编译第二次 project

徐德嘉 1700013024 sirius.caffrey@gmail.com
李泳民 1700012846 liyongmin@pku.edu.cn
韩昱 1700012921 vickyhan@pku.edu.cn
张天远 1600012888 tianyuanzhang@pku.edu.cn

1. 算法概览

具体分工：
自动求导技术设计：张天远，李泳民，徐德嘉，韩昱
具体实现：李泳民

2. 自动求导技术总述

我们的算法分两步：

- **梯度树生成**: 根据链式法则，对 project1 生成的 IR 进行一次 pass Tree 生成梯度公式（以 IR Tree 的形式）
- **梯度树翻译**: 再次调用 project1 的 printer 将梯度树变为 c 代码

即我们大量复用了 project1 中的代码，只对原表达式的抽象语法树进行了操作。

3. 梯度树生成

梯度树的生成也就是，根据 project1 的语法树来生成梯度的公式。我们要做的事情就是对这个语法树的每个 Expr 类的叶子结点求出其梯度公式。我们知道了根部节点的梯度，就能根据链式法则将根结点的梯度流传到每个叶子结点。我们对整颗语法树进行深度优先搜索，visit 每个结点的时候，为其各个儿子结点求得梯度（如果有的话），然后自顶向下递归。在这个过程中，有两个问题我们认为比较重要

- **梯度公式问题**: 节点的算子不同，梯度公式不同
- **梯度下标问题**：如何处理下标问题
- **梯度顺序问题**: 梯度运算结合律的问题，也就是 A*B*C 这类问题谁先计算谁后计算

3.1. 梯度公式问题

递归时，需要根据父节点的运算符进行分类讨论，具体如下：

- 父节点为: Expr C = Expr A:
  dA = C.grad

- 父节点为: Expr C = Expr A - Expr B
  dA = C.grad; dB = -C.grad
- 父节点为: Expr C = Expr A * Expr B
  dA = C.grad * B; dB = C.grad * A
- 父节点为: Expr C = Expr A / Expr B
  dA = C.grad / B dB = - C.grad * A / (B * B)

3.2. 梯度下标问题

另外对于梯度公式的下标难处理的问题，我们归纳后发现，根本不用对下标进行特殊的转换操作，只要初始化的时候把梯度初始化成 0，之后下标只需简单地将采用原下标，同时对每个下标的循环变量范围稍加注意即可。

举一个具体例子，在 case6 中，需要求导的式子为：

A<2, 8, 5, 5>[n, k, p, q] = B<2, 16, 7, 7>[n, c, p + r, q + s] * C<8, 16, 3, 3>[k, c, r, s];

给出的答案为：

dB<2, 16, 7, 7>[n, c, h, w] = select((h - p >= 0) && (w - q >= 0) && (h - p < 3) && (w - q < 3), dA<2, 8, 5, 5>[n, k, p, q] * C<8, 16, 3, 3>[k, c, h - p, w - q], 0.0);

而我们生成的结果为：

tmp0[n][c][(p + r)][(q + s)] = (tmp0[n][c][(p + r)][(q + s)] + (dA[n][k][p][q] * C[k][c][r][s]));

此处 tmp0 对应的就是 dB，而 dB 在一开始会初始化成 0。可以看出，数组下标对应替换之后，我们生成的结果和给出的答案逻辑上完全一致，而我们的下标则巧妙地直接采用了需要求导式子的下标，避免了下标的二次运算。说明了我们直接采用原下标的可行性和正确性。

3.3. 梯度顺序问题

我们采用类似回填的方式来处理梯度顺序问题。我们先举一个例子来说明什么是梯度顺序的问题。

比如 D = (A*B)*C，那么 dA 应当为 (B*C)dD，如果我们 dfs 遍历一遍语法树，然后按照上文说的规则来计算各个子节点的梯度，则可能会出现这样的状况：先计算 dAB = C* dD，然后才会计算 dA = B * (C *dD)，这样子会带来数值偏差。为了解决这个问题，我们发现只要永远根据树的结构，自底向上地结合运算即可获得正确的运算顺序。具体地说，在这个例子里，传入梯度的 Expr 为 dD*_，然后把 _ 替换成 C*_，再传给第一个乘号，在第一个乘号的 Expr 中将 _ 修正为 C*_ 传给 B，最后 B 把 _ 替换成 1. 这样就是一个类似回填的算法，最终产生结果为 dD*(C*(B*1))。

该例子也说明了我们自动求导技术的可行性和正确性。

4. 梯度树翻译

生成了梯度树之后只用调用第一次 project 的 printer 则可以翻译，需要注意的事情有两点

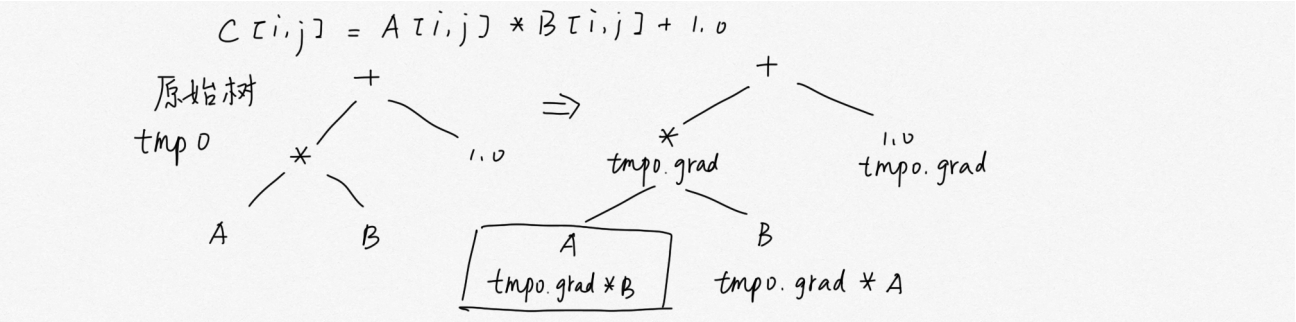
- ins, outs 的顺序
- 梯度初始化为全 0

5. 一个具体例子

以 case1

C<4, 16>[i, j] = A<4, 16>[i, j] * B<4, 16>[i, j] + 1.0;

对 A 求导为例。其具体运算过程如下图所示，对每个叶子结点向下求出梯度，在需要求导的节点处的梯度就是最终结果。



6. 总结

本 project 中我们主要运用了编译课程中学到的抽象语法树的运算，结点之间信息的传递，同时还采用了“回填”的思想解决了梯度顺序的问题。

A. 如何实现

为了实现上述算法，我们先调用第一次 project 的代码解析 json 表达式。然后对右边的每个 Einstein 求和式，我们为其中每次被求导变量的出现新建一个形状相同的临时变量临时承载其求导结果。

具体的计算方法是使用 IRVisitor 递归地 visit 每个节点，同时在 visitor 内保存必要的信息，例如到此为止的导数等。当发现被求导变量出现时将该变量与其导数一同保存。

我们采用的类似回填的技术，在实现中即将接下来需要进一步求导的位置用 Expr() 先临时代替，然后递归到子节点时替换所有这样的“空位”为结合了父节点的导数的 Expr 传给子节点。在发现目标变量时将所有“空位”替换为 1 即可。