doi:10.3772/j.issn.1002-0470.2024.03.005

基于真值表的函数自动生成的神经网络模型①

贺文凯②*,**,*** 支 天③* 胡 杏* 张曦珊*,*** 张 蕊*,*** 杜子东* 郭 崎*

(*中国科学院计算技术研究所处理器芯片全国重点实验室 北京 100190)

(**中国科学院大学 北京 100049)

(***中科寒武纪科技股份有限公司 北京 100191)

摘 要 作为目前最常见的程序综合问题,示例编程通过用户提供的输入/输出示例生成程序,为编程能力不足的开发者提供了便利。近年来,示例编程已经被应用于 Microsoft Office Excel 办公软件的自动编程,以及勘探、测井、航空航天等领域。鉴于目前示例编程鲜有关于二进制流的研究,本文针对基于真值表函数自动生成问题具有函数表达式的语法符号序列中各语法符号的关系与它们的距离大小无关、函数表达式的生成语义规则与布尔向量函数采样的结果长度无关的特点,设计了一种神经网络模型和算法,在程序综合、功能等价和序列匹配的指标上分别取得了 70.56%、64.66%、0.635 5 的结果,分别优于现有最先进的程序综合模型 55.07%、49.70%、0.569 0。

关键词 真值表;神经网络;序列模型;示例编程;程序综合

程序综合是程序设计理论中的核心问题之一,从底层编程语言中自动找到满足某种约束表示的用户意图的程序,其主要难点在于程序空间的复杂性和用户意图的模糊性[1]。当前的程序综合技术已用于数据处理[2-8]、策略设计[5,8-14]、代码修复[15-16]和代码优化[17-19]等领域,可为编程开发者提供便利,如 FlashFill[2]将 Excel 编程自动化,开发时间由几天缩短到了几秒。

目前最常用的程序综合问题是根据示例给定约束传达用户意图的示例编程。示例编程问题早在 1981 年的 Karel 问题中就有所涉及[9],给定少量的输入/输出示例采样,在程序语言或领域特定语言中生成程序。输入/输出示例可能是字符串[2,20]、整数列表[3]、电子表格[6]、网格[9]等形式的数据。二进制数据作为计算机中的通用数据表示,是当前信息系统最通用、最易得的数据表征。然而,目前鲜有使用二进制流作为示例格式的示例编程研究。因此,本文针对性地提出了基于真值表自动化生成函数的示例编程方法。

基于真值表生成函数的示例编程是指给定一个布尔向量函数和一组语法规则,通过少量的采样构造一个函数表达式。本文针对该问题做出了如下 3 方面贡献。

(1) 发现了基于真值表的问题中函数表达式各语法符号的关系与它们在实际处理序列中的距离大小无关,且函数表达式的生成语义规则与布尔向量函数示例采样的结果长度无关;

(2) 设计了一种编码器-解码器结构的神经网络模型用于解决基于真值表生成函数的问题,该模型用 2 个自注意力机制编码器分别对输入和输出进行编码,用 1 个自注意力机制解码器自回归地生成对应的函数表达式,在编码器和解码器之间使用注

① 国家重点研发计划(2020AAA0103802),国家自然科学基金(61925208,U20A20227,62002338,61906179,62102399,U19B2019,61732020),北京智源人工智能研究院以及北京市科技新星计划(Z191100001119093),中国科学院稳定支持基础研究领域青年团队计划(YSBR-029)和中国科学院青年创新促进会资助项目。

② 男,1997 年生,博士生;研究方向:计算机系统结构;E-mail:hewenkai19b@ict.ac.cn。

③ 通信作者,E-mail:zhitian@ict.ac.cn。

(收稿日期:2022-04-07)

意力机制；

（3）构造了一个对抗性人工合成的真值表生成函数数据集，将本文的神经网络模型和不同的先进基准模型[20-21]在该数据集上进行对比实验，在程序综合、功能等价和序列匹配的指标上作了评估，发现本文方法均优于同期主流基准模型。

1 相关工作

本节介绍示例编程与领域特定语言：示例编程是目前最常用的程序综合问题，本文研究的问题属于示例编程的范畴；领域特定语言是程序综合常用的目标程序语言类型，本文设计并使用的程序语言是一种领域特定语言。

1.1 示例编程

示例编程用程序的输入/输出示例描述用户意图，示例可以是各种形式的数据。1981 年出现的 Karel 问题[9]以智能体所在网格布局作为示例获取指示智能体行动规则的程序。2011 年 Gulwani[2]提出的 FlashFill 问题以微软的计算机软件 Excel 中字符串转换的示例交互地生成自动填充的程序。FlashFill 问题[2]自提出以来受到了广泛的关注与大量的研究，其中最先进的方法之一是 Devlin 等人[20]在 2017 年提出的 RobustFill 框架。RobustFill 采用循环神经网络和注意力机制的结合，用 3 个顺次连接的长短期记忆（long short-term memory，LSTM）网络[22]分别作为字符串样本输入和样本输出的编码器和生成程序的解码器。虽然在 FlashFill 上取得了较好的结果，但 RobustFill 中使用的 LSTM 网络结构不适合解决一部分示例编程问题，特别是基于真值表生成函数的问题，具体将在第 3 节中说明。2015 年 Balog 和 Gaunt[3]提出的 Deepcoder 问题以整数列表的示例生成链式的列表处理程序。2021 年 Chen 等人[6]以 Google Sheets 提供的电子表格作为示例生成操作表格的程序。

1.2 领域特定语言

2008 年，Solar-Lezama[23]提出了在程序综合中使用限定程序空间的语法，使得程序综合的解决方案具有结构化的特征。程序空间的语法可以使用计算机编程中实际使用的编程语言对应的语法，但由于编程语言定义的程序空间往往不利于程序综合的问题描述和解决方式，即通过约束表达用户意图和从程序空间中筛选出适用的程序，也有研究人员为特定的程序综合问题设计的领域特定语言。示例编程中一般都会使用领域特定语言[2-4,6-7,9-12]，它具有混合功能的操作。本文针对基于真值表生成函数的问题设计了一种与编译器的部分结构兼容的领域设计语言，将在 2.2 节中详细介绍。

2 问题描述

本节定义了基于真值表生成函数的示例编程问题：给定一个布尔向量函数 $f(X) = y$ 和一套语法规则，通过少量的采样构造一个函数表达式。其中布尔向量函数由一组真值表给出，输入的向量数与向量位宽在实际问题讨论中有所限制；语法规则是一套领域特定语言。

2.1 基于真值表的布尔向量函数

基于真值表的布尔向量函数 $f(X) = y$ 由一组真值表表示。在这个布尔向量函数的表达式中，输入 $X = \{\boldsymbol{X}_1, \boldsymbol{X}_2, \cdots, \boldsymbol{X}_k\}$ 由一组 k 个布尔向量组成，每一个 $X_i = \overline{x_0^i x_1^i \cdots x_{l_i}^i}$ 由 l_i 个布尔变量排列而成，代表一个二进制数的原码，在布尔向量函数中作为一个整体被使用；输出 $y = \overline{y_0 y_1 \cdots y_m}$ 由 m 个布尔变量排列而成。

每一个布尔向量函数可以表示为该映射规则下所有的输入位宽为 $[k, +\infty)$ 的真值表，真值表的输出与具体的映射关系相关。实际的问题讨论中，限定每一个输入布尔向量的最大位宽 B，则每一个布尔向量函数表示该映射规则下所有输入位宽为 $[k, kB]$ 的真值表。

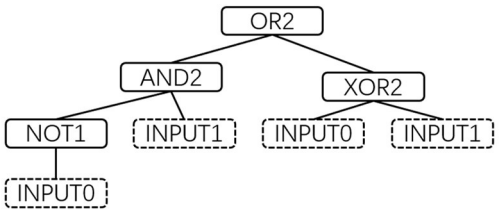
对于每一个布尔向量函数，从这些真值表中采样少量的行用于合成函数表达式，这个函数表达式等价地描述了这个布尔向量函数关系。

2.2 语法规则

本节定义了一套领域特定语言来表示函数表达式的语法规则。该领域特定语言与编译器中语法分析得出的语法树形式一致[24]，可替代从用户需求到

编译器语法分析的程序综合流程,成为计算机体系结构设计的自动化流程中一个高效的步骤。

领域特定语言指定了构造函数表达式的基本方式。函数表达式由一个语法符号序列表示,以“START”作为序列的开始,“END”作为序列的结束,两者之间的序列是函数表达式语法树的前序遍历结果。图1给出了一组语法树和函数表达式对应的例子。在语法树中,每一个实线框的非叶子节点代表一个运算符,每一个虚线框的叶子节点代表一个操作数,也就是布尔向量函数输入的一个布尔向量。将语法树前序遍历展开,首尾分别添加“START”和“END”,就得到了函数表达式。



(a) 语法树

START OR2 AND2 NOT1 INPUT0 INPUT1

XOR2 INPUT0 INPUT1 END

(b) 函数表达式

图1　函数表达式和语法树的对应关系

语法符号包括表示序列起止和用于序列结尾之后占位的功能符,表示逻辑运算、缩减运算、算术运算和移位运算的运算符,以及表示不同输入操作数的输入操作数符。运算符包括单目运算符和双目运算符,每个运算符的最后1个字符表示它所需的操作数数量。运算符的输出位宽与其种类和操作数的位宽相关,根据操作数的位宽取运算符计算结构中最大可能的有效位宽。各个语法符号及其功能在表1中给出。

3　问题特点及分析

基于真值表生成表达式是一个示例编程问题。本节通过比较基于真值表生成表达式与现有示例编程问题的区别,发现了基于真值表生成表达式问题的2个问题:(1)函数表达式的语法符号序列中各语法符号的关系与它们的距离大小无关;(2)函数表达式的生成语义规则与布尔向量函数示例采样的结果长度无关。本节对这2个特点进行说明,并分别分析用现有方法解决这2个问题时的不足,引出本文方法的出发点。

表1　领域特定语言的语法符号

分类	符号	功能
功能符	START|END|PAD	表示语法符号序列的起始、终止及占位
逻辑运算符	NOT1|AND2|NAND2|OR2|NOR2|XOR2|XNOR2	对二进制数做逻辑运算
缩减运算符	AND1|OR1	对二进制数做缩减运算
算术运算符	ADD2|MUL2	对二进制数做算术运算
移位运算符	MOVL1|MOVR1	将二进制数做移位运算
输入操作数符	INPUT0|INPUT1|……	表示不同的输入操作数

3.1　问题特点说明

语法符号关系与距离无关。除去“START”与“END”,语法符号序列是将函数表达式对应语法树前序遍历的结果。定性地定义语法符号的关系为它们在函数表达式的执行中相互关联的程度,例如语法树中父节点和孩子节点具有较紧密的关系,且对于具有交换律的操作符对应的父节点和不同孩子节点的关系紧密程度相近。2.2节定义的语法规则中,直观地定义了语法符号序列中语法符号的位置和任意2个不同语法符号的距离:语法符号的位置按照它们的排列顺序用自然数编码,语法符号的距离为它们位置编码之差。表2给出了图1中语法符号的2个问题:(1)函数表达式的语法符号序列中各语法符号的关系与它们的距离大小无关;(2)函

表2　语法符号序列位置编码(以图1的表达式为例)

语法符号	位置编码	语法符号	位置编码
OR2	0	INPUT1	4
AND2	1	XOR2	5
NOT1	2	INPUT0	6
INPUT0	3	INPUT1	7

数序列的位置编码,其中“OR2”的 2 个孩子节点“AND2”和“XOR2”与父节点“OR2”的距离分别为 1 和 5。因此,函数表达式的语法符号序列中各语法符号的关系与它们的距离大小无关。

生成规则与示例长度无关。每一个布尔向量函数对应着一组相同语义功能的不同的真值表,它们的输入或输出位宽不同,即每一个函数表达式的输入/输出采样得到的示例长度不固定。图 2 中给出了2组不同的示例(每组3对输入/输出示例)。它们都由图 1 给出的函数表达式采样所得,但示例长度不同。因此,函数表达式的生成语义规则与布尔向量函数示例采样的结果长度无关。

Input: [010,000], [01011100,010001], [0101,011]
Output: [010], [01001101], [0110]
(a) 采样结果1

Input: [10,0001], [0000,1000], [00,1111011]
Output: [0011], [1000], [1111011]
(b) 采样结果2

图 2 同一表达式对应的 2 种不同的采样结果

3.2 语法符号关系与距离无关特点分析

本文发现函数表达式的语法符号序列中各语法符号的关系与它们的距离大小无关。然而,RobustFill 的处理过程并不符合这一规则。

RobustFill[20] 中 LSTM 处理表 2 中语法符号序列的示意在图 3 给出。LSTM 的处理过程中,一般地,距离越大的语法符号相互影响程度越小。因此 LSTM 将父节点和左孩子节点设定了高于父节点和其他孩子节点的相互影响程度。在图 1、图 2 和表 2 的例子中,根节点“OR2”和左孩子节点“AND2”之间的距离为 1,它们间相互影响的程度高于根节点与距离为 5 的右孩子节点“XOR2”。

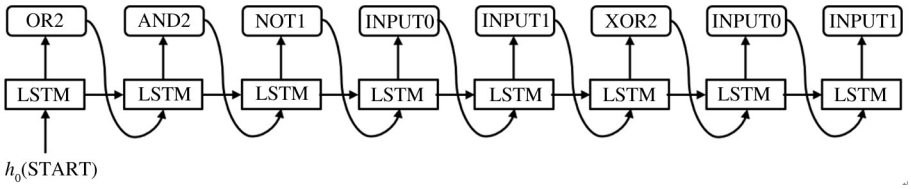


图 3 LSTM 处理语法符号序列示意图

上述处理方式与函数表达式的实际语义关系并不符合。在实际的语义中,父节点与它全部的孩子节点影响程度应相对接近,因为它们对应某一次具体的运算中的运算符和全部操作数。在图 1 的例子中,根节点“OR2”和它的左右孩子,代表函数表达式的最后一次运算。

本文使用自注意力机制[21] 而不是 LSTM 来处理函数表达式语法符号序列以解决上述问题。自注意力机制将序列中所有符号间的距离视为 1,通过数据驱动训练各语法符号间的关系,不受语法符号间的距离影响。

3.3 生成规则与示例长度无关特点分析

本文发现函数表达式的生成语义规则与布尔向量函数示例采样的结果长度无关。事实上,同一个布尔向量函数代表的所有真值表对应的都是同一个函数表达式,不同的采样仅是用户采用不同的方式描述自己的意图,生成函数表达式的过程不应该受到示例长度的影响。

在 RobustFill[20] 的 LSTM 网络中,处理不同长度的采样需要 LSTM 迭代不同的次数。LSTM 网络为了能够进行批处理,一般的做法是确定采样的最大长度,将长度不足的采样结果末尾补足占位符,同时用同等规模的布尔张量标识有效长度。在前向计算的时候,不同长度的采样需要不同的迭代次数,占位符会补足一定的迭代次数,因此计算的结果仍会受到示例长度的影响。

本文使用自注意力机制[21] 而不是 LSTM 来处理采样结果序列以解决上述问题。自注意力机制将序列长度放在了批尺寸的维度,在处理所有的有效位置时共享权重,训练与推理的过程不受采样结果序列长度的影响。

4 方法设计

基于第 3 节中的 2 个问题特点,本节针对性地

设计了一种解决方案,用一个神经网络模型来学习从布尔向量函数的输入/输出采样和2.2节中定义的语法规则生成函数表达式的方法。

4.1 设计原理

针对基于真值表的函数表达式生成问题,本文设计了一个编码器-解码器的神经网络模型,该模型具有2个编码器和1个解码器,以及这三者两两之间的注意力机制,其外层算法和网络结构分别在算法1和图4中给出。该神经网络模型的编码器-解码器结构、注意力机制和自注意力机制都适应基于真值表生成函数问题的特点。

算法1 *Model*(*I*, *O*, *P* = *None*)

输入:示例输入 *I*, 示例输出 *O*, 生成函数语法符号序列 *P* = *None*

输出:生成函数语法符号序列 *P*

1. *OutputI* = *EncoderI*(*I*)
2. *OutputO* = *EncoderO*(*O*, *OutputI*)
3. *P* = *Decoder*(*P*, *Pool*(*OutputI*), *Pool*(*OutputO*))
4. RETURN *P*

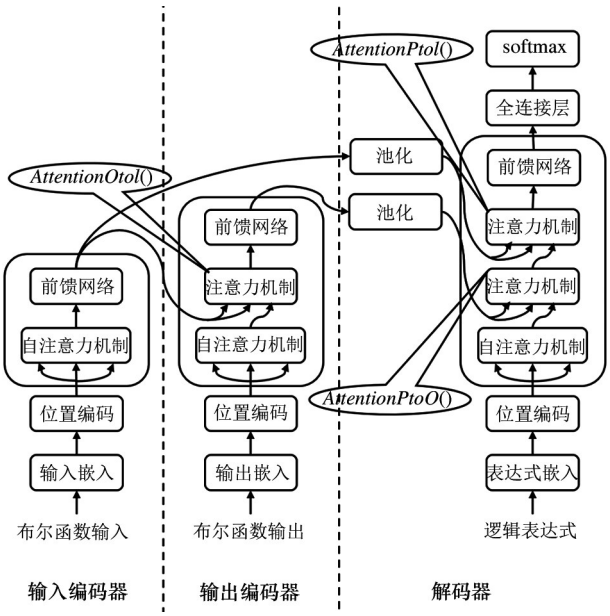


图4 神经网络模型的网络结构

2个编码器和1个解码器的结构契合基于真值表生成函数的问题形式:2个编码器分别处理布尔向量函数的输入和输出,1个解码器用于自回归地生成函数表达式。

注意力机制适合处理从可变长序列生成可变长序列的生成问题。基于真值表的函数表达式生成问题中,布尔函数输入和输出之间具有不易抽象的映射关系,它们和函数表达式也没有显然的对应关系。注意力机制给布尔函数的输入、输出和函数表达式两两之间赋予了关联,且这种关联作用于每个语法符号,使得抽象其中的关系、生成函数表达式更容易。

自注意力机制适合处理基于真值表生成函数的布尔函数采样结果和生成的函数表达式对应语法符号序列的较复杂的内部关系,成功解决了第3节描述的RobustFill[20]中LSTM网络的不足。

4.2 编码器

输入编码器接收布尔向量函数的输入,产生一个编码向量。与Transformer[15]的编码器类似,输入编码器接收一组序列输入,对其位置进行编码,然后经过 *N* 个串联的编码子层,每个编码子层由自注意力机制和前馈网络串联而成,每个自注意力机制和前馈网络周围均使用残差连接并进行层归一化,最终输出一组编码向量。这一组编码向量不直接作为后续输出编码器和解码器的输入,而是在输出编码器和解码器的注意力机制中使用。

输出编码器接收布尔向量函数的输出,产生一个编码向量。输出编码器的结构与输入编码器类似,但编码子层有所不同:在自注意力机制和前馈网络之间,增加了关于输入编码器的注意力机制 *AttentionOtoI*()。*AttentionOtoI*() 的计算过程在算法2给出,其请求(query)是自注意力机制的输出(第1行),键(key)和值(value)均为输入编码器的输出(第2、3行)。*AttentionOtoI*() 周围也使用残差连接并进行层归一化。

算法2 *AttentionOtoI*(*EmbeddedO*, *OutputI*)

输入:示例输出的嵌入 *EmbeddedO*, 输入编码器的输出 *OutputI*

输出:输出编码器的输出 *OutputO*

1. *query* = *EmbeddedO*
2. *key* = *OutputI*
3. *value* = *OutputI*
4. *OutputO* = *Attention*(*query*, *key*, *value*)
5. RETURN *OutputO*

4.3 解码器

解码器采用自回归的方式,生成一个函数表达式。以“START”为开始,解码器逐一接收函数表达式中已经生成的全部语法符号,对其进行位置的编码,然后经过 N 个串联的解码子层,再经过全连接层和 softmax 后生成函数表达式的下一个语法符号,直到出现“END”为止。

每一个解码子层均由自注意力机制、注意力机制和前馈网络组成,每个注意力机制或前馈网络周围均使用残差连接并进行层归一化。区别于 2 个编码器,解码器的自注意力机制对感受野增加了一定的限制,依据生成函数表达式的顺序,保证每一个语法符号只关注它之前的语法符号,而不关注它之后的语法符号。

注意力机制包括关于输出编码器的注意力机制 *AttentionPtoO*( ) 和关于输入编码器的注意力机制 *AttentionPtoI*( )。*AttentionPtoO*( ) 的计算过程在算法 3 给出,其请求是自注意力机制的输出(第 1 行),键和值均为输出编码器的输出在采样数量维度上池化的结果(第 2、3 行)。*AttentionPtoI*( ) 的计算过程在算法 4 给出,其请求是 *AttentionPtoO*( ) 的输出(第 1 行),键和值均为输入编码器的输出在采样数量维度上池化的结果(第 2、3 行)。

算法 3 *AttentionPtoO*(*EmbeddedP*, *PooledOutputO*)

输入:生成函数的嵌入 *EmbeddedP*, 输出编码器输出的池化 *PooledOutputO*

输出:生成函数的中间结果 *OutputP*'

1. *query* = *EmbeddedP*
2. *key* = *PooledOutputO*
3. *value* = *PooledOutputO*
4. *OutputP*' = *Attention*(*query*, *key*, *value*)
5. RETURN *OutputP*'

算法 4 *AttentionPtoI*(*OutputP*', *PooledOutputI*)

输入:生成函数的中间结果 *OutputP*', 输入编码器输出的池化 *PooledOutputI*

输出:生成函数的张量表示 *OutputP*

1. *query* = *OutputP*'
2. *key* = *PooledOutputI*
3. *value* = *PooledOutputI*
4. *OutputP* = *Attention*(*query*, *key*, *value*)
5. RETURN *OutputP*

5 实 验

本节首先介绍实验中使用到数据集的对抗性构造方法,然后介绍实验中的基线和模型设定,最后从不同的指标评估本文模型与基线的对比。

5.1 数据集

本文使用的数据集按照函数表达式-输入/输出示例的顺序采样生成。基于表 1 的语法,本节参考了文献[25,26]中的对抗性采样方式,构造了一个基于真值表生成函数的人工合成的数据集。对抗性体现在每一个可能的采样环节均使用均匀随机采样,使得采样的结果能够尽可能均匀地散布在采样空间(包括程序空间和示例空间)。数据集生成算法在算法 5 中给出,对于数据集中的每一条数据,先对函数表达式采样(第 4 行),将获取的函数表达式去重(第 5、6 行)之后,给每一个函数表达式采样输入/输出对(第 7 行)。

算法 5 *GenerateDataset*(K, M, B, num, $\mathbb{O}$)

输入:生成函数最大输入操作数数量 K,生成函数最大不同操作符数量 M,示例输入最大位宽 B,数据集大小 num,操作符库$\mathbb{O}$

输出:数据集$\mathbb{D}$

1. $\mathbb{D} = \Phi$
2. $\mathbb{P} = \Phi$
3. **while** $|\mathbb{D}| < num$ **do**
4. $[T, P]$ = *SampleFunction*(K, M, $\mathbb{O}$)
5. **if** P not in $\mathbb{P}$ **then**
6. $\mathbb{P} = \mathbb{P} \cup \{P\}$
7. $\mathbb{D} = \mathbb{D} \cup \{(P, SampleExamples(T, K, N, B))\}$
8. **end if**
9. **end while**
10. RETURN $\mathbb{D}$

对于每一条数据,按照算法 6 采样它的函数表达式。定义 L_T 代表所有已经被生成出来的语法树(节点)列表,初始状态为所有 K 个输入操作数符代表的叶子节点(第 1 ~ 6 行)。首先在 $U\{1, 2, \cdots, M\}$ 上采样 m(第 7 行),在所有运算符中有放回地均

匀抽取 m 个运算符(第 10 行),对于每个运算符,从 L_T 中依次有放回地随机抽取若干个元素作为这个运算符的孩子,组成一棵新的语法树(第 11 行),添加到 L_T 的末尾(第 12 行);选择 L_T 中最后一个元素作为本次生成的函数表达式对应的语法树 T;将 T 的节点进行前序遍历,分别在其开始和末尾添加"START"和"END"(第 15 行),得到本次生成函数表达式的语法符号序列 P。

算法 6 *SampleFunction*(K, M, $\mathbb{O}$)

输入:生成函数最大输入操作数数量 K,生成函数最大不同操作符数量 M,操作符库$\mathbb{O}$

输出:生成函数语法树 T,生成函数语法符号序列 P

1. $L_T = [\,]$
2. $i = 0$
3. **while** $i < K$ **do**
4. $\quad L_T = L_T + [node(\text{"}INPUT_i\text{"})]$
5. $\quad i = i + 1$
6. **end while**
7. $m = randomchoice(\{1, 2, \cdots, M\})$
8. $i = 0$
9. **while** $i < m$ **do**
10. $\quad T = node(randomchoice(\mathbb{O}))$
11. $\quad T = addchildren(T, L_T)$
12. $\quad L_T = L_T + [T]$
13. $\quad i = i + 1$
14. **end while**
15. $P = treetoexpression(T)$
16. RETURN T, P

对于每一条函数表达式,按照算法 7 采样 N 对输入/输出作为生成函数表达式的参考依据。这 N 对输入/输出逐一采样(第 3 ~ 10 行),每次采样后与采样的输入/输出进行去重操作(第 11 ~ 13 行)。对于每一对输入/输出样本,先对它的 K 个输入逐一在 $U\{1, 2, \cdots, B\}$ 上采样该输入的位宽 *bitwidth*(第 5 行),然后对于每一位宽 *bitwidth* 在 $U\{1, 2, \cdots, 2^{bitwidth} - 1\}$ 上采样它对应输入的值 *value*(第 6 行),位宽 *bitwidth* 和值 *value* 指定了一个特定位宽的布尔向量(第 7 行),指定这 K 个布尔向量为函数表达式的一个输入样本$\mathbb{I}$;将$\mathbb{I}$代入到函数表达式中得到一个新的布尔向量 $\boldsymbol{O}$(第 10 行),指定 $\boldsymbol{O}$ 为函数表达式对应$\mathbb{I}$的一个输出样本。

算法 7 *SampleExamples*(T, K, N, B)

输入:生成函数语法树 T, 生成函数最大输入操作数数量 K, 示例数量 N, 示例输入最大位宽 B

输出:示例集$\mathbb{D}$

1. $\mathbb{E} = \Phi$
2. **while** $|\mathbb{E}| < N$ **do**
3. $\quad \mathbb{I} = \Phi$
4. $\quad$ **while** $|\mathbb{I}| < K$ **do**
5. $\quad\quad bitwidth = randomchoice(\{1, 2, \cdots, B\})$
6. $\quad\quad value = randomchoice(\{0, 1, \cdots, 2^{bitwidth} - 1\})$
7. $\quad\quad I = dectobin(value, bitwidth)$
8. $\quad\quad \mathbb{I} = \mathbb{I} \cup \{I\}$
9. $\quad$ **end while**
10. $\quad \boldsymbol{O} = calculate(T, \mathbb{I})$
11. $\quad$ **if** $(\mathbb{I}, \boldsymbol{O})$ not in $\mathbb{E}$ **then**
12. $\quad\quad \mathbb{E} = \mathbb{E} \cup \{(\mathbb{I}, \boldsymbol{O})\}$
13. $\quad$ **end if**
14. **end while**
15. RETURN $\mathbb{E}$

为了能够方便、便捷地在实验平台上进行训练,且和基线方法 RobustFill[20] 原本处理的数据集选取尽可能接近的设定,取 $K = 2$, $M = 5$, $N = 5$, $B = 8$。按照上述方式逐条采样数据,构造了一个 200 000 条数据的训练数据集和一个 20 000 条数据的测试数据集。数据集中的函数表达式对应语法符号序列的最大长度为 41。

5.2 基线设置

实验使用示例编程的先进方法 RobustFill[20] 和处理序列的先进方法 Transformer[21] 作为基线与本文设计的神经网络模型进行比较。

RobustFill 是程序综合中 FlashFill[2] 任务的最先进方法之一。实验中分别复现了 RobustFill 的 Attention-B 和 Attention-C 模型作为基线,其中 Attention-C 的 2 个编码器分别使用双向长短期记忆网络。实验设置编码维度为 128,隐层维度为 512。

Transformer 是处理序列的先进方法之一。Transformer 使用自注意力机制和前馈网络,多层叠加分别构成了编码器和解码器。实验复现了 Transformer 作为另一组基线。实验将 Transformer 与文献[21]中描述的设置相同的参数,层数为 6,编码维度为 2 048,隐层维度为 512。

5.3 实验设置

相比 5.2 节中 Transformer 的模型设置,实验使用了一个层数与超参数更小的模型,期望使用更少的模型参数获取更好的效果。实验使用了层数为 $N=3$ 的神经网络模型。其中编码维度设置为 512,隐层维度设置为 256。实验使用 Adam[27] 优化器修改模型参数,批尺寸设置统一为 800。各模型的可训练参数数量在表 3 中给出,本文设计模型的参数量分别是 RobustFill 的 Attention-B、RobustFill 的 Attention-C、Transformer 的 82.27%、24.73% 和 16.17%。

5.4 实验结果

本节分别从示例编程、功能等价、序列匹配 3 个领域的评价指标来评估模型优劣。各评价指标的结果在表 3 中汇总。

表 3 各模型可训练参数量及测试集上不同评价指标结果

	可训练参数量	泛化正确率/%	等价正确率/%	平均 BLEU[28]
RobustFill [Attention-B]	8 688 402	54.66	49.57	0.577 9
RobustFill [Attention-C]	28 907 282	55.07	49.70	0.569 0
Transformer	44 201 491	68.81	62.94	0.627 5
本文模型	7 148 306	70.56	64.66	0.635 5

示例编程评价指标参考文献[20]的评价方式,在保证数据集给定的 5 对输入/输出样本拟合正确的前提下,按算法 7 中示例采样的方法随机生成 1 对与数据集给定均不同的输入/输出样本检查其是否拟合正确。在数据集给定的 5 对输入/输出样本和新生成的 1 对输入/输出样本上均拟合正确的样本数量与测试集大小的比值称为泛化正确率,以百分数表示(保留百分数的小数点后 2 位)。

功能等价评价指标判断模型生成的函数表达式与数据集给定的函数表达式是否具有相同的功能。由于基于真值表生成函数的问题中每个函数表达式理论上对应着无穷多真值表导致输入/输出数量的无穷性(实际上随着限定的输入位宽呈指数增长),实验选择类似示例编程评价指标的采样方式,并用类似泛化正确率的定义来定义等价正确率,不同之处是将其中的 1 对样本替换为 100 对样本。

序列匹配指标采取 BLEU[28] 指标,它判断 2 个序列中部分语法符号序列成功匹配的程度。实验调用 Python 中封装好的库函数计算每个生成的函数表达式与其目标表达式的 BLEU 值,将测试集中全部的结果取平均值作为最终的评价结果(保留小数点后 4 位)。

表 3 汇总了各评价指标的结果。其中数据显示实现了 5.3 节中所述的实验目的,即使用模型参数更少的神经网络模型实现更优的效果。相比不同的基线,本文设计的神经网络模型使用了最少的参数量,在示例编程、功能等价和序列匹配的评价指标上分别取得了 70.56%、64.66%、0.635 5 的结果,比 RobustFill 的 Attention-B 模型的 54.66%、49.57%、0.577 9分别取得了 1.29 倍、1.30 倍、1.10 倍的提升,比 RobustFill 的 Attention-C 模型的55.07%、49.70%、0.569 0 分别取得了 1.28 倍、1.30 倍、1.12 倍的提升,比 Transformer 模型的68.81%、62.94%、0.627 5 分别取得了 1.03 倍、1.03 倍、1.01 倍的提升。使用了比 RobustFill 和本文模型更多模型参数的 Transformer 模型的效果也要优于 RobustFill,不过尚不如本文设计的神经网络模型。

实验结果验证了本文设计的神经网络模型的合理性和有效性。与目前最先进的程序综合框架之一 RobustFill[20] 和先进的序列处理框架 Transformer[21] 相比,本文设计的神经网络模型在基于真值表生成函数的问题上取得了更好的效果。

6 结 论

本文介绍了基于真值表生成函数的示例编程问题。结合现有的示例编程问题与示例编程方法,本文解释了基于真值表生成函数问题具有函数表达式的语法符号序列中各语法符号的关系与它们的距离大小无关、函数表达式的生成语义规则与布尔向量函数示例采样的结果长度无关的问题特点。本文针对性地提出了一种运用了注意力机制和自注意力机制的编码器-解码器的神经网络模型来解决该问题,取得了不错的效果。本文设计了一种对抗性构造数

据集的采样方法,构造了一个基于真值表生成函数的人工合成的数据集。本文分别从程序综合、功能等价和自然语言处理的指标上进行了评估,其中本文设计的方法分别取得了70.56%、64.66%、0.635 5的结果,分别优于现有示例编程最先进的方法之一RobustFill[20]的结果55.07%、49.70%、0.569 0,也优于序列编程的先进方法Transformer[21]的68.81%、62.94%、0.627 5。

参考文献

[1] GULWANI S, POLOZOV O, SINGH R. Program synthesis[J]. Foundations and Trends © in Programming Languages, 2017,4(1-2):1-119.

[2] GULWANI S. Automating string processing in spreadsheets using input-output examples[J]. ACM Sigplan Notices, 2011,46(1):317-330.

[3] BALOG M, GAUNT A L, BROCKSCHMIDT M, et al. Deepcoder: learning to write programs[EB/OL]. (2016-11-07)[2022-04-06]. https://arxiv.org/pdf/1611.01989v1.pdf.

[4] SUN S H, NOH H, SOMASUNDARAM S, et al. Neural program synthesis from diverse demonstration videos[C]//International Conference on Machine Learning. Stockholm, Sweden: PMLR, 2018:4790-4799.

[5] ELLIS K, WONG C, NYE M, et al. Dreamcoder: growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning[EB/OL]. (2020-06-15)[2022-04-06]. https://arxiv.org/pdf/2006.08381v1.pdf.

[6] CHEN X, MANIATIS P, SINGH R, et al. Spreadsheetcoder: formula prediction from semi-structured context[C]// International Conference on Machine Learning. Virtual: PMLR, 2021:1661-1672.

[7] PADHI S, POLGREEN E, RAGHOTHAMAN M, et al. The SyGuS language standard version 2.1[EB/OL]. (2023-12-10). https://arxiv.org/pdf/2006.08381v1.pdf.

[8] WONG C, ELLIS K M, TENENBAUM J, et al. Leveraging language to learn program abstractions and search heuristics[C]// International Conference on Machine Learning. Virtual: ICML, 2021:11193-11204.

[9] PATTIS R E. Karel the robot: a gentle introduction to the art of programming[M]. London: Wiley, 1981.

[10] DANG-NHU R. PLANS: neuro-symbolic program learning from videos[J]. Advances in Neural Information Processing Systems, 2020,33:22445-22455.

[11] PU Y, ELLIS K, KRYVEN M, et al. Program synthesis with pragmatic communication[J]. Advances in Neural Information Processing Systems, 2020,33:13249-13259.

[12] MARIÑO J R H, MORAES R O, OLIVEIRA T C, et al. Programmatic strategies for real-time strategy games[C]//Proceedings of the AAAI Conference on Artificial Intelligence. Virtual: AAAI Press, 2021,35(1):381-389.

[13] 刘江佾. 基于细粒度指令组合的GPGPU极限功耗测试程序自动生成方法研究[D]. 深圳:中国科学院大学, 2021:30-31.

[14] 冯雁星. AORBCO模型中的程序生成研究[D]. 西安:西安工业大学, 2021

[15] DINELLA E, DAI H, LI Z, et al. Hoppity: learning graph transformations to detect and fix bugs in programs[C]//International Conference on Learning Representations. Virtual: ICLR, 2020:1-14.

[16] YASUNAGA M, LIANG P. Graph-based, self-supervised program repair from diagnostic feedback[C]// International Conference on Machine Learning. Virtual: ICML, 2020:10799-10808.

[17] GUPTA K, CHRISTENSEN P E, CHEN X, et al. Synthesize, execute and debug: learning to repair for neural program synthesis[J]. Advances in Neural Information Processing Systems, 2020,33:17685-17695.

[18] LIU Q, AN S, LOU J G, et al. Compositional generalization by learning analytical expressions[J]. Advances in Neural Information Processing Systems, 2020,33:11416-11427.

[19] NYE M, PU Y, BOWERS M, et al. Representing partial programs with blended abstract semantics[EB/OL]. (2020-12-23)[2022-04-06]. https://arxiv.org/pdf/2012.12964v1.pdf.

[20] DEVLIN J, UESATO J, BHUPATIRAJU S, et al. Robustfill: neural program learning under noisy I/O[C]// International Conference on Machine Learning. Toulon, France: ICML, 2017:990-998.

[21] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]//Advances in Neural Information Processing Systems. Long Beach, USA: Association for

Computing Machinery, 2017:5998-6008.

[22] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural Computation, 1997, 9(8): 1735-1780.

[23] SOLAR-LEZAMA A. Program synthesis by sketching [M]. Berkeley: University of California, Berkeley, 2008:1-24.

[24] Alfred, V. A. 编译原理[M]. 北京: 机械工业出版社, 2021:3-5.

[25] SHIN R, KANT N, GUPTA K, et al. Synthetic datasets for neural program synthesis[EB/OL]. (2019-12-27) [2022-04-06]. https://arxiv.org/pdf/2012.12964v1.pdf.

[26] SUH A, TIMEN Y. Adversarial synthetic datasets for neural program synthesis[EB/OL]// 2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). Melbourne, Australia: IEEE, 2020:98-104.

[27] KINGMA D P, BA J. Adam: a method for stochastic optimization[EB/OL]. (2015-06-23)[2022-04-06]. https://arxiv.org/pdf/1412.6980v6.pdf.

[28] PAPINENI K, ROUKOS S, WARD T, et al. BLEU: a method for automatic evaluation of machine translation [C]//Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia, USA: Association for Computing Machinery, 2002:311-318.

Automatically generating function expressions based on truth tables with neural networks

HE Wenkai[*,**,***], ZHI Tian[*], HU Xing[*], ZHANG Xishan[*,***], ZHANG Rui[*,***], DU Zidong[*], GUO Qi[*]

([*]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

([***]Cambricon Technologies, Beijing 100191)

Abstract

Programming by examples is a common program synthesis problem that involves generating programs based on input/output examples provided by users, it offers convenience for novice programmers. Recently, programming by examples has been applied to automatic programming of Microsoft Office Excel, as well as in exploration, logging, and aerospace. In order to address the gap resulting from limited research on binary data flow, the problem of generating function expressions based on truth tables is introduced. This problem has the characteristic that the relationship between the syntactic symbols in the sequence of symbols in the function expression is independent of their distances. Additionally, the generation of semantic rules for the function expression is unrelated to the resulting length of the Boolean vector function sampling. Based on the aforementioned characteristics, this paper introduces a neural network model and algorithm that achieve results of 70.56%, 64.66%, and 0.635 5 in program synthesis, functional equivalence, and sequence matching, respectively. These results outperform the existing state-of-the-art program synthesis model, which achieves 55.07%, 49.70%, and 0.569 0, respectively.

Key words: truth table, neural network, sequential model, programming by examples, program synthesis