编译第 38 组 project2 设计报告

陈学智、高玉泉、邵俊宁

June 21, 2020

1 自动求导技术设计

1.1 理论推导

问题描述：对于一个给定的表达式 $Output = expr(Input_1, Input_2, ..., Input_n)$, 已知最终 loss 对于 Output 的导数 $dOutput = \frac{\partial loss}{\partial Output}$, 求 $dInput_i = \frac{\partial loss}{\partial Input_i}$。我们可以计算得到下式（表达遵从爱因斯坦求和约定）：

$$dInput_i = \frac{\partial loss}{\partial Input_i} = \frac{\partial loss}{\partial Output}\frac{\partial Output}{\partial Input_i} = dOutput\frac{\partial Output}{\partial Input_i}$$

1. 当 $Output$ 和 $Input_i$ 为线性关系时，即 $Output = kInput_i + b$ 时，$\frac{\partial Output}{\partial Input_i} = k$, 带入得到求导结果为

$$dInput_i = dOutput \cdot k$$

2. 当 $Output$ 和 $Input_i$ 为幂函数关系时，即 $Output = kInput_i^m$, 此时 $\frac{\partial Output}{\partial Input_i} = kmInput_i^{m-1}$, 带入得到求导结果为

$$dInput_i = dOutput \cdot kmInput_i^{m-1}$$

3. 当 $Output$ 和 $Input$ 的指标存在平移关系时，即 $Output[i] = Input[i + k]$, 此时 $\frac{\partial Output}{\partial Input} = I(t = i - k)$,t 为求和指标，相当于输入和输出存在常数项的平移，此时求导结果可以表示为

$$dInput[i + k] = dOutput[i]$$

1.2 样例

以 case1 为例，$C < 4, 16 > [i, j] = A < 4, 16 > [i, j] * B < 4, 16 > [i, j] + 1.0$; 此时是满足条件 1 的线性关系，因此结果为 $dA < 4, 16 > [i, j] = dC < 4, 16 > [i, j] * B < 4, 16 > [i, j]$; 由此可见，在这种情况下，输出的结果只要忽略常数项，再交换 Input 和 Output 再分别微分就可以得到结果。

2 实验流程及实验结果

本生成器主要分为四个模块：

2.1 输入输出模块

主要是读取并解析 json 文件，并将运行结果输出到目标文件。使用了 [JSON for Modern C++](https://github.com/nlohmann/json) 进行 json 解析，该库接口比较简洁，并且只需要包含一个头文件 'json.hpp'就可以使用。运行时会逐个读取./cases 下所有文件，将 kernel 项的值写入临时文件 tmp 供词法分析器读取。

2.2 词法/文法分析模块

读取原始表达式并生成一棵抽象语法树。使用了 Lex/Yacc 工具来生成词法分析程序 lex.yy.cc 和文法分析程序 parser.tab.cc，语法树的节点是 Node 基类的派生类，在 syntaxtree.hpp 中定义，每个非终结符号和终结符号都拥有一个 Node* 类型的综合属性 yylval，表示其指向的语法树节点对象，不同类型的符号对应不同的语法树节点类型，在自底向上分析过程中，每识别出一个符号（终结或者非终结）时，新建一个该类型的节点，并将子树的节点信息插入到新节点上，最终构成一棵以代表整个函数的 Function 类型节点为根节点的抽象语法树。

2.3 IR 树生成模块

根据生成的抽象语法树，翻译成一棵 IR 树用于后续的输出。主要分为两步，首先是数组/循环范围的确定，然后是 IR 树的生成。

范围确定是这样实现的：每条赋值语句拥有一个 IndexTable，存储该语句中出现在数组下标中的变量的最小范围，整个函数用一个 TrefTable 记录出现过的数组的范围。使用 checkRange 函数深度优先遍历语法树来寻找张量类型的节点，每个张量节点查找 TrefTable，如果尚未记录同名数组的范围，则添加进去，之后调用每个下标表达式节点的 checkRange 函数。下标表达式的 checkRange 函数与上层节点不同，有一个 range 参数，表示当前表达式的最大允许值，向下一层传递时，如果当前表达式是表达式 + 运算符 + 整数的形式，就根据运算符和整数进行一次逆运算算出下一层表达式的最大允许值; 否则直接向下传递。最终传递到变量名节点时，每个变量查找 IndexTable，若记录的循环范围大于当前最大允许值，则更新 IndexTable 中的记录值。张量节点向下标表达式节点传递的最大允许范围则是该数组对应维度的大小。

由 1.1 节的讨论我们可以知道，如果假定 *Input* 与 *Output* 是线性关系，那么可以忽略常数项，交换 *Input* 和 *Output* 的位置再分别微分，就能得到所求的求导表达式。反映到 AST 的处理上，就是要将所有表示 *Input* 的结点替换成 *dOutput*，所有表示 *Output* 的结点替换成 *dInput*。

具体来说，在 `checkRange` 函数执行的过程中，每遇到一个 Tref 结点 A，如果之前没有为它生成过对应的导数结点，那么就为它生成一个导数结点 dA，导数结点除了 `name` 不同之外，其他成员变量都与原结点相同。然后，在调用 `makeExpr` 生成相应的 IR 树结点时，会根据当前 *Output* 和被求导的 *Input*，按照上文提到的替换规则生成对应的导数结点。最后，为每个被求导的项生成求导的表达式，并生成正确的函数签名。对于满足线性关系的情况来说，求导函数的输出是被求导的项的导数，而输入是原函数输出项的导数和原函数除被求导项之外的输入项。如果有多个被求导项，那么确定输入项时要分别计算，最后取并。

IR 树的生成也是通过原语法树的深度优先遍历再向上综合得到的，通过调用定制的 make 接口体系，从最底层的单个符号开始组装 IR 树的节点并返回上一层，在赋值语句节点处组装循环结构，最后在函数节点处完成整个函数体的组装，调用 IR 树翻译模块得到结果。

2.4 IR 树翻译模块

将给出的 IR 树翻译为 C/C++ 源代码。基本建立在给出的代码上，通过修改 ‘IRPrinter.cc‘中的各个 ‘visit‘函数，使其能够按照 C/C++ 的语法规范进行输出。同时修改了 ‘IR.h‘中部分类的 ‘make‘函数和 IRMutator.cc‘中的对修改过的 ‘make‘函数的调用，因为生成 C/C++ 代码有时需要比原有代码获取更多的信息，需要重新定制 make 函数。

3 使用到的编译知识

词法和语法分析工具的使用。项目使用了 Lex/Yacc 工具对输入程序进行词法和语法分析，生成抽象语法树。

SDD/SDT。通过遍历 AST 生成 IR 树，通过 IRVisitor，IRMutator 和 IRPrinter 对 IR 树进行操作，本质上是实现了 SDT 的功能。

4 小组分工

模块一：高玉泉、陈学智
模块二：陈学智
模块三：邵俊宁、高玉泉、陈学智
模块四：高玉泉
写设计报告以及最终 debug 的部分三人都有参加。