编译实习 Project - Part II

罗煜楚 夏逸宽 武俊宏 那天行

0 任务目标

对于一个给定的表达式$Output=expr(Input_1, Input_2, ..., Input_n)$（$Output, Input_i$是张量或标量, $expr()$表示用其参数构造一个表达式），已知最终 loss 对于$Output$的导数$dOutput = \frac{\partial loss}{\partial Output}$，求解 loss 对于某个输入的导函数。

1 自动求导技术设计

首先定义语法树的节点属性：inh 和 val。inh 表示一个继承属性，用作导数求值，其在开始的时候是未知的，根据后面给出的 SDD 求解。val 是在进行 json 文本分析的时候得到的变量值。

SDD 的定义如下：

```
S ::= LHS = RHS { RHS.inh = LHS.inh }
RHS ::= RHS1 + RHS2 { RHS1.inh = RHS.inh; RHS2.inh = RHS.inh; } |
        RHS1 - RHS2 { RHS1.inh = RHS.inh; RHS2.inh = - RHS.inh; } |
        RHS1 * RHS2 {
            RHS1.inh = RHS2.val * RHS.inh;
            RHS2.inh = RHS1.val * RHS.inh;
            } |
        RHS1 / RHS2 {
            RHS1.inh = RHS.inh / RHS2.val;
            RHS2.inh = - (RHS1.val * RHS.inh) / (RHS2.val * RHS2.val);
            } |
        RHS1 // RHS2 { RHS1.inh = 0; RHS2.inh = 0; } |
        RHS1 % RHS2 { RHS1.inh = RHS.inh; RHS2.inh = 0; } |
        (RHS1) { RHS1.inh = RHS.inh; } |
        TRef { TRef.inh = RHS.inh } |
        SRef { SRef.inh = RHS.inh } |
        Constant { do nothing }
```

这里使用 case1 来进行可行性和正确性的论述。case1 的 kernel 为：“C<4, 16>[i, j] = A<4, 16>[i, j] * B<4, 16>[i, j] + 1.0”，并且是对 A 进行求导。那么期望的结果为 dA<4, 16>[i, j] = dC<4, 16>[i, j] * B<4, 16>[i, j]。使用 SDD 进行分析：

① 首先原式 RHS 为 RHS1 + constant 结构，故 A<4, 16>[i, j] * B<4, 16>[i, j]对应的 RHS1 有 RHS1.inh = RHS.inh=LHS.inh。(之后使用的“RHS1”与①中意义不同)

② 对式子 A<4, 16>[i, j] * B<4, 16>[i, j]使用“RHS ::= RHS1 * RHS2 { RHS1.inh = RHS2.val * RHS.inh; RHS2.inh = RHS1.val * RHS.inh;}”分析，对于 A<4, 16>[i, j]对应的 RHS1，RHS1.inh = RHS2.val * RHS.inh，对于 B<4,16>[i, j]对应的 RHS2，RHS2.inh = RHS1.val * RHS.inh。

③ ②中的 RHS1 和 RHS2 都将使用“RHS ::= TRef { TRef.inh = RHS.inh }”进行推导。那么对于 A，对应的 TRef 有 TRef.inh = RHS1.inh = RHS2.val * RHS.inh，也就是：

$$A.inh = (A*B).inh * B.val$$

即：

$$dA<4, 16>[i, j]=d(A<4, 16>[i, j] * B<4, 16>[i, j]) * B<4, 16>[i, j]$$

由此可知所得到的结果就是我们所期待的 dA<4, 16>[i, j] = dC<4, 16>[i, j] * B<4, 16>[i, j]，故举例论述完毕。

2 算法流程

① 对输入 case 的 json 文法进行分析，构建 json 语法树。

② 通过构建的 json 语法树获取 RHS 的 val 属性。

③ 对建立的 json 语法树进行深度优先遍历，使用 1 中设计的自动求导技术的 SDD，最终求出语法树中的所有 TRef 和 SRef 的 inh 属性，即求导属性。

④ 对于待求导矩阵 A，如果 A 为张量则遍历所有对应的 TRef 节点，如果 A 为标量则遍历所有对应的 SRef 节点：

a. 对于 LHS 是简单表达式的情况下，即形如 $A<c_1, c_2, \cdots, c_k> [i_1, i_2, i_3, \cdots, i_k]$不存在下标运算的情况下，直接生成如下语句：

$$dA<c_1, c_2, \cdots, c_k>[i_1, i_2, i_3, \cdots, i_k] \mathrel{+}= TRef.inh$$

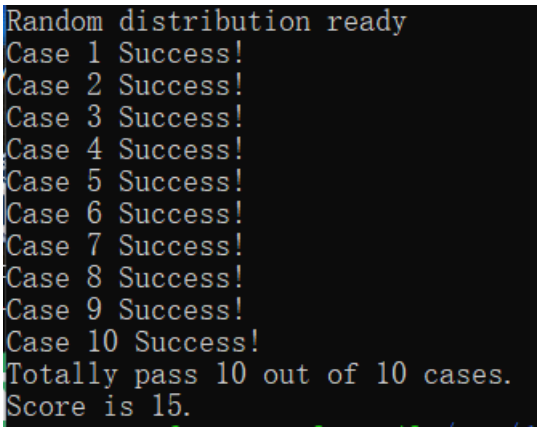
b 对于 LHS 是复杂表达式的情况下，对每一个 LHS 中带有运算的下标，引入一个全新下标进行替代，并在式子右侧添加迪利克雷函数进行判断。譬如如果最后生成的式子为 dA[p+q]=dC[p]*B[q]，引入新下标，并生成下述表达式：

$$dA[z] = if(p + q == z)\ dC[p] * B[q]$$

⑤ 将④中生成的语句（可能不止一个）按顺序拼接在一起，调用 project1 中已经实现的代码生成工具（这里在 project2 中进行了修改，添加了对于“+=”运算的处理），即可生成最终的求导代码。

3 实验结果

最终的实验结果为顺利通过 10 个测试点。



4 总结

在代码生成工具的设计和实现中：在第一阶段读入输入表达式生成中间代码的抽象语法树阶段，我们在词法和文法分析方面将输入表达式的文法分为了五个层次，自上而下分别为

表达式组、总体表达式、不可分割部分、值引用以及索引和形状五层，每一层提供给更高层信息（综合属性）并向更底层传递信息（继承属性）。在这样的分析和语义结构下，我们获得了中间表达形式，通过维护一个存放已构建的 IR Node 的节点栈和一个存放尚未处理符号的符号栈，最终构建出了所需的抽象语法的表达式树。在第二阶段使用 visitor 遍历第一阶段生成的抽象语法树，在相关 IR 节点处产生输出生成最终代码。

在自动求导算法的设计和实现中：我们使用语法制导翻译辅助进行了语法分析，通过定义一个求导的继承属性和一个传值的综合属性最终实现了一个可行并正确的语法制导翻译方案，利用实现的 SDT 将最初的 json 文本转化成了求导形式的输入文本，之后直接调用已实现的代码生成工具最终获得求导的 c++代码。

5 分工

罗煜楚：自动求导技术设计与实现，代码生成工具修改。
夏逸宽：自动求导技术设计与实现。
武俊宏：自动求导技术设计，代码生成工具维护。
那天行：自动求导技术设计，代码生成工具修改，报告撰写。