Incorporating Domain Knowledge through Task Augmentation for Front-End JavaScript Code Generation —hoho

论文试图解决什么问题？

在代码生成领域中，尤其是在真实业务场景，可用于训练的代码数据有限，导致模型达不到一定的效能，容易过拟合。本文试图解决这个问题。

这是否是一个新的问题？

不是。

这篇文章要验证一个什么科学假设？

让模型通过额外的任务学习领域知识，从可以改善因为代码训练数据少而导致的代码生成的效果。

有哪些相关研究？如何归类？谁是这一课题在领域内值得关注的研究员？

hoho_todo

论文中提到的解决方案之关键是什么？

本文需要处理基于业务需求文档的业务代码生成。

- 数据的形式

Category	String template expression
Description	显示“优惠券已抵扣xx元”，xx为折扣价
Desc. Trans.	Displays “The coupon has been deducted xx yuan”, xx is the discounted price
Code	{`优惠券已抵扣${discountPrice}元`}
Category	OR logic expression
Description	动态展示用户昵称，兜底为空
Desc. Trans.	Dynamically display the user’s nickname, the default value is empty
Code	{ user && user.nick || " " }
Category	Condition expression
Description	如果内容类型为直播，则展示直播时间描述，否则展示营销时间描述
Desc. Trans.	If the content type is live, show the description of the live time, otherwise show the description of the marketing time
Code	{contentType === ’live’ ? liveTimeDesc : marketingTimeDesc}
Category	Data processing expression
Description	动态展示金币展示价格小数部分
Desc. Trans.	Dynamically display the fractional part of coin show price
Code	{`${data.coinShowPrice.split(".")[1]}`}

Table 2: Examples of data in different categories.

- 数据的预处理

1. 代码的规范化（如表示字符串的时候，单双引号使用一致）

使用Esprima parse将代码转换为标准AST，后用Escodegen 将AST转换为规范代码

2. 字符串字面量用通配符替换，如<STR>，<STR2>

3. 简化成员变量

如对于成员对象引用，task.status简化为status

- 任务增强

本文发现变量名占据代码的大部分，所以如果能让模型通过领域知识正确学习这些变量名，将大有帮助。

1. 数据的收集

本文通过以下方式整理出“变量名-语义”的数据对：

— 需求文档

— 代码库（譬如Protocol Buffer文件，通常会有变量名声明，后跟随中文注释，可以从中提取配对）

— 数据库列名定义

最终形成如下表格：

Variable Name	Variable Semantic
shopLogo	店铺标志(shop logo)
beforePromotionPrice	促销前价格(price before promotion)
storeCityName	门店所属城市(city the store located)
roomStatus	直播状态(live room status)
picUrl	图片链接(link of the picture)
...	...

Table 5: Examples of the variable semantic table

2. 设计辅助任务

将以上表格数据视为领域知识，可以预训练一个模型，譬如通过语义预测变量名，但是模型的输出并不适合应用与基于AST的树代码生成模型（TranX）。但本文通过String template expression数据发现，字符串表达式一般只是一个变量名+字符字面表达（不会再拼接其他东西），可以将上面的表格数据改写为形如：

对于数据对“picUrl-图片链接(link of the picture)”，将其改写为输入是：“展示图片链接(show the link of the picture)”，输出是代码表达式：{picUrl;} (这里主要看各种语言的语法规则，输出“picUrl;”也是可以的)

由此，模型的输出就可以应用于基于AST的模型。

3. AST的生成

（1）子词划分，基于变量名驼峰式命名规则或下划线连接规则直接划分子词，如liveTimeDesc划分为live, ##Time, ##Desc （形式类似wordpiece，但是不用wordpiece那种基于统计的方法）

（2）基于TranX方法建立Subtokens-TranX。将自然语言编码为代码有三个阶段：

2.1）使用encoder-decoder将自然语言描述转化为构建AST的序列动作，同时还会检查语法是否符合规则。

2.2）构建AST

根据TranX，构建过程有三种动作：

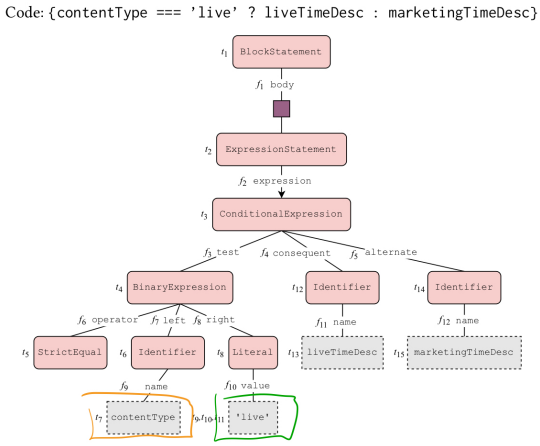
— APPLYCONSTR[c]：在当前域应用一个构建规则c，生成节点（hoho_todo：没理解，看看TranX）

— REDUCE: 表示当前域（域即树的分支）构建结束

— GENTOKEN[v]：生成TOKEN，因为是发生在叶节点，也称为生成终止符

由于本文用的是SubToken，所以GENTOKEN[v]改为GENSUBTOKEN方式，连续生成若干子词直到<EOT>符号。

例子：



t	n_{f_t}	Action
t_1	root	BlockStatement(stmt* body)
t_2	f_1	ExpressionStatement(expr expression)
t_3	f_2	ConditionalExpression(expr test, expr alternate, expr consequent)
t_4	f_3	BinaryExpression(binary_operator operator, expr left, expr right)
t_5	f_6	StrictEqual()
t_6	f_7	Identifier(name)
$t_{7,1}$	f_9	GENSUBTOKEN[content]
$t_{7,2}$	f_9	GENSUBTOKEN[##Type]
$t_{7,3}$	f_9	GENSUBTOKEN[<EOT>]
t_8	f_8	Literal(literal? value)
t_9	f_{10}	GENSUBTOKEN[<SOS>]
t_{10}	f_{10}	GENSUBTOKEN[live]
t_{11}	f_{10}	GENSUBTOKEN[<EOS>]
t_{12}	f_4	Identifier(identifier name)
$t_{13,1}$	f_{11}	GENSUBTOKEN[live]
$t_{13,2}$	f_{11}	GENSUBTOKEN[##Time]
$t_{13,3}$	f_{11}	GENSUBTOKEN[##Desc]
$t_{13,4}$	f_{11}	GENSUBTOKEN[<EOT>]
t_{14}	f_5	Identifier(identifier name)
$t_{15,1}$	f_{12}	GENSUBTOKEN[marketing]
$t_{15,2}$	f_{12}	GENSUBTOKEN[##Time]
$t_{15,3}$	f_{12}	GENSUBTOKEN[##Desc]
$t_{15,4}$	f_{12}	GENSUBTOKEN[<EOT>]
t_{16}	f_1	REDUCE (close the frontier field f_1)

2.3）生成目标代码

3. SubToken-TranX网络架构：（hoho_todo，具体还要看看TranX论文）

使用双向LSTM编码输入$\{x\}_{i=1}^{n}$

输出使用LSTM：

$$s_t = f_{LSTM}\left([a_{t-1}; \tilde{s}_{t-1}; p_t], s_{t-1}\right)$$

$$\tilde{s}_{t-1} = \tanh(W_c[c_{t-1}; s_{t-1}])$$

a_{t-1}是上个时间步的动作

c_{t-1}是用注意力机制计算的上下文

p_t: parent feeding information which is the concatenation of the embedding of the frointer field n_{f_t} and s_{p_t}, the decoder's state at which the constructor of n_{f_f}

(hoho_todo，理解不了)

论文中的实验是如何设计的？

1. 数据集：使用自然语言描述+代码段的数据集（参考定量评估的数据集），以及上面提到的“变量名-语言”数据对的数据集
2. Baseline model: Transformer和TranX
3. 数据的预处理：

   使用工具Esprima和Escodegen作为Javascript代码与AST间的转换。

   Transformer方使用Esprima对Javascript代码进行tokenize

   对自然语言描述使用NLTK和Jieba分词进行tokenize

   本文的模型子词tokenize只根据变量的驼峰命名规则和下划线命名规则进行
4. 模型参数配置：

   SubToken-TranX和TranX:

   encoder和decoder的LSTM隐含层size为256，

   subtoken和action的embedding size为128，

   batch size: 32

   epoch: 300

   learning rate: 1e-3

   Adam

   beam search, width: 5 (for SubToken-TranX)

   Transformer：

   encoder和decoder 层数： 4

   model size : 128

   ffn size: 512

   multi-head attention count: 4

   batch size: 32

   epoch: 300

learning rate: 1e-4

AdamW

beam search, width: 5

5. 验证指标：

   Exact Match Accuracy:

   进行了Top-1和Top-5的精确度测试

   严格的精确度匹配。注意：即使生成代码功能一样当代码字面表达不一样也认为是False

   BLEU

   Edit Similarity

   对于代码字符串

6. **对于三种模型（SubToken-TranX，TranX，Transformer）模型，进行了使用Task augmentation和不使用时的对比验证**
7. **针对四类数据分布进行各种验证指标的对比**
8. **单独对Transformer，对于“代码变量-语义”的数据集，进行下面三种辅助任务和不进行任何辅助任务：**

   PT: pre-training+fine tunning，

   VP: predict vairable name from vairable semantics

   CG: rewriting variable semantic table into code generation paired data. (本文推荐的方式)

用于定量评估的数据集是什么？代码有没有开源？

数据集：https://tianchi.aliyun.com/dataset/dataDetail?dataId=107819 （只有少量）

代码：没开源

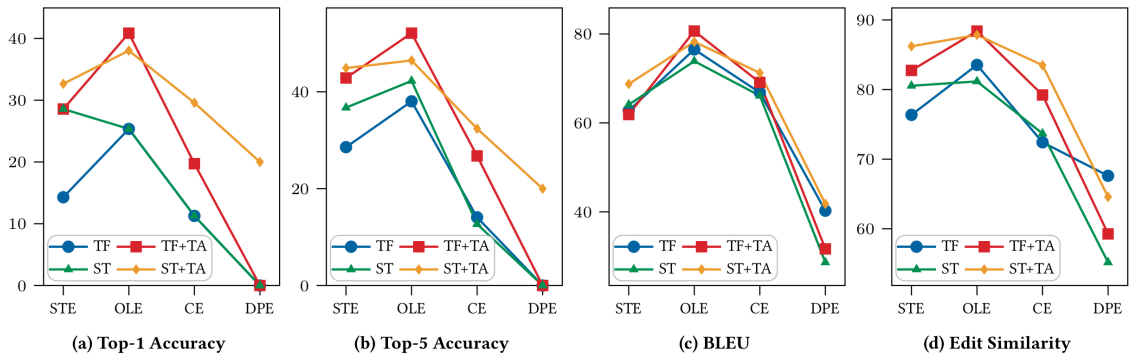
论文中的实验及结果有没有很好地支持需要验证的科学假设？

- 可以发现使用Task Augmentation的SubToken-Tran效果最好

Method	Acc-1	Acc-5	BLEU	EditSim
Transformer	16.84	26.02	67.98	77.31
TranX	17.35	22.45	67.56	76.20
Subtoken-TranX	20.41	29.08	66.77	77.63
Transformer+TA	29.08	39.29	69.72	82.92
TranX+TA	16.33	27.55	61.66	74.77
Subtoken-TranX+TA	**33.16**	**40.31**	**71.94**	**85.27**

Table 7: Main results of code generation

- 对于四种数据类型的比较：SubToken-TranX+task augmentation略胜于Transformer+task augmentation



(a) Top-1 Accuracy (b) Top-5 Accuracy (c) BLEU (d) Edit Similarity

Figure 2: Metrics of different categories on test data

- 各类辅助任务和没辅助任务的对比

Method	Acc-1	Acc-5	BLEU	EditSim
Transformer	16.84	26.02	67.98	77.31
Transformer+PT	26.53	38.27	69.91	82.47
Transformer+VP	25.00	37.24	68.82	82.38
Transformer+CG	29.08	39.29	69.72	82.92

Table 9: Results of different uses of variable semantic table

这篇论文到底有什么贡献？

1. 提出了一种真实环境下的代码生成方案，并公布了鲜有的数据集
2. 提出了一种任务增强方法，配合领域知识作为辅助任务，来生成代码
3. 提出了SubToken-TranX模型，支持subtoken级别的代码生成
4. 证明以上任务增强方法和SubToken-TranX模型的有效性，并将模型收入BizCook（阿里的代码生成方案集成）。目前可用于真实业务场景下的前端开发。

下一步呢？有什么工作可以继续深入？

目前在condition expression和data processing expression两类数据中，SubToken-X还表现不佳，是因为在Task Augmentation中，数据集只有“变量名-语义”数据对。

未来可以从代码库、需求文档那挖掘一下关于条件表述和数据处理API的语言描述，用本文类似的方法进行数据增强，可以进一步得到改善。