《人工神经网络》大作业开题报告

李欣隆*
2018010324
计92
xl-li18@mails.tsinghua.edu.cn

陈炫中
2019011236
计92
xz-chen19@mails.tsinghua.edu.cn

甘乔尹
2019011240
计92
ganqy19@mails.tsinghua.edu.cn

1 选题

我们的选题是一个自选任务：探究神经网络在基于例子的表达式求值任务上的表现。

1.1 任务定义

表达式求值是一类推理问题，其中含有许多不同的具体任务，对每个具体任务，我们会给出表达式的语法和语义的定义。

1.2 数据集

计划采用的数据集为我们自己设计的基于例子的表达式求值数据集，数据集中包含许多不同的具体任务。对每种任务有对应该任务的一些数据，数据是以例子的形式给出的。可以对每个具体任务训练神经网络并测试，也可以将所有数据一起训练。

以下按直观上从简单到难的顺序列出表达式求值任务的一些例子。

0. 简单字符串操作表达式求值

这个任务考虑字符串拼接（concat）、字符串翻转（rev）、将第一个字符串翻转后和第二个字符串拼接（rev-concat）三个操作。

使用上下文无关文法描述这个任务需要考虑的表达式:
expr ::= "concat" expr expr | "rev-concat" expr expr | "rev" expr | data
data ::= "data" str
str ::= "0" str | "1" str | <空串>

表达式expr 的值是递归定义的:
eval("data" str)=str
eval("concat" expr1 expr2) 定义为eval(expr1) 和eval(expr2) 的字符串拼接
eval("rev-concat" expr1 expr2) 定义为eval(expr1) 翻转后和eval(expr2) 的字符串拼接
eval("rev" expr1)=eval("rev-concat" expr1 ("data"))

任务的输入是表达式expr，输出是字符串表示eval(expr)。

1. 布尔表达式求值

*组长

使用上下文无关文法描述这个任务需要考虑的表达式:
expr ::= "and" expr expr | "or" expr expr | "xor" expr expr | "not" expr | data
data ::= "data" "0" | "data" "1"

表达式expr的值eval(expr)是递归地定义的:
eval("data" "0") = 0
eval("data" "1") = 1
eval("and" expr1 expr2) = min(eval(expr1),eval(expr2))
eval("or" expr1 expr2) = max(eval(expr1),eval(expr2))
eval("xor" expr1 expr2) = (eval(expr1)+eval(expr2))eval("not" expr) = 1-eval(expr)

任务的输入是表达式expr，输出是0或1表示eval(expr)的值。

2. 自然数加法和乘法

这个任务考虑二进制表示的整数的加法和乘法:
expr ::= "+" expr expr | "*" expr expr | number
number ::= "data" "0" | "data" positive
positive ::= "0" positive | "1" positive | "1"

表达式expr的值eval(expr)是递归地定义的:
eval("data" "0") = 0
eval("data" positive) = eval(positive)
eval("0" positive) = 2eval(positive)
eval("1" positive) = 2eval(positive)+1
eval("1") = 1
eval("+" expr1 expr2) = eval(expr1)+eval(expr2)
eval("*" expr1 expr2) = eval(expr1)+eval(expr2)
eval("xor" expr1 expr2) = (eval(expr1)+eval(expr2))%2 eval("not" expr) = 1-eval(expr)

3. 变量代入

这个任务是任务2的扩展，需要按顺序对若干个表达式求值，并把结果存储在指定的变量中，这些变量可能出现在之后的表达式中，最后只需输出最后一个表达式的值。
Expr ::= def Expr | expr
def ::= "def" var expr
var := "var" number
expr ::= "+" expr expr | "*" expr expr | number | var
number ::= "data" "0" | "data" positive
positive ::= "0" positive | "1" positive | "1"

表达式的值的定义可以从任务2扩展得到:

eval("var" number) 定义为编号eval(number)的变量的值，保证这个变量已经存在

按顺序执行每个def，对于"def" ("var" number) expr2，执行eval(expr2)后将结果保存在编号为eval(number)的变量中

在所有def之后还有一个expr，执行eval(expr)后将结果作为输出

4. lambda演算

expr ::= "lambda" var expr | "apply" expr expr
var ::= "a" | "b" | "c" | ... //为了简单，这里只支持有限个变量

eval("lambda" var expr)="lambda" var expr
eval("apply" expr1 expr2)=apply(eval(expr1),expr2)
apply(var,x)="apply" var x
apply("lambda" var expr,x)=expr[var:=x]

("lambda" var expr)[a:=b] = "lambda" var expr[a:=b]，若var不为a
("lambda" var expr)[a:=b] = "lambda" var expr，若var为a
("apply" expr1 expr2)[a:=b] = "apply" expr1[a:=b] expr2[a:=b]
var[a:=b] = var，若var不为a
var[a:=b] = b，若var为a

数据需要保证这个计算是停机的，且按定义计算时，递归的总步数小于某个常数。

数据的生成：

前面已经给出了上下文无关文法描述的每种数据，再加上每个规则的选取概率，得到概率上下文无关文法，即可用于生成数据集，另外需要去除不符合语义规范的数据。

随机生成了训练集后，再同分布地生成测试集，但去除训练集中已经存在的样本。另外可以调整概率参数，生成一些单组数据的平均规模更大的测试集，用于测试模型是否能泛化到更大的递归深度。

模型评价方式：

对任务0,1,2,3，采用全文比较的方式判断结果是否正确，评价指标为测试集错误率。对任务4，如果仅通过保持语义不变地替换变量名能使得模型的输出和答案一致，则视为这组数据正确，评价指标为测试集错误率。

1.3 基线结果

对于这个任务，目前没有已知基线，我们准备采用普通的transformer 模型作为基线，其中encoder 输入表达式，decoder 输出表达式的值。

表达式和表达式的值被编码为序列，序列的元素以双引号标识的词为单位。

2 研究计划

我们的任务的重点在于通过例子使得模型推断出一个合理的方法来解决数据集中的符号推理任务，并且主要关注模型在递归深度上的泛化能力。按照预期，对于数据集中的部分任务，transformer 模型能有好的表现，对于其他一些任务，我们测试的所有神经网络模型都不会达到很好的表现。

2.1 相关研究

目前与我们的任务相关性较大的研究有四类：

1. 使用语言模型求解应用题，如OpenAI 最近研究了使用微调后的GPT-3 模型解决小学数学题[1]。他们的任务和我们的共同点是都使用了带中间结果标注的数据来辅助模型学习，区别是我们的任务是符号推理任务，不需要自然语言；我们的任务更关注递归深度上的泛化能力，而他们的任务中数字都较小。

2. 使用seq2seq 模型求解简单微积分[2] 等表达式求值类问题，他们的任务和我们的共同点是都属于符号推理任务，不需要自然语言。他们的任务和我们的区别是我们的任务更关注递归深度上的泛化能力；而他们的任务中系数都非常小，原文中描述为$-10, 10$ 之间的整数。

3. 使用seq2seq 模型实现自然语言到代码的映射，如使用带有抽象语法树先验的transformer 模型来实现自然语言到代码的翻译[3]。他们的任务更偏向于机器翻译任务，需要有一个较为清晰的自然语言描述，才能将该描述直接翻译成代码，而我们的模型侧重于通过给出例子的中间结果来学习。

4. 使用带有启发式的与或图搜索框架实现基于例子的程序生成，如[4]。他们的数据集中每个任务有一些例子，例子中指名一些允许使用的基本操作函数，以及一些形如$f(x_1, x_2, ...) = y$ 的限制，模型需要搜索到一份只允许使用给定的基本操作函数的代码，使得能通过限制。

他们的任务和我们的共同点是都属于符号推理任务，不需要自然语言。他们的任务和我们的区别是这一类搜索框架中一部分框架是不可学习的，另一部分只能学习搜索一个分支的

Example 5 (Programming by Examples (PBE) with Strings)**.** Consider the following example:

```
(set-logic PBE_SLIA)
(synth-fun f ((fname String) (lname String)) String
  ((y_str String) (y_int Int))
  ((y_str String (" " fname lname
                   (str.++ y_str y_str)
                   (str.replace y_str y_str y_str)
                   (str.at y_str y_int)
                   (str.from_int y_int)
                   (str.substr y_str y_int y_int)))
     (y_int Int (0 1 2
                  (+ y_int y_int)
                  (- y_int y_int)
                  (str.len y_str)
                  (str.to_int y_str)
                  (str.indexof y_str y_str y_int)))))
(constraint (= (f "Nancy" "FreeHafer") "Nancy FreeHafer"))
(constraint (= (f "Andrew" "Cencici") "Andrew Cencici"))
(constraint (= (f "Jan" "Kotas") "Jan Kotas"))
(constraint (= (f "Mariya" "Sergienko") "Mariya Sergienko"))
(check-synth)
```

Figure 1: 字符串拼接的例子

概率，学习能力较差；他们的搜索框架不需要搜索出递归的代码（如循环和递归的函数调用），只需要利用给定的基本操作，这些基本操作中有一部分是支持传入参数后递归的；同时为了实现与或图搜索，他们需要人工设计的针对给定的基本操作的反函数。

2.2 我们的任务

对于表达式求值的一组数据，输入X 和输出Y 都是序列，使用transformer 模型，有几种处理方式如下：

1. 无额外空间的处理

encoder 输入X，decoder 输出Y [end]，对应基线方法。

此时模型必须将所有计算过程在|X|+|Y|步中完成，我们期望模型无法泛化到更大规模的数据，以及输出长度短而计算过程长的情况。

2. 有额外空间的处理

encoder 输入X，decoder输出S [sep] Y [end]，其中S不含[sep]，[sep] 是一个特殊的字符，用于分割答案和用于计算的额外空间。

我们期望模型能利用S 做为额外的存储空间，表示表达式求值的中间结果，从而有更好的泛化能力。

3. 有过程提示的处理

encoder输入X，decoder输出S [sep] Y [end]，S 由人或程序生成标注。

我们期望模型能利用S，按类似于标注的方式输出表达式求值的中间结果，且通过显式地标注的中间结果，希望提高模型的泛化能力；

另外，通过不同方式给出S 的标注也可能可以影响模型的表现。

2.3 挑战

1. 对数据集中的任务0,1,2,3，可以直接用全文比较的方式计算测试集正确率，而任务4由于需要考虑到变量名不同的影响，需要特殊设计训练和测试的方式。

2. 需要合理设计数据集中数据的分布，避免数据集退化为较为平凡的情况（例如布尔表达式的短路计算导致大部分表达式不影响结果，lambda 演算不同的数据所需计算步数相差很大）。

3. 对于研究计划2的方案，需要控制模型使用合适的长度的额外空间，避免花费不可接受的时间/空间，也避免其退化为接近基线的不使用额外空间的情况。

4. 可能需要新增除任务0,1,2,3,4外的相似任务，探索transformer 表现较差的情况。

2.4 可行性

transformer 模型被普遍用在使用序列作为输入输出的推理任务中。

数据集可以调节大小，任务需要观察计算资源对效果的影响，可以在短时间进行大量对比实验，与transformer 模型对比的模型可以使用RNN, LSTM, NTM（神经图灵机）等。

预期的结果是随着表达式求值的任务难度增大，transformer 的表现逐渐变差，而不是完美解决所有表达式求值任务（达到零的测试集错误率），RNN 与LSTM 模型在短的串上表现不错，随着串变长表现迅速变差。

组员对任务有足够经验，本学期有足够精力。

3 算力估计

申请并行科技的算力，可能之后需要申请额外时间。

参考文献

References

[1] Cobbe, K., V. Kosaraju, M. Bavarian, et al. Training verifiers to solve math word problems. vol. abs/2110.14168. 2021.

[2] Lample, G., F. Charton. Deep learning for symbolic mathematics. 2020.

[3] Sun, Z., Q. Zhu, Y. Xiong, et al. Treegen: A tree-based transformer architecture for code generation. pages 8984–8991, 2020.

[4] Ji, R., Y. Sun, Y. Xiong, et al. Guiding dynamic programing via structural probability for accelerating programming by example. *Proc. ACM Program. Lang.*, 4(OOPSLA):224:1–224:29, 2020.