(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 108388425 A
(43)申请公布日 2018.08.10

(21)申请号 201810230691.2

(22)申请日 2018.03.20

(71)申请人 北京大学
地址 100871 北京市海淀区颐和园路5号

(72)发明人 李戈 金芝

(74)专利代理机构 北京辰权知识产权代理有限公司 11619
代理人 刘广达

(51)Int.Cl.
G06F 8/30(2018.01)
G06F 8/41(2018.01)

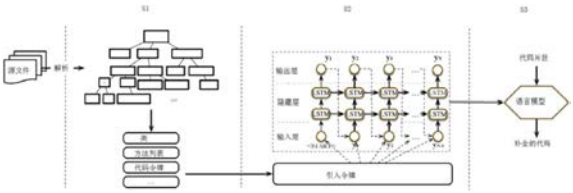
权利要求书1页 说明书6页 附图2页

(54)发明名称

一种基于LSTM自动补全代码的方法

(57)摘要

本发明提供了一种基于LSTM自动补全代码的方法，包括：源代码处理步骤，使用抽象语法树解析源代码；线下训练步骤，使用LSTM模型训练语言模型；线上代码补全步骤，根据训练过的语言模型补全代码。所述LSTM模型包括约束字符级LSTM和使用前上下文标识符编码器的标识符级LSTM。本发明实现了在编程过程中，任意地方输入任意字符都可以实现代码的自动补全，以及任意代码的推荐，并保证推荐过程的准确性。本发明的技术方案具有简单、快速的特点，能够较好地提高代码推荐的准确率和推荐效率。



CN 108388425 A

1.一种基于LSTM自动补全代码的方法,其特征在于,包括:

源代码处理步骤,使用抽象语法树解析源代码;

线下训练步骤,使用LSTM模型训练语言模型。

线上代码补全步骤,根据训练过的语言模型补全代码。

2.根据权利要求1所述的基于LSTM自动补全代码的方法,其特征在于:

在源代码处理步骤中,所述源代码被解析为不同形式,以获得代码的类、方法列表、代码标识符。

3.根据权利要求1或2所述的基于LSTM自动补全代码的方法,其特征在于:

所述LSTM模型包括约束字符级LSTM和使用前上下文标识符编码器的标识符级LSTM。

4.根据权利要求3所述的基于LSTM自动补全代码的方法,其特征在于:

使用LSTM模型引入解析源代码得到的标识符,并在不同场景中分别训练语言模型。

5.根据权利要求2所述的基于LSTM自动补全代码的方法,其特征在于:

所述LSTM模型为串联的两层LSTM模型,所述两侧LSTM模型位于隐藏层。

6.根据权利要求3所述的基于LSTM自动补全代码的方法,其特征在于:

所述约束字符级LSTM用于引入标识符以完成方法调用预测。

7.根据权利要求6所述的基于LSTM自动补全代码的方法,其特征在于:

所述方法调用预测的过程为:

加入约束,提取意图援引方法的对象和类;

通过遍历所述抽象语法树获得所有的类声明的方法;

预测方法名的第一个字符,并依次预测该方法的后续字符。

8.根据权利要求3所述的基于LSTM自动补全代码的方法,其特征在于:

所述使用前上下文标识符编码器的标识符级LSTM通过以下四种途径中的一种或多种编码标识符:

(1)索引,一个程序中不同位置的相同标识符代表相同的索引;

(2)类型加索引,将标识符的类型和索引结合使用;

(3)前标识符,通过评价一个、两个或三个前标识符来分别编码标识符;

(4)标识符ID,使用标识符ID替换所有标识符。

9.根据权利要求8所述的基于LSTM自动补全代码的方法,其特征在于:

引入所述标识符后,将所述源代码的序列输入到所述LSTM模型中,所述语言模型根据给定的部分程序的可能性分布来生成后续标识符。

10.根据权利要求1所述的基于LSTM自动补全代码的方法,其特征在于:

在线上代码补全步骤中,将部分代码片段输入已经训练过的语言模型,从而根据编程环境输出推荐的代码元素。

一种基于LSTM自动补全代码的方法

技术领域

[0001] 本发明涉及计算机软件工程技术领域，尤其是涉及一种基于LSTM自动补全代码的方法。

背景技术

[0002] 计算机自动生成代码是近年来软件工程的研究热点之一。代码自动生成极大的减少了程序员的工作量，提高了开发效率。随着开源社区的发展，我们可以通过分析大量的代码从而进行代码生成。代码自动生成的一大困难在于源代码本身具有诸多的约束和限制。近年来，在原有的基于组合优化方法进行程序综合研究的基础上，产生了一些新的基于机器学习技术进行程序生成的方法。

[0003] 按照所采取的技术及应用场景的不同，可将当前的程序生成方法分成两类：一类为基于程序输入输出结果的程序生成，一类为基于程序代码语言特性的代码生成。基于输入输出结果的程序综合主要基于机器学习模型，利用程序输入输出结果之间的对应关系构造训练数据集，并利用该数据集对机器学习模型进行训练，以达到在输入输出效果上模拟程序行为的目的。该类方法尤以基于深度神经网络的方法为代表。基于程序设计语言模型的程序生成主要利用程序设计语言自身所具有的的统计特性，通过对已有大量程序代码的学习建立相应程序设计语言的机器学习模型，并基于该模型在已有程序代码的基础上通过自动补全的方式生成新的代码。

[0004] LSTM(Long Short-Term Memory)是长短期记忆网络，是一种时间递归神经网络，适合于处理和预测时间序列中间隔和延迟相对较长的重要事件。LSTM已经在科技领域有了多种应用。基于LSTM的系统可以学习翻译语言、控制机器人、图像分析、文档摘要、语音识别图像识别、手写识别、控制聊天机器人、预测疾病、点击率和股票、合成音乐等等任务。

[0005] 中国发明专利申请号201710687197.4，涉及一种基于长短期记忆网络(LSTM)的代码推荐方法，针对现有代码推荐技术普遍存在推荐准确率低、推荐效率低等问题，该发明先将源代码提取成API序列，利用长短期记忆网络构建一个代码推荐模型，学习API调用之间的关系，然后进行代码推荐。并使用了dropout技术防止模型过拟合。同时提出运用ReLu函数代替传统饱和函数，解决梯度消失问题加快模型收敛速度，提高模型性能，充分发挥神经网络的优势。

[0006] 然而，上述专利实际上进行的是API推荐，与代码级推荐或者自动补全的目标仍有较大差距。不能实现在任意地点对任意代码的推荐。

[0007] 如图1所示，为现有技术中常见的代码自动补全方式。当在“accuracy＝tf”之后输入“.”时，会自动出现一个下拉菜单，编程人员可以选择例如“framework_lib”、“client_lib”等类名进行代码自动补全。然而，这种方式的缺陷在于：只有当用户输入“.”等特殊字符后才能够出现下拉菜单进行代码补全，无法实现在任意地点(例如输入任意一个字母时)进行代码补全或推荐；下拉菜单里推荐的仅仅是类名而非一段代码，仍然无法直接使用。

发明内容

[0008] 为解决以上问题，本发明提出深度自动代码生成，采用基于LSTM的引入标识符实现代码自动补全的任务，将训练语言模型用于从大规模代码集合中提取出的程序，预测代码元素。

[0009] 具体的，本发明提供了一种基于LSTM自动补全代码的方法，包括：

[0010] 源代码处理步骤，使用抽象语法树解析源代码；

[0011] 线下训练步骤，使用LSTM模型训练语言模型。

[0012] 线上代码补全步骤，根据训练过的语言模型补全代码。

[0013] 优选的，在源代码处理步骤中，所述源代码被解析为不同形式，以获得代码的类、方法列表、代码标识符。

[0014] 优选的，所述LSTM模型包括约束字符级LSTM和使用前上下文标识符编码器的标识符级LSTM。

[0015] 优选的，使用LSTM模型引入解析源代码得到的标识符，并在不同场景中分别训练语言模型。

[0016] 优选的，所述LSTM模型为串联的两层LSTM模型，所述两侧LSTM模型位于隐藏层。

[0017] 优选的，所述约束字符级LSTM用于引入标识符以完成方法调用预测。

[0018] 优选的，所述方法调用预测的过程为：

[0019] 加入约束，提取意图援引方法的对象和类；

[0020] 通过遍历所述抽象语法树获得所有的类声明的方法；

[0021] 预测方法名的第一个字符，并依次预测该方法的后续字符。

[0022] 优选的，所述使用前上下文标识符编码器的标识符级LSTM通过以下四种途径中的一种或多种编码标识符：

[0023] (1)索引，一个程序中不同位置的相同标识符代表相同的索引；

[0024] (2)类型加索引，将标识符的类型和索引结合使用；

[0025] (3)前标识符，通过评价一个、两个或三个前标识符来分别编码标识符；

[0026] (4)标识符ID，使用标识符ID替换所有标识符。

[0027] 优选的，引入所述标识符后，将所述源代码的序列输入到所述LSTM模型中，所述语言模型根据给定的部分程序的可能性分布来生成后续标识符。

[0028] 优选的，在线上代码补全步骤中，将部分代码片段输入已经训练过的语言模型，从而根据编程环境输出推荐的代码元素。

[0029] 本发明实现了在编程过程中，任意地方输入任意字符都可以实现代码的自动补全，以及任意代码的推荐，并保证推荐过程的准确性。本发明的技术方案具有简单、快速的特点，能够较好地提高代码推荐的准确率和推荐效率。

附图说明

[0030] 通过阅读下文优选实施方式的详细描述，各种其他的优点和益处对于本领域普通技术人员将变得清楚明了。附图仅用于示出优选实施方式的目的，而并不认为是对本发明的限制。而且在整个附图中，用相同的参考符号表示相同的部件。在附图中：

[0031] 图1为现有技术中自动补全代码的方法示例图；

[0032] 图2为本发明基于LSTM自动补全代码的方法流程图；

[0033] 图3为本发明使用约束字符级LSTM进行方法调用补全的示意图；

[0034] 图4为本发明基于LSTM自动补全代码的方法获得的代码自动补全结果示例图。

具体实施方式

[0035] 下面将参照附图更详细地描述本发明的示例性实施方式。虽然附图中显示了本发明的示例性实施方式，然而应当理解，可以以各种形式实现本发明而不应被这里阐述的实施方式所限制。相反，提供这些实施方式是为了能够更透彻地理解本发明，并且能够将本发明的范围完整的传达给本领域的技术人员。

[0036] 本发明公开了一种深度自动代码生成方法，基于长短期记忆网络(LSTM)的引入标识符而实现。深度学习的途径可以很好地捕捉有用特征并自动建立从输入到输出的映射。本发明的深度自动代码生成采用基于LSTM的引入标识符实现代码自动补全的任务。将训练语言模型用于从大规模代码集合中提取出的程序，预测代码元素。

[0037] 图2为本发明基于LSTM自动补全代码的方法流程图，包括如下步骤：

[0038] S1、源代码处理步骤，使用抽象语法树解析源代码。在这个步骤中，源代码被解析为不同形式，以用于各种途径。具体的，采用抽象语法树(Abstract Syntax Tree)来解析源代码，来获得代码的类、方法列表、代码标识符等等。

[0039] 抽象语法树(abstract syntax tree或者缩写为AST)，或者语法树(syntax tree)，是源代码的抽象语法结构的树状表现形式，这里特指编程语言的源代码。和抽象语法树相对的是具体语法树(concrete syntaxtree)，通常称作分析树(parse tree)。一般的，在源代码的翻译和编译过程中，语法分析器创建出分析树。一旦AST被创建出来，在后续的处理过程中，比如语义分析阶段，会添加一些信息。

[0040] S2、线下训练步骤，使用LSTM模型训练语言模型。

[0041] 使用不同途径引入解析得到的标识符，例如约束字符级LSTM和使用前上下文标识符编码器的标识符级LSTM，在接下来的子部分的在不同场景中分别训练。通过使用深度学习模型训练集合中的程序，例如图2所示的两层LSTM模型。以下重点介绍本发明所使用的约束字符级LSTM和标识符级LSTM。

[0042] 本发明所使用的约束字符级LSTM用于引入标识符以完成方法调用过程。由于使用频率高，方法调用的推荐是代码补全中的关键部分。很多集成开发环境(IDE)，例如Eclipse和IntelliJ，会在程序员在对一个对象后输入点字符“.”时自动列表出所有可用的成员函数。然后程序员可以选择列表中合适的方法进行调用。这些候选方法以字母表顺序列出，或者根据程序员的使用频率而排列。从列表中选择合适的候选方法非常耗费程序员的时间。为了进行更准确的推荐，本发明使用约束字符级LSTM以进行方法调用预测。基于LSTM模型的引入标识符是基础LSTM模型中的一个变量。

[0043] 图3为本发明使用约束字符级LSTM进行方法调用补全的示意图。该模型没有采用源代码的标识符，而是采用了代码字符作为输入。例如，输入序列是字符“browser.webBrowser”，并且它的独热矢量是X_1、X_2、……X_T。图3中h_i代表在当前时间戳下的LSTM单元的隐藏状态，其基于前一个LSTM单元的隐藏状态h_{i-1}而计算。最终，部分程序编

码为一个固定长度的矢量C。

[0044] 当一个字符一个字符地生成方法名称时，本发明将约束加入深度自动编码器。深度自动编码器提取意图援引方法的对象(图3中的Object)和类(Class)。然后可以通过遍历抽象语法树获得所有的类声明的方法。通过加入约束，将生成空间限制在这些可能的方法内。在产生的方法范围内，本发明预测方法名的第一个字符，并依次预测后续字符。由此，本发明根据概率列出了所有可能的候选方法。在预测过程的每一步都使用了LSTM，并且在每一步本发明的深度自动编码器将已产生的字符可能性进行分类。如图3所示，首先根据候选方法的可能性将其第一个字符分类为“d”、“e”、“j”，并根据相同的规则依次产生后续字符。最终，第一个方法名确认为“destroyFunction”，其是根据上下文环境寻找到的最合适方法。后面的方法名依次为“evaluateResult”、“jsEnabled”、“jsEnabledChanged”。

[0045] 本发明还使用前上下文标识符编码器的标识符级LSTM。在编程时，在任意可能的位置都能够进行代码补全是人工智能中进行代码补全的理想结果。它的实现难度远大于方法调用的补全。原因是方法调用空间被限制在特定类的声明方法里。大规模词汇量的生成对于LSTM来说是一个挑战。为了减小词汇量，本发明提出多个途径以引入标识符。这些途径的目标是将标识符与上下文环境结合编码。

[0046] 程序员经常根据上下文环境信息声明这些标识符。它们的文本信息在表达程序的语义时没有意义。因此，上下文信息能够将标识符的概念表述为更大的范围。本发明的深度自动编码器采用前上下文来编码标识符，并且极大减少了词汇量中用户自定义的标识符。本发明为不同的前上下文给出了经验结果以编码标识符。具体的，本发明给出了以下四种途径来编码标识符：

[0047] (1)索引。程序中的标识符表示为索引1,2,……,n。一个程序中不同位置的相同标识符代表相同的索引。例如，代码片段“for(int i;i<100;i++)”表示为“for(int ID_1;ID_1<100;ID_1++)”。

[0048] (2)类型加索引。将标识符的类型和索引结合起来。因此，上面的代码可以表示为“for(int INT_1;INT_1<100;INT_1++)”。通过加入标识符类型，既可以通过位置区分标识符，又可以通过类型区分标识符。

[0049] (3)前标识符。本发明中，可以通过评价一个、两个或三个前标识符来分别编码标识符。

[0050] (4)标识符ID。为了评价标识符级LSTM的上界精度，本发明的深度自动编码器使用标识符ID来替换所有标识符。上面的代码片段表示为“for(int ID;ID<100;ID++)”。这种编码方法不关心标识符之间的差异。并且通过将源代码作为自然语言处理，本发明能够在任意可能位置都给出代码补全。

[0051] 引入标识符后，代码序列输入到两层LSTM模型中。语言模型根据给定的部分程序的可能性分布来生成后续标识符。

[0052] S3、线上代码补全步骤，根据训练过的语言模型补全代码。在这个步骤中，将部分代码片段输入已经训练过的语言模型，从而根据特定的编程环境输出推荐的代码元素。

[0053] 图4为本发明基于LSTM自动补全代码的方法获得的代码自动补全结果示例图。其中在该编译器环境中，采用本发明所使用的自动补全代码方法，每输入一个任意字符，例如字母、“.”、“=”、“_”、“,”“(”等字符后，在其下面(深灰色部分)会出现若干行(行数根据训

练结果而不定)推荐的代码,例如最下面的8行代码是自动补全的代码。如果这些推荐的代码是用户想要的代码行,则直接输入空格键即可确认;如果这些推荐的代码不是用户想要的代码行,则用户继续输入自己想要的代码的下一个字符即可,同理输入下一个字符后编译器仍然会继续推荐若干行代码作为预测的代码,这些代码可能和上一次推荐的代码相同,也可能不同。如此循环往复,直到完成整个程序代码的编译。

[0054] 从上述过程可以看出,本发明实现了在编程过程中,任意地方输入任意字符都可以实现代码的自动补全,以及任意代码的推荐,并且由于采用了LSTM模型训练了多个场景,因此能够保证推荐过程的准确性。本发明的技术方案具有简单、快速的特点,能够较好地提高代码推荐的准确率和推荐效率。

[0055] 需要说明的是:

[0056] 在此提供的算法和显示不与任何特定计算机、虚拟装置或者其它设备固有相关。各种通用装置也可以与基于在此的示教一起使用。根据上面的描述,构造这类装置所要求的结构是显而易见的。此外,本发明也不针对任何特定编程语言。应当明白,可以利用各种编程语言实现在此描述的本发明的内容,并且上面对特定语言所做的描述是为了披露本发明的最佳实施方式。

[0057] 在此处所提供的说明书中,说明了大量具体细节。然而,能够理解,本发明的实施例可以在没有这些具体细节的情况下实践。在一些实例中,并未详细示出公知的方法、结构和技术,以便不模糊对本说明书的理解。

[0058] 类似地,应当理解,为了精简本公开并帮助理解各个发明方面中的一个或多个,在上面对本发明的示例性实施例的描述中,本发明的各个特征有时被一起分组到单个实施例、图、或者对其的描述中。然而,并不应将该公开的方法解释成反映如下意图:即所要求保护的本发明要求比在每个权利要求中所明确记载的特征更多的特征。更确切地说,如下面的权利要求书所反映的那样,发明方面在于少于前面公开的单个实施例的所有特征。因此,遵循具体实施方式的权利要求书由此明确地并入该具体实施方式,其中每个权利要求本身都作为本发明的单独实施例。

[0059] 本领域那些技术人员可以理解,可以对实施例中的设备中的模块进行自适应性地改变并且把它们设置在与该实施例不同的一个或多个设备中。可以把实施例中的模块或单元或组件组合成一个模块或单元或组件,以及此外可以把它们分成多个子模块或子单元或子组件。除了这样的特征和/或过程或者单元中的至少一些是相互排斥之外,可以采用任何组合对本说明书(包括伴随的权利要求、摘要和附图)中公开的所有特征以及如此公开的任何方法或者设备的所有过程或单元进行组合。除非另外明确陈述,本说明书(包括伴随的权利要求、摘要和附图)中公开的每个特征可以由提供相同、等同或相似目的的替代特征来代替。

[0060] 此外,本领域的技术人员能够理解,尽管在此所述的一些实施例包括其它实施例中所包括的某些特征而不是其它特征,但是不同实施例的特征的组合意味着处于本发明的范围之内并且形成不同的实施例。例如,在下面的权利要求书中,所要求保护的实施例的任意之一都可以以任意的组合方式来使用。

[0061] 本发明的各个部件实施例可以以硬件实现,或者以在一个或者多个处理器上运行的软件模块实现,或者以它们的组合实现。本领域的技术人员应当理解,可以在实践中使用

微处理器或者数字信号处理器(DSP)来实现根据本发明实施例的虚拟机的创建装置中的一些或者全部部件的一些或者全部功能。本发明还可以实现为用于执行这里所描述的方法的一部分或者全部的设备或者装置程序(例如,计算机程序和计算机程序产品)。这样的实现本发明的程序可以存储在计算机可读介质上,或者可以具有一个或者多个信号的形式。这样的信号可以从因特网网站上下载得到,或者在载体信号上提供,或者以任何其他形式提供。

[0062] 应该注意的是上述实施例对本发明进行说明而不是对本发明进行限制,并且本领域技术人员在不脱离所附权利要求的范围的情况下可设计出替换实施例。在权利要求中,不应将位于括号之间的任何参考符号构造成对权利要求的限制。单词“包含”不排除存在未列在权利要求中的元件或步骤。位于元件之前的单词“一”或“一个”不排除存在多个这样的元件。本发明可以借助于包括有若干不同元件的硬件以及借助于适当编程的计算机来实现。在列举了若干装置的单元权利要求中,这些装置中的若干个可以是通过同一个硬件项来具体体现。单词第一、第二、以及第三等的使用不表示任何顺序。可将这些单词解释为名称。

[0063] 以上所述,仅为本发明较佳的具体实施方式,但本发明的保护范围并不局限于此,任何熟悉本技术领域的技术人员在本发明揭露的技术范围内,可轻易想到的变化或替换,都应涵盖在本发明的保护范围之内。因此,本发明的保护范围应以所述权利要求的保护范围为准。

```
W_fc2 = weight_variable([1024,10])
b_fc2 = bias_variable([10])
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2)+b_fc2)

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y_conv),
                                              reduction_indices=[1]))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.arg_max(y_conv,1), tf.arg_max(y_, 1))
accuracy = tf.
    framework_lib          tensorflow.python.framework.framework_lib
    client_lib             tensorflow.python.client.client_lib
    print_function                                      __future__
    app                    tensorflow.python.platform.app
    array_grad             tensorflow.python.ops.array_grad
    attr_value_pb2         tensorflow.core.framework.attr_value_pb2
    clip_ops               tensorflow.python.ops.clip_ops
    common_shapes          tensorflow.python.framework.common_shapes
    confusion_matrix_m     tensorflow.python.ops.confusion_matrix
    confusion_matrix(labels, …  tensorflow.python.ops.confusion_matrix
    ^↓ and ^↑ will move caret down and up in the editor >>
tf.global_v
tf.client_l
for i in ra
    batch =
    if i%10
        tra
        pri
    train_s
                                                    _prob: 1.0},
```

图1

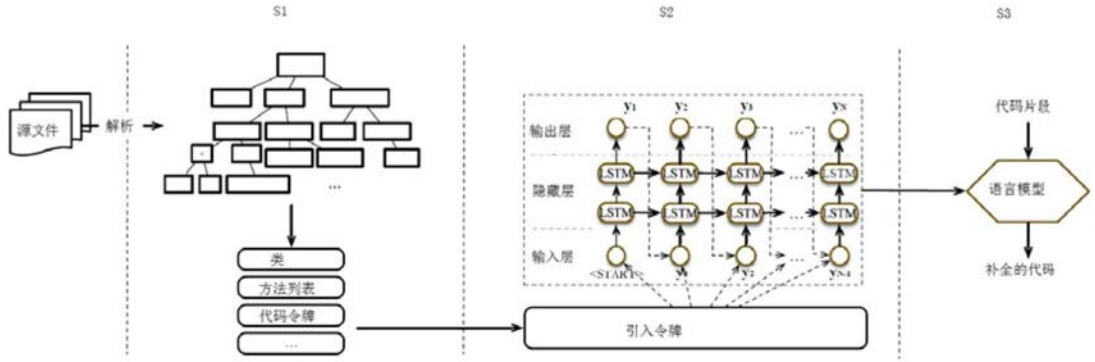


图2

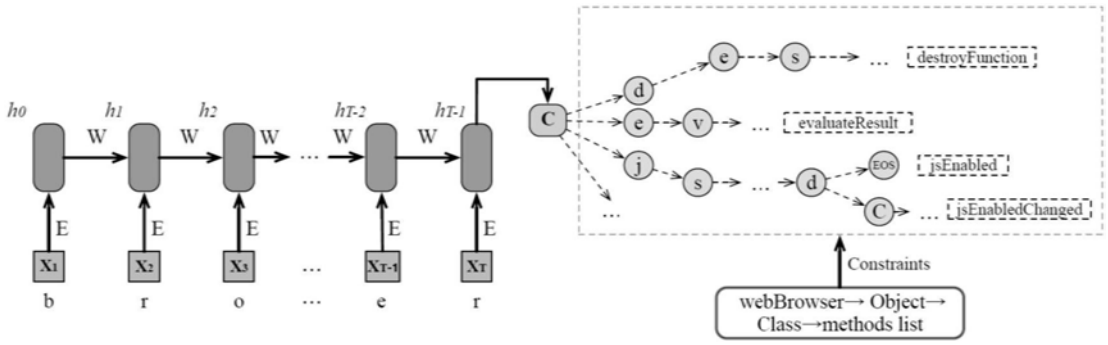


图3

```
import numpy as np
import tensorflow as tf
from tensorflow.python.ops import rnn_cell
from tensorflow.python.ops import seq2seq
import sys
import os

class Model():
    def __init__(self, args,infer = False):
        self.args = args
        if infer:
            args.batch_size = 1
            args.seq_length = 1
        if args.model == 'rnn':
            cell_fn = rnn_cell.BasicRNNCell
        elif args.model == 'gru':
            cell_fn = rnn_cell.GRUCell
        elif args.model == 'lstm':
            cell_fn = rnn_cell.BasicLSTMCell
        else:
            raise Exception('model type not supported: {}'.format(args.model))

        cell = cell_fn(args.rnn_size, state_is_tuple = True)
        self.cell = cell = rnn_cell.MultiRNNCell( [cell] * args.num_layers,state_is_tuple = True)
        self.input_data = tf.placeholder(tf.int32, [args.batch_size, args.seq_length])
        self._targets = tf.placeholder(tf.int32, [args.batch_size, args.seq_length])
        self.initial_state = cell.zero_state(args.batch_size,tf.float32)

        with tf.variable_scope('rnnlm'):
            softmax_w = tf.get_variable('softmax_w', [args.rnn_size,args.vocab_size])
            softmax_b = tf.get_variable('softmax_b', [args.vocab_size])
            with tf.device('/cpu:0'):
                embedding = tf.get_variable('embedding',[args.vocab_size, args.rnn_size])
                inputs = tf.split(1, args.seq_length, tf.nn.embedding_lookup(embedding,self.input_data))
                inputs = [tf.squeeze(input_, [1]) for input_ in inputs]
                for layer in inputs [ <int> : ] :
                    with tf . variable_scope ( <str> ) :
                        a = tf . add ( <UNK> , <UNK> , name = <str> )
                    return regularizers , t
                def <UNK> ( <UNK> , <UNK> ) :
                    with tf . name_scope ( <str> ) :
                        image_size = IMAGE_SIZE / / <int>
                        output = tf
```

自动补全的代码

图4