描述语句特征的方向进行优化. 两者所生成的代码表征向量和代码描述表征向量能够高质量地表示代码和描述文档的特征, 仅需要计算两个向量的余弦相似度即可获得两者的相似度, 进一步用于代码搜索任务.

2.1 代码及描述语句的特征提取

本节介绍如何在向量空间模型中构建代码编码器和描述语句编码器分别用于生成代码表征和描述语句表征. 代码与描述语句最明显的区别在于代码不仅是纯粹的文本数据, 其还包含了一系列人为规定的语法结构特征. 本文参考已有工作, 重点考虑了代码的 3 个方面的特征, 分别是方法名、API 调用序列和词汇单元序列[24]. 本文将这些特征提取出来, 再将其转换为特征向量进行模型训练. 以下是对 3 个部分的简要介绍.

首先是代码方法名的提取. 代码方法名的设置是要求其能够充分表达代码所实现的功能, 因此方法名可以认为是对代码功能的简要总结. 如图 2 所示, 对于采用驼峰法命名的方法名, 本文使用 Gensim 库中的分词工具对其进行切割, 从而将一个方法名拆分为 3 个字符, 并记方法名为 M.

其次是对代码片段中的 API 调用序列的提取. API 指的是代码中不同组成部分衔接的约定. 本文使用 Eclipse 中自带的 Java 编译器遍历代码所对应的抽象语法树, 并参照文献 [24] 中所采用的规则生成 API 的调用序列. 如图 2 所示, 从该代码中能够提取出长度为 2 的 API 序列 A.

最后是代码中的词汇单元部分. 其所表示的是代码中所出现的字符信息. 本文采用 Gensim 库中的分词工具对整个代码片段进行切割, 并且移除了包含信息量少的停用词以及重复出现的词汇单元. 之后获得了代码的词汇单元序列 T . 同时本文对代码的描述语句采用了与提取代码词汇单元序列相同的操作, 获得了描述语句词汇单元序列 D .

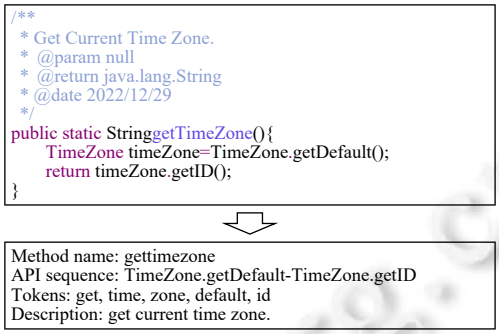

图 2　从 Java 代码中抽取特征的示例

在获得了不同特征信息之后, 本文构建了代码编码器和代码描述语句编码器分别用于获得代码表征和描述语句表征, 将两者用于计算代码和描述语句之间的相似程度. 现有的做法通常是将代码和查询语句均视为符号序列, 使用序列神经网络将两者转换为对应的表征向量之后进行相似度的计算. 在本文中, 考虑到代码的 3 个部分以及描述语句仍然具有不同的统计规律, 因此分别对每个部分单独设计一个神经网络模块用于特征提取.

首先是针对代码方法名采用 RNN 模型对其进行特征提取, 如图 3 方法名提取部分, 其特征抽取过程可以用公式 (1) 和公式 (2) 表示:

$$\boldsymbol{h}_i = \tanh\left(\boldsymbol{W}^M\left[\boldsymbol{h}_{i-1};\boldsymbol{e}_{M_i}\right]\right),\ i \in [1, N_M] \tag{1}$$

$$\boldsymbol{m} = \text{maxpooling}\left(\left[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{N_M}\right]\right) \tag{2}$$

其中, N_M 表示方法名拆分后的符号数量, $\boldsymbol{e}_{M_i}$ 表示方法名序列中第 i 个符号 M_i 的所对应的向量, $\boldsymbol{W}^M$ 表示该 RNN 模型中的所有参数, tanh 表示使用正切函数作为激活函数, $\boldsymbol{h}_i$ 表示 RNN 模型输出的第 i 个隐藏状态, maxpooling 表示对这些向量采用最大池化操作, $\boldsymbol{m}$ 表示所生成的方法名特征向量.

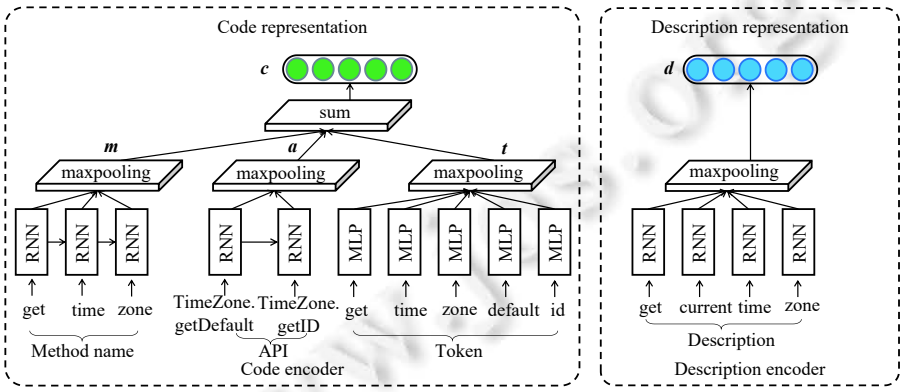

图 3　代码编码器和描述语句编码器

对 API 序列同样采用 RNN 神经网络对其进行特征提取, 如图 3 中 API 序列提取部分所示, 其提取过程可以用公式 (3) 和公式 (4) 表示:

$$\boldsymbol{h}_i = \tanh\left(\boldsymbol{W}^A\left[\boldsymbol{h}_{i-1};\boldsymbol{e}_{A_i}\right]\right),\ i \in [1, N_A] \tag{3}$$

$$\boldsymbol{a} = \text{maxpooling}([\boldsymbol{h}_1,\ldots,\boldsymbol{h}_{N_A}]) \tag{4}$$

其中, N_A 表示 API 序列的数量, $\boldsymbol{e}_{A_i}$ 表示 API 序列中第 i 个序列符号 A_i 的所对应的向量, $\boldsymbol{W}^A$ 表示该 RNN 模型中的所有参数, $\boldsymbol{a}$ 表示所生成的 API 序列特征向量.

对代码中的词汇单元序列, 由于已对词汇单元符号进行了去除停用词和去重处理, 其内部结构信息存在丢失, 因此本文采用多层感知机对其进行特征抽取, 其提取过程可以用公式 (5) 和公式 (6) 表示:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}^T\boldsymbol{e}_{T_i}),\ i \in [1,N_T] \tag{5}$$

$$\boldsymbol{t} = \text{maxpooling}([\boldsymbol{h}_1,\ldots,\boldsymbol{h}_{N_A}]) \tag{6}$$

其中, N_T 表示词汇单元的数量, $\boldsymbol{e}_{T_i}$ 表示 API 序列中第 i 个词汇单元 T_i 的所对应的向量, $\boldsymbol{W}^T$ 表示该 MLP 模型中的所有参数, $\boldsymbol{t}$ 表示所生成的词汇单元特征向量.

在获得了代码 3 个部分的特征向量之后, 将三者按维度对应位置相加, 获得融合的代码表征向量, 并在融合特征向量上使用一个全连接层, 获得最终的代码表征, 如公式 (7) 所示:

$$\boldsymbol{c} = \tanh(\boldsymbol{W}^C[\boldsymbol{m}+\boldsymbol{a}+\boldsymbol{t}]) \tag{7}$$

其中, $\boldsymbol{W}^C$ 表示在该全连接层中的所有参数.

而对于代码描述文本, 本文中使用了简单 RNN 神经网络进行特征提取. 如图 3 描述文本编码器部分, 其特征生成过程可以用公式 (8) 和公式 (9) 表示:

$$\boldsymbol{h}_i = \tanh(\boldsymbol{W}^D[\boldsymbol{h}_{i-1};\boldsymbol{e}_{D_i}]),\ i \in [1,N_D] \tag{8}$$

$$\boldsymbol{d} = \text{maxpooling}([\boldsymbol{h}_1,\ldots,\boldsymbol{h}_{N_D}]) \tag{9}$$

其中, N_D 表示描述语句的长度, $\boldsymbol{e}_{D_i}$ 表示描述语句中第 i 个词汇单元 D_i 的所对应的向量, $\boldsymbol{W}^D$ 表示该模型中的所有参数, $\boldsymbol{d}$ 表示所生成的描述文本特征向量.

至此, 我们获得了代码和对应描述文本的表征向量. 需要注意的是, 符号向量 $\boldsymbol{e}_{M_i}$ 、 $\boldsymbol{e}_{A_i}$ 、 $\boldsymbol{e}_{T_i}$ 和 $\boldsymbol{e}_{D_i}$ 的初始值是 PyTorch 库中随机初始化获得, 之后根据本文所提出的基于生成对抗策略的代码搜索方法对这些表征向量进行更新优化, 直至尽可能全面地表征代码和描述文本的特征信息.

2.2 基于生成对抗策略的代码搜索方法

现有基于神经网络的代码搜索方法[9–11]的训练方式均采用如图 1 所展示的模型训练方式获得代码表征和描述语句表征, 通过计算两者的相似度来完成代码搜索任务. 首先从代码库中抽取代码片段和代码描述, 对于每一个代码片段 $\boldsymbol{c}$, 从代码描述中选取与之对应的一个正样本 $\boldsymbol{d}^+$ 和一个负样本 $\boldsymbol{d}^-$, 作为神经网络的三元输入 $(\boldsymbol{c}, \boldsymbol{d}^+, \boldsymbol{d}^-)$. 接着, 使用代码编码器和描述语句编码器对三者进行建模, 获得所对应的表征向量. 最后, 根据三者的表征向量计算三者之间的三元损失函数从而完成表征模型的训练. 上述代码搜索模型的训练过程中所使用的优化目标为三元损失函数 $Loss = \max(0, \epsilon - \cos(\boldsymbol{c},\boldsymbol{d}^+) + \cos(\boldsymbol{c},\boldsymbol{d}^-))$, 其中 ϵ 为人为设定的阈值. 此类代码搜索方法存在以下两点不足: (1) 这些方法没有考虑数据集中数据分布对代码搜索任务的影响. (2) 三元损失函数在训练过程中的收敛速度慢, 同时由于需要人工选择 ϵ 的取值, 不合适的 ϵ 容易使得模型过拟合.

为了解决上述问题, 本文提出了一个基于生成对抗策略的代码搜索方法. 如图 4 所示. 该方法与现有的基于深度学习的代码搜索方法的不同在于本文方法增加了两个模块: 生成器和判别器. 生成器生成虚假的代码数据与描述文本样本对, 判别器从中判别真假样本. 该过程采用极大极小策略对生成器和判别器交替优化实现[25–28]. 其中, 由于本文所处理的数据均为文本数据, 其离散性质使得生成器无法直接生成原始的代码序列, 因此本文提出从代码数据集中进行采样来模拟生成器生成数据的过程, 使生成器从代码数据集中选择最接近真实样本的数据供判别器训练. 判别器使用简单的二分类器进行真假样本的判断, 该方式使得代码编码器和描述语句编码器在交替优化过程中更加关注代码片段和描述语言之间的细微差别, 进而获得高质量的代码表征用于代码搜索任务. 本文提出的基于生成对抗策略的代码搜索方法可以用算法一进行描述.

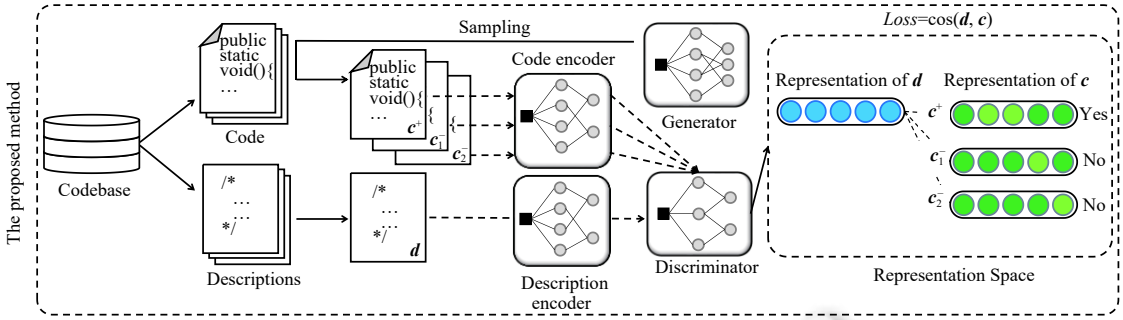

图 4 基于生成对抗策略的代码搜索方法

算法 1. 基于生成对抗策略的代码搜索方法.

输入: 生成器 $p_\theta(c|d_n,r)$; 判别器 $f_\phi(d,c)$; 训练数据 (D,C);

输出: 生成器 $p_\theta(c|d_n,r)$; 判别器 $f_\phi(d,c)$.

1. $p_\theta(c|d_n,r) \leftarrow \theta; f_\phi(d,c) \leftarrow \phi$ // 使用随机生成权重 θ 和 ϕ 初始化 $p_\theta(c|d_n,r)$ 和 $f_\phi(d,c)$
2. **for** *global_epoch* **do**
3. **for** *d_epoch* **do**
4. $c^- \sim p_\theta(c|d_n,r)$ // 使用生成器 $G=p_\theta(c|d_n,r)$ 生成负样本 c^-
5. $f_\phi(d,c) \leftarrow [c^+,c^-]$ // 将正样本和负样本结合, 通过公式 (12) 训练判别器
6. **end for**
7. **for** *g_epoch* **do**
8. $c^* \sim p_\theta(c|d_n,r)$ // 使用生成器 $G=p_\theta(c|d_n,r)$ 生成样本 c^*
9. $p_\theta(c|d_n,r) \leftarrow \nabla_\theta J^G(d_n)$ // 使用策略梯度 (公式 (17)) 更新生成器参数 θ
10. **end for**
11. **end for**

生成器: 在本文中其目的为生成一个概率分布 $p_\theta(c|d,r)$, 使该概率分布尽可能地接近描述语句 D 和代码片段 C 数据集中两者的真实分布 $p_{\text{true}}(c|d,r)$, 使得生成器能够生成判别器难以判别的样本数据. 其中, θ 表示生成器中所有参数, r 表示 c 和 d 两者之间的相关程度.

判别器: 其目的在于构建一个判别模型 $f_\phi(d,c)$, 用于区分输入样本 (d,c) 是数据集中真实存在的数据还是生成器所生成的虚假数据. 本文中所使用的判别器为一个二分类器, 其具体构造方式见下文.

统一优化目标: 本文通过极小极大策略统一优化以上两个模块[26]. 首先, 对于判别器 $f_\phi(d,c)$ 要最大限度地提高其对数似然性以便正确识别真实样本和生成样本. 具体做法是将真实样本 (记为正样本) 和生成器所生成的样本 (记为负样本) 共同作为训练数据供判别器 $f_\phi(d,c)$ 进行训练, 此过程中要求判别器能够尽可能正确地区分样本的真假. 其次, 对于生成器 $p_\theta(c|d,r)$ 通过最小化其对数似然性, 使生成器尽可能地生成接近正样本的负样本数据, 用于欺骗判别器. 以上两种模型的优化目标可以统一表示为公式 (10):

$$J^{G^*,D^*}=\min_\theta\max_\phi\sum_{n=1}^{N}(\mathbb{E}_{c\sim p_{\text{true}}(c|d_n,r)}[\log D(c|d_n)]+\mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\log(1-D(c|d_n))]) \tag{10}$$

其中, $\mathbb{E}_{c\sim p_{\text{true}}(c|d_n,r)}$ 表示对所有正样本 (d,c) 的期望值. $\mathbb{E}_{c\sim p_\theta(c|d_n,r)}$ 表示生成器对所有输入数据对的期望值. $D(c|d_n)$ 表示判别器对输入样本 (d,c) 是否为真的判断概率, 其计算方式如下:

$$D(c|d_n)=\sigma(f_\phi(d_n,c)) \tag{11}$$

其中, σ 为 Sigmoid 函数.

至此, 基于生成对抗策略的代码搜索基本流程介绍完毕, 接下来的内容将结合代码搜索任务, 分别介绍如何优化生成器和判别器的目标函数, 具体构建适用于代码搜索任务的生成器和判别器.

判别器优化目标: 判别器的目标用于最大限度地区分正样本和负样本, 具体做法是通过最大化其对数似然性. 联立公式 (10) 和公式 (11), 获得判别器的优化目标函数为公式 (12):

$$\phi^* = \underset{\phi}{\operatorname{argmax}} \sum_{n=1}^{N} (\mathbb{E}_{c\sim p_{\text{true}}(c|d_n,r)}[\log(\sigma(f_\phi(d,c)))] + \mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\log(1-\sigma(f_\phi(d,c)))]) \tag{12}$$

公式 (12) 的第 2 项表示, 如果描述语句 d 与代码片段 c 越相似, 其优化函数值也越大, 此项能够使代码编码器和描述语句编码器更加关注于代码和描述语句之间的细微差别, 从而生成高质量的代码表征和描述语句表征用于代码搜索任务. 在本文中, 将判别器设置为计算代码和描述语句之间的余弦相似度, 如公式 (13) 所示:

$$f_\phi(d,c) = \cos(d,c) \tag{13}$$

生成器优化目标: 与判别模型的目标相反, 生成器 $p_\theta(c|d_n,r)$ 的目的在于使生成器模型所生成的负样本对能够尽可能地接近正样本, 进而达到欺骗判别器的目的. 通过最小化其损失函数实现该目标, 只考虑公式 (10) 中生成器的部分, 得到生成器的优化目标为公式 (14):

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{n=1}^{N} (\mathbb{E}_{c\sim p_{\text{true}}(c|d_n,r)}[\log(\sigma(f_\phi(d,c)))] + \mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\log(1-\sigma(f_\phi(d,c)))]) \tag{14}$$

由于生成器所包含的参数仅参与公式 (14) 中第 2 项的计算, 联立公式 (11) 和公式 (14), 将生成模型的优化目标从最小化公式 (14) 转换为最大化公式 (15):

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{n=1}^{N} (\mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\log(1+\exp(f_\phi(d_n,c)))]) \tag{15}$$

在本文中, 使用公式 (16):

$$J^G(d_n) = \mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\log(1+\exp(f_\phi(d_n,c)))] \tag{16}$$

作为生成模型的最终优化目标. 按照生成器的定义, 生成器应该生成虚假的代码片段和描述文本供判别器进行判断. 但目前存在以下两个难点尚未解决: (1) 当生成器所需要生成的数据为文本数据时, 其离散的性质使得生成器的损失函数无法采用梯度下降策略进行训练[29]. (2) 生成器生成代码序列会出现误差累积现象, 因为代码片段通常长度较长, 即使能够使用如 SeqGAN[30]等模型生成虚假代码片段, 也需要额外对所生成数据进行评估, 其效果也会影响模型的效果和效率. 为了解决以上两个问题, 本文提出使用近似采样策略, 从已有的数据集中选择与正样本最相似的负样本模拟生成器生成样本的过程, 实现生成器损失函数的梯度计算, 完成对生成器的训练. 首先本文给出连续情况下生成器优化目标 $J^G(d_n)$ 对参数 θ 的梯度计算推导过程:

$$\begin{aligned}
\nabla_\theta J^G(d_n) &= \nabla_\theta \mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\log(1+\exp(f_\phi(d_n,c)))] \\
&= \nabla_\theta \sum_{i=1}^{M} p_\theta(c_i|d_n,r)\log(1+\exp(f_\phi(d_n,c_i))) \\
&= \sum_{i=1}^{M} \nabla_\theta p_\theta(c_i|d_n,r)\log(1+\exp(f_\phi(d_n,c_i))) \\
&= \sum_{i=1}^{M} p_\theta(c_i|d_n,r)\frac{\nabla_\theta p_\theta(c_i|d_n,r)}{p_\theta(c_i|d_n,r)}\log(1+\exp(f_\phi(d_n,c_i))) \\
&= \sum_{i=1}^{M} p_\theta(c_i|d_n,r)\nabla_\theta \log(p_\theta(c_i|d_n,r))(1+\exp(f_\phi(d_n,c_i))) \\
&= \mathbb{E}_{c\sim p_\theta(c|d_n,r)}[\nabla_\theta \log(p_\theta(c|d_n,r))\log(1+\exp(f_\phi(d_n,c)))]
\end{aligned} \tag{17}$$

可以看到, 生成器的优化目标梯度等价于求变量 $\nabla_\theta \log(p_\theta(c|d_n,r))\log(1+\exp(f_\phi(d_n,c)))$ 符合 $c\sim p_\theta(c|d_n,r)$ 分布情况下的期望值. 为了解决前文提及的两个问题, 本文使用采样的方式计算该期望值, 通过在数据集中抽取 K 个

负样本, 近似实现生成器生成样本的过程. 因此公式 (17) 转换为公式 (18):

$$\nabla_\theta J^G(d_n) \approx \frac{1}{K}\sum_{k=1}^{K}\nabla_\theta \log(p_\theta(c_k|d_n,r))\log(1+\exp(f_\phi(d_n,c_k))) \tag{18}$$

在本文中, K 表示负样本数. 该近似公式两点优势, 一是能够实现对生成器目标函数的梯度计算; 二是不需要对整个代码数据集进行全量计算就能获得样本的近似概率分布, 选择与正样本最相近的负样本供判别器识别. 其中 $\log(1+\exp(f_\phi(d_n,c_k)))$ 作为判别器给生成器的反馈[31,32], 当 (d_n,c_k) 在“代码-描述语句”语义空间中越相近时, 生成器获得的反馈越大, 促使生成器更加关注这些样本数据. 对于负样本数 K, 在本文实验代码的具体实现过程中, 采用的是在每一个批处理数据中进行采样的方式, 这一步骤避免在全量数据集进行采样, 极大地提高了模型训练速度.

同时为了提高生成器对最相似样本的关注程度, 本文提出了温度控制机制, 通过改变温度参数 τ 的值, 使生成器更加关注与正样本相似的负样本. 当温度参数数值 τ 越小时, 生成器就会越关注于与正样本相似的负样本数据. 因此本文中生成器的最终构建方式如公式 (19) 所示:

$$p_\theta(c_k|d,r) = \frac{\exp(g_\theta(d,c_k)/\tau)}{\sum_{i=1}^{N}\exp(g_\theta(d,c_i)/\tau)} \tag{19}$$

其中, $p_\theta(c_k|d,r)$ 表示从数据集中抽取与描述文档相近的代码片段的概率, 在此步骤中, 本文使用描述语句和代码片段之间的余弦相似度作为衡量标准, 如公式 (20) 所示:

$$g_\theta(d,c_k) = \cos(d,c_k) \tag{20}$$

可以注意到, 公式 (13) 和公式 (20) 是相同的, 两者的含义均表示 c 和 d 两者之间的相关程度 r. 在本文中这样设置的目的是使生成器和判别器在训练过程中能多次完成对代码编码器和描述语句编码器的训练.

至此, 基于生成对抗策略的代码搜索方法介绍完毕. 正如流程图图 4 所示, 在生成器和判别器的优化过程中, 同时实现了对代码编码器和描述语句编码器的训练, 使代码编码器和描述语句编码器能够生成更高质量的代码表征和描述语句表征用于代码搜索任务. 对于代码搜索任务验证的具体实现, 本文采用与文献 [9–11] 中相同的方式, 故本文略过此部分.

表 1 数据集详细信息

数据集划分	数量	语言
训练集	18 233 872	Java
测试集	10 000	Java

表 2 模型关键参数设置信息

参数	参数值	参数	参数值
方法名长度	6	批处理大小	128
API序列长度	30	训练次数	10
Token长度	50	学习率	1E–4
描述文本长度	30	代码嵌入长度	512
词汇表大小	10 000	描述文本嵌入长度	512

3 实验分析

3.1 实验数据

本文采用文献 [9] 所公开的数据集进行实验, 该数据集在 GitHub 网站上收集了超过 1 800 万条包含描述文档的 Java 代码. 该数据集的详细信息如表 1 所示.

本工作实验中所有代码使用 Python 编程语言编写, 所使用的 PyTorch 版本为 1.8.1. 代码运行的环境为一个运行 Ubuntu 16.04 操作系统的服务器, 该服务器包括 128 GB 运行内存以及两块具有 11 GB 显存的 NVIDIA RTX 2080 Ti 显卡. 本文所提出的模型在训练过程中所使用的关键参数设置如表 2 所示.

3.2 评价指标及基准模型

本文使用搜索任务中常见的评价指标对实验结果进行评估, 这些指标分别是: 查全率, 归一化折损累计增益和平均倒数排名[9–11], 本节将对这些指标进行介绍.

查全率 (*Recall*) 所表示的是模型预测的所有结果中, 最优结果所对应的索引值. 查全率越高表示模型能够搜索到更多的有效结果. 在本文中, 我们采用 *Recall@k* 表示在返回 k 个预测结果时所获得的查全率.

$$Recall@k = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\epsilon\left(FRank_{Q_j} \leqslant k\right) \tag{21}$$

其中, $|Q|$ 表示测试集中查询序列数目, ϵ 为一个指示函数, 当 Q_i 出现在 k 个返回结果中时 $\epsilon = 1$, 否则, $\epsilon = 0$.

平均倒数排名 (mean reciprocal rank, *MRR*) 反映的是正确的搜索结果在模型所返回的结果列表中第 1 次出现的位置信息, 该指标的计算方式如下:

$$MRR@k = \frac{1}{|Q|}\sum_{j=1}^{|Q|}1/FRank_{Q_j} \tag{22}$$

其中, $FRank_{Q_j}$ 表示对于查询序列 Q_j, 在模型搜索返回的 k 个结果中, 第 1 次出现正确结果的索引. 需要注意的是, 如果没有正确的搜索结果, 那么 $1/Index_{Q_j} = 0$. *MRR* 值越高, 表明模型的搜索能力越强, 反之越弱.

归一化折损累计增益 (normalized discounted cumulative gain, *NDCG*) 反映的是查询片段和对应代码片段之间的相关性, 其计算公式如下:

$$NDCG@k = \frac{1}{|Q|}\sum_{j=1}^{k}\frac{2^{r_j}-1}{\log_2(1+j)} \tag{23}$$

其中, Q 表示测试集中查询语句, r_j 表示返回的第 j 个结果在 k 个返回结果中的相关性. 标准化收益指标越高, 表明搜索模型不仅能够返回正确的搜索结果, 而且该正确结果在返回结果列表中的排名还靠前.

3.3 实验方法

为了比较本文所提出方法的有效性, 我们选择了目前在基于深度学习的代码搜索领域中比较著名的几种方法作为基准方法进行实验对比. 以下将对这些方法进行介绍.

DeepCS[9]: Gu 等人在 2018 年首次提出将深度学习技术应用在代码搜索领域. 该方法将代码片段所包含的 3 个部分: 方法名、API 调用序列和词汇单元与该代码对应的描述进行联合嵌入表征, 通过计算代码片段与其正样本和负样本之间的三元损失函数进行模型训练, 使得代码片段和其描述文档具有相近的空间表征, 从而完成代码搜索任务.

CARLCS[10]: Shuai 等人在 2020 年提出使用卷积神经网络替换 DeepCS 模型中用于序列特征抽取的序列神经网络模型, 该方法还考虑了共同注意力机制, 学习代码片段和查询语句的相互依赖表示.

SAN-CS[11]: Fang 等人在 2021 年提出了 SAN-CS 方法. 该方法考虑到 DeepCS 模型中代码序列信息抽取不完全的不足, 使用具有自注意力机制的神经网络用于代码不同特征之间关系抽取. 由于其所采用的是可并行的神经网络模块, 故其检测速度要远高于 DeepCS 模型. 本文使用该文章提供的代码进行代码搜索模型的训练, 并且在模型预测阶段, 去除了其向量联合嵌入部分以避免出现标签泄露现象.

GCS: 本文所提出的方法. 采用生成对抗策略对代码和描述语句进行建模, 着重考虑到了代码和描述语句之间中存在的词汇鸿沟问题. 采用生成对抗机制, 生成器用于生成虚假代码的表征向量, 判别器判断所输入的数据是否是真实数据.

3.4 研究问题

研究问题 1: 本文所提代码搜索方法的效果.

为了回答这个问题, 我们选择了几种现有效果最好的代码搜索方法作为对比方法进行实验结果比较, 本文所使用的数据集与这些方法所使用的数据集是相同的. 同时, 本文还比较了不同方法的训练和搜索效率.

研究问题 2: 不同超参数对实验效果的影响.

为了回答这个问题, 在本实验中我们选取了两个关键的超参数进行研究, 分别是负样本采样数 K 和温度 τ. 负样本采样数表示生成器所生成最接近真实样本的负样本数量. 温度表示的是生成器对最相似样本的关注程度.

研究问题 3: 不同代码结构对实验效果的影响.

本文将代码转换为了 3 个部分, 分别是方法名、API 序列和词汇单元序列, 此问题研究这 3 个部分单独对实

验效果的影响, 同时也分析了三者之间不同的组合关系对实验效果的影响.

4 实验结果和讨论

4.1 研究问题 1: 本文所提代码搜索方法的效果

为了验证本文所提出的方法在代码搜索任务上的有效性, 本文选取了 3 种不同的代码搜索方法进行实验, 分别是 DeepCS、CARLCS 和 SAN-CS. 所有方法均采用了相同的数据集以及相同的数据集划分进行实验. 在这 3 种方法中, 我们直接从文献 [11] 中引用了其复现 CARLCS 方法的实验结果. 对于 DeepCS 和 SAN-CS 两种方法, 本文使用其作者所公开的源代码进行实验复现, 所采用的参数均为其论文中所提供的参数, 其中对于 DeepCS 和 SAN-CS 方法中所使用的三元损失函数, 其锚定参数设置为 $\epsilon = 0.3986$. 并且在 SAN-CS 方法的测试过程中, 为了避免出现“标签泄露”现象, 本文在其模型预测阶段去除了训练时的联合嵌入部分, 因此记该方法为 SAN-CS*. 本文方法中的两个超参数分别设置为: 负采样数 $K = 3$, 温度参数 $\tau = 0.1$. 最终的实验结果如表 3 所示.

表 3 不同方法的实验结果对比 (%)

模型	*Recall*@1	*Recall*@5	*Recall*@10	*MRR*@10	*NDCG*@10
DeepCS	58.4	75.1	81.0	55.3	61.7
CARLCS	54.9	71.3	78.2	53.5	59.2
SAN-CS*	59.2	74.9	80.1	56.5	62.3
Ours	**75.9**	**84.8**	**87.8**	**73.3**	**76.7**

从表 3 中可以看出本文所提出的方法在各个指标上均取得了最好的效果, 本文方法相比于 DeepCS、CARLCS 和 SAN-CS*在 *MRR*@10 指标上分别提升了 32.5%、37.0% 和 29.7%. 本文方法相比于 DeepCS、CARLCS 和 SAN-CS*在 *NDCG*@10 指标上分别提升了 24.3%、29.6% 和 23.1%. 本文方法相比于 DeepCS、CARLCS 和 SAN-CS*在 *Recall*@10 指标上分别提升了 8.4%、12.3% 和 9.6%. 本文中所采用的代码特征提取模型和文本特征提取模型与 DeepCS 相同, 因此通过对比 DeepCS 和本文所提出模型的效果可知, 本文提出的基于生成对抗策略的代码搜索方法是有效的. 具体如下.

(1) CARLCS 用于代码搜索任务的效果最差. 主要原因在于其所采用的 CNN 神经网络对于文本信息的提取要弱于 RNN 神经网络, 尤其是该模型对长度较长的词汇单元的特征提取也采用了 CNN 模型对其建模, 而其余模型均采用适合对文本信息提取的 RNN 神经网络结构.

(2) 本文提出的基于生成对抗策略的代码搜索方法在所有指标上均要明显优于其他方法, 最主要的原因在于本方法中提出的生成器采样方法, 通过生成器和判别器的交替训练, 间接利用数据集中隐含的数据分布信息, 从数据集中选择最接近真实样本的数据供判别器训练. 这一过程中同时也会对代码编码器和描述语句编码器进行优化, 使两者更加关注于代码和描述语句之间的更细微的差别, 从而生成高质量的表征向量用于提高代码搜索任务的效果.

同时本文也考察了不同方法的效率对比. 对于这 3 种模型, 我们均设置其批处理大小为 128, 计算三者模型训练和模型测试时的速度. 在代码训练过程中, 本文计算总样本数与总训练时间之间的比值作为训练速度. 在模型测试阶段, 给出一个描述语句表征向量, 在 10 000 个代码片段对应的表征向量中查找最匹配的代码列表作为结果返回. 如表 4 所示, 由于 SAN-CS*所采用的是可并行计算的神经网络模块, 因此其训练速度要远高于剩余两种模型, 13.7 sample/s 表示的是该模型平均每秒能够训练 13.7 个样本. DeepCS 模型的训练速度几乎相当于本文所提出模型的一半, 这是因为 DeepCS 模型采用的是三元损失函数进行训练, 不但要计算代码片段与正描述语句之间的相似度, 还需要计算代码片段与负描述语句之间的相似度. 本文提出方法的测试速度要远高于 DeepCS 和 SAN-CS 模型, 因为在这两个模型所公开的代码实现中, 其测试阶段仍然保留了负样本的表征生成步骤, 这使得两者在测试阶段的速度要低于本文方法. 表 4 表示本文方法

表 4 不同模型的效率比较

模型	训练速度 (sample/s)	测试速度 (query/s)
DeepCS	5.88	9.50
SAN-CS*	**13.7**	9.52
Ours	9.52	**37.7**

在单块 NVIDIA RTX2080Ti 显卡上每秒能够返回 37.7 条查询语句的搜索结果.

4.2 研究问题 2: 不同超参数对实验效果的影响

本文所提出的基于生成对抗策略的代码搜索方法中有两个重要的超参数, 负样本采样数和温度参数. 本节内容将介绍不同超参数对代码搜索任务的影响.

首先考察的是负样本采样数对模型的影响. 从图 5 中我们可以观察到, 在训练轮数小于 4 的情况下, 负样本采样数越高, 模型的效果越差. 这是因为在模型训练初期, 判别器尚未学习到足够的参数对样本的真假情况进行区分. 尤其是负样本数量多的情况下, 判别器更加难以判断样本的真假. 而随着训练轮数的增加, 模型状态趋于稳定, 此时当负样本采样数为 3 时, 所提出模型在各个指标上均取得了最好的效果. 具体如下.

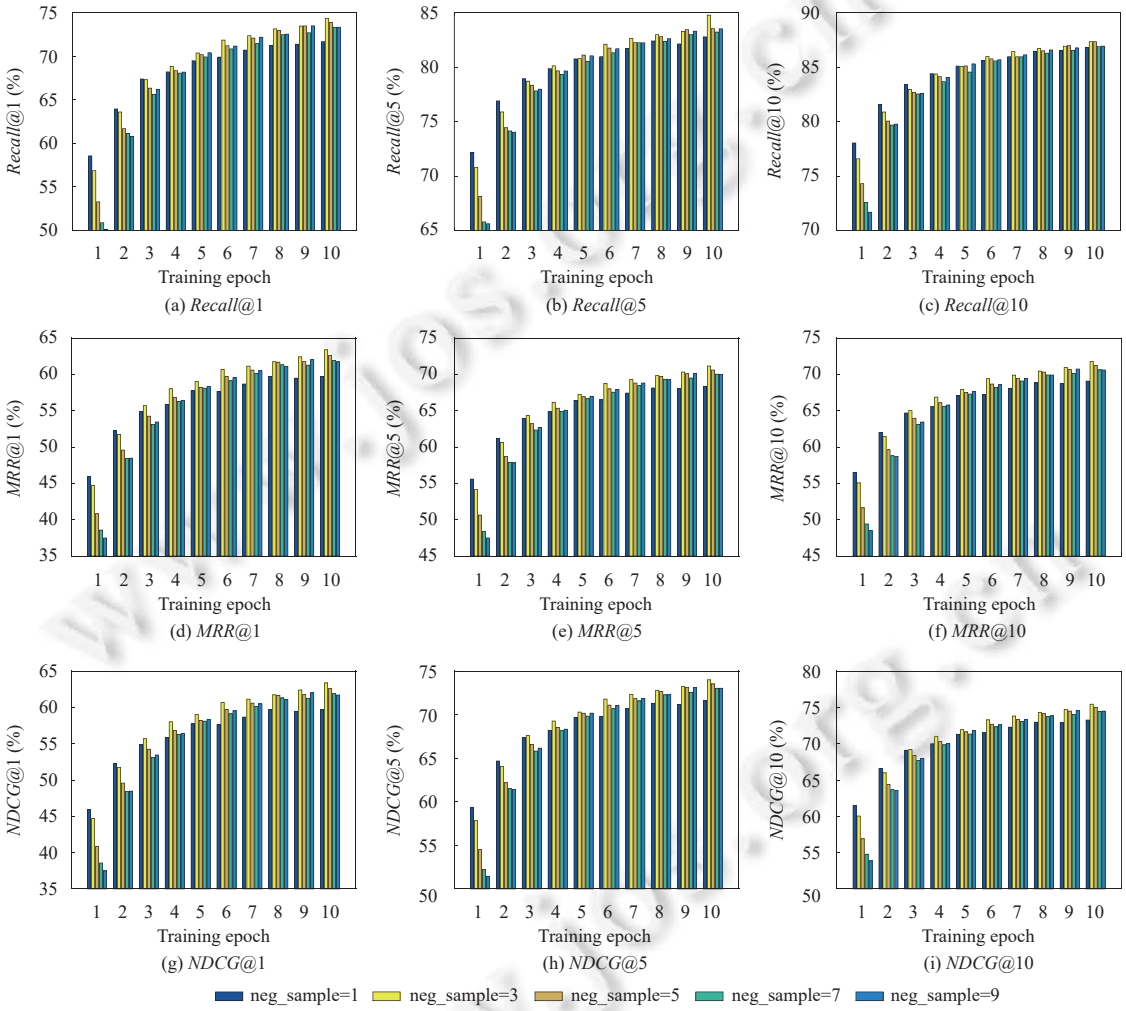

图 5 不同负采样值对实验效果的影响

(1) 当负采样数为 1 时, 本文方法获得的与排序相关的评价指标 (*MRR* 和 *NDCG*) 的结果要明显低于其他负采样数. 出现这种现象的原因是因为生成器所生成的负样本是与正样本极其相似的, 用合适数量的负样本去训练判别器, 能够使其获得更强的分辨能力, 因此负样本数多的情况模型效果好.

(2) 当负样本数超过 3 时, 负样本数的增加并没有增加代码搜索任务的效果. 出现这一现象的原因在于本文中所使用的判别器仅仅是普通的二分类器, 其判别能力有限, 即使有更多的有效样本进行训练, 判别器的判别能力也

无法得到提升. 可以设计更加适合代码搜索任务的判别器来改善这一现象.

其次是温度参数 τ. 在本实验中, 将温度设置为 $[0,1]$, 步长为 0.1 进行实验, 实验的结果如图 6 所示. 从图 6 中可以观察到, 模型的效果随着温度参数的上升而下降, 造成这一现象的主要原因在于, 当温度参数数值较低时, 生成器能够更加关注于代码和描述语句更为接近的样本. 如公式 (18) 所示, 当温度参数较小时, 生成器能够从相似度更高的代码和描述语句中获得更高的奖励, 这会使得生成器更加关注接近真实样本的负样本, 并将其输入判别器进行判断, 从而强化训练判别器的分辨能力.

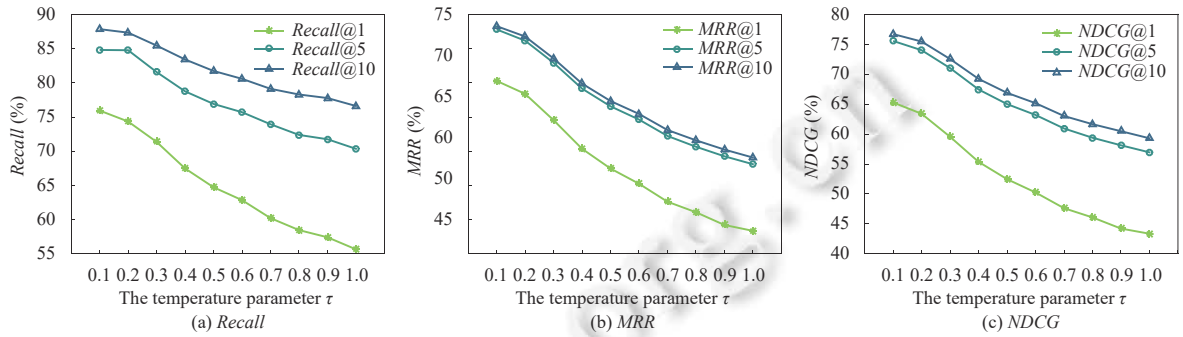

图 6 不同温度参数值对实验效果的影响

4.3 研究问题 3: 不同代码结构对实验效果的影响

本文将代码转换为方法名、API 序列和 Token 序列 3 个部分, 将这 3 个部分的特征合并起来作为代码的表征向量. 本实验要研究的是这 3 个部分对实验效果的影响. 为此本文首先分别使用方法名、API 序列和词汇单元序列作为代码的特征进行实验, 分别记为 Ours(M), Ours(A), Ours(T), 并设计了三者之中两两组合的实验, 即 Ours(MA) 表示使用方法名和 API 序列作为模型输入, Ours(AT) 表示使用 API 序列和 Token 序列作为模型输入, Ours(MT) 表示使用方法名和词汇单元序列作为模型输入. 实验结果如表 5 所示.

表 5 不同代码结构对代码搜索任务的影响 (%)

模型结构	*Recall*@1	*Recall*@5	*Recall*@10	*MRR*@1	*MRR*@5	*MRR*@10	*NDCG*@1	*NDCG*@5	*NDCG*@10
Ours(M)	59.8	72.8	78.3	49.0	57.5	58.4	49.0	60.9	63.0
Ours(A)	55.9	63.3	69.4	40.0	48.5	49.4	40.0	51.8	54.0
Ours(T)	64.3	74.5	78.9	53.4	61.3	62.0	53.4	64.3	66.0
Ours(MA)	73.1	82.8	86.8	62.2	70.0	70.6	62.2	73.0	74.4
Ours(AT)	66.1	76.2	80.7	55.5	63.2	63.9	55.4	66.2	67.8
Ours(MT)	73.4	83.0	87.1	62.8	70.4	71.1	62.8	73.3	74.8
Ours	**75.9**	**84.8**	**87.8**	**65.3**	**72.8**	**73.3**	**65.3**	**75.6**	**76.7**

从表 5 中可知, 不同的代码结构对代码搜索任务均有影响, 具体如下.

(1) 使用多种特征组合进行实验所获得的效果比仅使用一种特征进行实验更好. 这是因为多种特征组合可以提供更多的有效信息用于生成代码表征向量. 单独使用 3 个不同代码特征进行代码搜索任务, 其中使用词汇单元所取得的效果是最好的, 这是由于词汇单元长度比其余两者长, 能够包含更多与描述语句相匹配的符号. 仅使用方法名作为特征时也取得了不错的效果, 原因在于代码的方法名通常都是以体现代码功能内容进行命名的, 同时代码的描述语句也是为了进一步体现代码功能信息.

(2) 当 3 个特征两两组合时, 使用方法名和词汇单元 (Ours(MT)) 的组合效果最好, 出现这一现象的原因在于其组合了两种有效的特征, 因此能够获得更好的效果. 同时, 可以观察到, 将 3 个特征组合使用时, 所取得的效果在 *NDCG*@10 指标上仅比使用方法名和词汇单元 (Ours(MT)) 的效果提升了 2.5%, 这说明在引入 API 序列特征之后得到的效果提升有限, 这一现象启发我们应采用更有效的特征提取方式以及特征组合方法使用 API 序列特征.

5 总 结

本文提出了一种基于生成对抗策略的代码搜索方法. 与现有的代码搜索训练方法不同, 本文方法包含生成器和判别器, 其中生成器用于生成负样本, 判别器判断输入其中的数据是真实存在的还是由生成器生成的. 通过两者的交互训练, 使得模型能够更好地提取代码片段和描述语句之间相关联的特征, 从而提高代码搜索的精度.

在接下来的工作中, 我们将从以下两点来提升代码搜索的效果和效率. (1) 采用效率和效果更好的模型提取代码特征, 如使用 Transformer 模块用于代码特征的提取. (2) 降低代码搜索空间, 提升代码搜索的效率, 如事先对代码库中代码进行分类, 查询语句只需在分好类的代码集合中进行搜索.

References:

[1] Singer J, Lethbridge T, Vinson N, Anquetil N. An examination of software engineering work practices. In: Proc. of the 1997 Conf. of the Centre for Advanced Studies on Collaborative Research (CASCON). Toronto: IBM Press, 1997. 174–188. [doi: 10.5555/782010.782031]

[2] Sadowski C, Stolee KT, Elbaum S. How developers search for code: A case study. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). Bergamo: ACM, 2015. 191–201. [doi: 10.1145/2786805.2786855]

[3] Hill E, Pollock L, Vijay-Shanker K. Improving source code search with natural language phrasal representations of method signatures. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lawrence: IEEE, 2011. 524–527. [doi: 10.1109/ASE.2011.6100115]

[4] Lv F, Zhang HY, Lou JG, Wang SW, Zhang DM, Zhao JJ. CodeHow: Effective code search based on API understanding and extended boolean model (E). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln: IEEE, 2015. 260–270. [doi: 10.1109/ASE.2015.42]

[5] Cambronero J, Li HY, Kim S, Sen K, Chandra S. When deep learning met code search. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE). Tallinn: ACM, 2019. 964–974. [doi: 10.1145/3338906.3340458]

[6] Salza P, Schwizer C, Gu J, Gall HC. On the effectiveness of transfer learning for code search. IEEE Trans. on Software Engineering, 2023, 49(4): 1804–1822. [doi: 10.1109/TSE.2022.3192755]

[7] Zhang XP, Liu JX. Overview of deep learning-based code representation and its applications. Journal of Frontiers of Computer Science and Technology, 2022, 16(9): 2011–2029 (in Chinese with English abstract). [doi: 10.3778/j.issn.1673-9418.2110073]

[8] Hu HZ, Liu JX, Zhang XP, Cao B, Cheng SQ, Long T. A mutual embedded self-attention network model for code search. Journal of Systems and Software, 2023, 198: 111591. [doi: 10.1016/j.jss.2022.111591]

[9] Gu XD, Zhang HY, Kim S. Deep code search. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE). Gothenburg: ACM, 2018. 933–944. [doi: 10.1145/3180155.3180167]

[10] Shuai JH, Xu L, Liu C, Yan M, Xia X, Lei Y. Improving code search with co-attentive representation learning. In: Proc. of the 28th Int'l Conf. on Program Comprehension (ICPC). Seoul: ACM, 2020. 196–207. [doi: 10.1145/3387904.3389269]

[11] Fang S, Tan YS, Zhang T, Liu YP. Self-attention networks for code search. Information and Software Technology, 2021, 134: 106542. [doi: 10.1016/j.infsof.2021.106542]

[12] Manning CD, Raghavan P, Schütze H. Introduction to Information Retrieval. Cambridge: Cambridge University Press, 2008. [doi: 10.1017/CBO9780511809071]

[13] Subramanian S, Inozemtseva L, Holmes R. Live API documentation. In: Proc. of the 36th Int'l Conf. on Software Engineering (ICSE). Hyderabad: ACM, 2014. 643–652. [doi: 10.1145/2568225.2568313]

[14] Du L, Shi XZ, Wang YL, Shi ES, Han S, Zhang DM. Is a single model enough? MuCoS: A multi-model ensemble learning approach for semantic code search. In: Proc. of the 30th ACM Int'l Conf. on Information & Knowledge Management (CIKM). ACM, 2021. 2994–2998. [doi: 10.1145/3459637.3482127]

[15] Ling CY, Lin ZQ, Zou YZ, Xie B. Adaptive deep code search. In: Proc. of the 28th Int'l Conf. on Program Comprehension (ICPC). Seoul: ACM, 2020. 48–59. [doi: 10.1145/3387904.3389278]

[16] Gu WC, Li ZJ, Gao CY, Wang CZ, Zhang HY, Xu ZL, Lyu MR. CRaDLe: Deep code retrieval based on semantic dependency learning. Neural Networks, 2021, 141: 385–394. [doi: 10.1016/j.neunet.2021.04.019]

[17] Chen ZZ, Jiang RH, Zhang ZJ, Pei Y, Pan MX, Zhang T, Li XD. Enhancing example-based code search with functional semantics. Journal of Systems and Software, 2020, 165: 110568. [doi: 10.1016/j.jss.2020.110568]

[18] Gu J, Chen ZM, Monperrus M. Multimodal representation for neural code search. In: Proc. of the 2021 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Luxembourg: IEEE, 2021. 483–494. [doi: 10.1109/ICSME52107.2021.00049]

[19] Meng Y. An intelligent code search approach using hybrid encoders. Wireless Communications and Mobile Computing, 2021, 2021: 9990988. [doi: 10.1155/2021/9990988]

[20] Xu L, Yang HH, Liu C, Shuai JH, Yan M, Lei Y, Xu Z. Two-stage attention-based model for code search with textual and structural features. In: Proc. of the 2021 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Honolulu: IEEE, 2021. 342–353. [doi: 10.1109/SANER50967.2021.00039]

[21] Zou YZ, Ling CY, Lin ZQ, Xie B. Graph embedding based code search in software project. In: Proc. of the 10th Asia-Pacific Symp. on Internetware. ACM, 2018. 1. [doi: 10.1145/3275219.3275221]

[22] Sun ZS, Li L, Liu Y, Du XN, Li L. On the importance of building high-quality training datasets for neural code search. In: Proc. of the 44th Int'l Conf. on Software Engineering (ICSE). Pittsburgh: ACM, 2022. 1609–1620. [doi: 10.1145/3510003.3510160]

[23] Yu Y, Si XS, Hu CH, Zhang JX. A review of recurrent neural networks: LSTM cells and network architectures. Neural Computation, 2019, 31(7): 1235–1270. [doi: 10.1162/neco_a_01199]

[24] Gu XD, Zhang HY, Zhang DM, Kim S. Deep API learning. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering (FSE). Seattle: ACM, 2016. 631–642. [doi: 10.1145/2950290.2950334]

[25] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial networks. Communications of the ACM, 2020, 63(11): 139–144. [doi: 10.1145/3422622]

[26] Wang J, Yu LT, Zhang WN, Gong Y, Xu YH, Wang BY, Zhang P, Zhang D. IRGAN: A minimax game for unifying generative and discriminative information retrieval models. In: Proc. of the 40th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Shinjuku: ACM, 2017. 515–524. [doi: 10.1145/3077136.3080786]

[27] Gui J, Sun ZN, Wen YG, Tao DC, Ye JP. A review on generative adversarial networks: Algorithms, theory, and applications. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(4): 3313–3332. [doi: 10.1109/TKDE.2021.3130191]

[28] Xing Y, Qian XM, Guan Y, Zhang SH, Zhao MC, Lin WT. Cross-project defect prediction method using adversarial learning. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2097–2112 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6571.htm [doi: 10.13328/j.cnki.jos.006571]

[29] Razavi-Far R, Ruiz-Garcia A, Palade V, Schmidhuber J. Generative Adversarial Learning: Architectures and Applications. Cham: Springer, 2022. [doi: 10.1007/978-3-030-91390-8]

[30] Yu LT, Zhang WN, Wang J, Yu Y. SeqGAN: Sequence generative adversarial nets with policy gradient. In: Proc. of the 31st AAAI Conf. on Artificial Intelligence. San Francisco: AAAI, 2017. 2852–2858. [doi: 10.1609/aaai.v31i1.10804]

[31] Zhao JL, Li H, Qu LJ, Zhang QZ, Sun QX, Huo H, Gong MG. DCFGAN: An adversarial deep reinforcement learning framework with improved negative sampling for session-based recommender systems. Information Sciences, 2022, 596: 222–235. [doi: 10.1016/j.ins.2022.02.045]

[32] Chen XC, Yao L, Wang XZ, Sun AX, Sheng QZ. Generative adversarial reward learning for generalized behavior tendency inference. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(10): 9878–9889. [doi: 10.1109/TKDE.2022.3186920]

附中文参考文献:

[7] 张祥平, 刘建勋. 基于深度学习的代码表征及其应用综述. 计算机科学与探索, 2022, 16(9): 2011–2029. [doi: 10.3778/j.issn.1673-9418.2110073]

[28] 邢颖, 钱晓萌, 管宇, 章世豪, 赵梦赐, 林婉婷. 一种采用对抗学习的跨项目缺陷预测方法. 软件学报, 2022, 33(6): 2097–2112. http://www.jos.org.cn/1000-9825/6571.htm [doi: 10.13328/j.cnki.jos.006571]

张祥平(1993－), 男, 博士生, 主要研究领域为代码表征, 代码搜索.

扈海泽(1989－), 男, 博士生, CCF 学生会员, 主要研究领域为代码搜索.

刘建勋(1970－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为云计算, 代码大数据.

刘益(1984－), 女, 博士生, 主要研究领域为代码补全, 代码大数据.