编译大作业

第二部分——自动求导的编译器

1. 前言

在第一部分的作业中，我们做的事情是根据输入的表达式生成C/C++代码，并且在10个例子上测试正确性（6个公开，4个隐藏）。此时，每位同学手头都应该有一个可用的代码生成器了。回忆我们做这个project的初衷，我们想要做一个面向当前重要应用——深度学习——的代码生成工具，利用我们编译课上学习的知识完成这一任务。在第一部分中，我们体会了词法分析、语法分析、中间表示形式（IR Node）、语法树构建、语法树遍历（通过IRVisitor）和代码生成，并且还可能用到了少数SDD, SDT中的知识。我们第二次project将继续这个方向，利用编译技术做更多有趣的功能，这一次，我们的重点将放在语法树的变换（一个个pass）上来，对于变换的设计可能会用到课本上更多的知识（但不一定是严格局限课本的例子，同学们可以根据实际情况活用）。这一次project可能对于一些同学来说比较困难，希望通过小组合作，大家都能掌握这个过程中需要的知识和技术。

2. 问题描述

2.1 传统的深度学习框架求导

自动求导是深度学习中当前必不可少的功能（依赖于梯度优化的算法都摆脱不了求导过程）。在深度学习框架中（如Tensorflow, PyTorch），自动求导都是由框架完成的，它们的方法论是，首先形成计算图，然后根据链式法则构建计算梯度的图。举一个例子，一个简单的计算过程为：

```
X is a tensor of shape [4, 3, 28, 28]
T is a label of shape [4, 8 * 28 * 28]
Y1 = Conv2d(X, kernel=(8, 3, 3, 3), padding=1, stride=1) # result shape is [4, 8, 28, 28]
Y2 = flatten(Y1)  # result shape is [4, 8 * 28 * 28]
loss = mse_loss(Y2, T)  # loss is scalar
```

如果想要求出对于X的导数（虽然常见情况是对于网络参数求导，而不是网络输入，但这里只是做一个例子），就要从loss开始求，首先loss对于自己的导数是1，然后求Y2的导数，框架发现Y2用来计算loss时，使用的时mse_loss函数，于是找到了mse_loss函数的导函数grad_mse_loss，用于计算Y2的导数；接着对于Y1，框架又发现Y2是通过flatten函数求出来的，于是找到了flatten函数的导函数grad_flatten，利用这个导函数求出对于Y1的导数，继续向上求X的导数，框架又发现了Conv2d层，于是找到了对应的卷积求导函数，用于求X的导数。可以看到，这个过程除了链式法则，框架还在不断地识别正向传播时使用的函数/层的名字，然后在自己的函数库里寻找对应的导函数，框架知道应该找哪个

导函数，都是依赖于编写框架的人了解这些知识，然后在框架的库里准备好需要的函数们。
这是一种传统的求导方式，它的粒度是算子（加减乘除也算是算子），而在我们这次project中，我们将使用编译技术自动地根据前向算子计算定义生成其反向计算导数的函数，整个过程，不需要特意知道算子的名称，只需要看到数学表达式即可。这样做的一个优势是，深度学习应用的可扩展性将被加强，当有人希望自己设计一个算子时，他/她不再需要自己推导导函数的定义，然后自己实现出来再注册到框架里使用，而是只需要提供一个正向传播的计算表达式，就可以得到对应的导函数计算定义。

2.2 问题定义

现在我们开始进行问题描述：
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），我们如果已知了最终loss对于$Output$的导数$dOutput = \frac{\partial loss}{\partial Output}$，我们想知道loss对于某个输入的导函数是什么，也就是求$dInput_i = \frac{\partial loss}{\partial Input_i}$的问题，**我们要求必须从表达式IR层面分析出求导的表达式，而不能根据case的名字硬编码答案。**

2.3一个例子

为了帮助理解，我们给一个例子：

```
C<M, N>[i, j] = A<M, K>[i, k] * B<K, N>[k, j]
```

基于第一次project的知识，我们知道这个式子表达了一个矩阵乘法。
现在已知了某个$loss$对于C的导数dC（是个张量，大小与C的大小相同，注意这里的dC是个名字，不是算符），假设想要求dA，那么根据求导的数学方法得到

$$dA[i,k] = \frac{\partial loss}{\partial A[i,k]} = \sum_j \frac{\partial loss}{\partial C[i,j]} \times \frac{\partial C[i,j]}{\partial A[i,k]} = dC[i,j] \times B[k,j]$$

所以可以得到对于A的导数计算式为

```
dA<M, K>[i, k] = dC<M, N>[i, j] * B<K, N>[k, j]
```

翻译为C代码就是

```
for (int i = 0; i < M; ++i) {
  for (int k = 0; k < K; ++k) {
    dA[i][k] = 0.0;
    for (int j = 0; j < N; ++j) {
      dA[i][k] += dC[i][j] * B[k][j];
    }
  }
}
```

对于B也可以类似写出求导的式子：

```
dB<K, N>[k, j] = dC<M, N>[i, j] * A<M, K>[i, k]
```

3. Project输入与输出

基于上一次project的测试法，我们这次仍然给出10个例子，与第一次project的测试case不同的是，我们给出的json文件中多了一个"grad_to"的键值，这个键值的信息是对哪个/哪些输入（可能一个或多个输入）进行求导。
比如看case1的json文件：

```
{
  "name": "grad_case1",
  "ins": ["A", "B"],
  "outs": ["C"],
  "data_type": "float",
  "kernel": "C<4, 16>[i, j] = A<4, 16>[i, j] * B<4, 16>[i, j] + 1.0;",
  "grad_to": ["A"]
}
```

这里指明了对于A进行求导，所以得到的式子应该是

```
dA<4, 16>[i, j] = dC<4, 16>[i, j] * B<4, 16>[i, j]
```

这里的dC符号就是C的导数，我们认为所有的输出的导数张量都是已知，命名规则都是原来的名字前面加个d，此外，我们只考虑正向传播表达式有且仅有一个输出的情况。
同学们读如json文件，分析正向表达式后，根据编译技术分析出反向传播表达式，然后对这个反向传播的表达式生成C/C++代码，放在kernels/目录对应的文件内。每次cmake这个project时，都会先自动运行solution下的代码，然后运行run2.cc，run2.cc里有测试逻辑，会测试同学们生成的反向传播代码的正确性。

3.1 测试例子

这次一共10个测试例子，全部是公开的，公开理由为：

- 自动求导本身蕴含NP问题，只有下标的变换满足一定条件（如线性）才是易解的，即使是易解的，其求解细节也比较复杂，所以给出具体的10个例子，同学们不必花费太多精力担心输入不可预测性。
- 隐藏例子测试法（第一次project）是防止同学们通过打表法做题，只给出trivial的解决方案（比如直接输出字符串），这个问题可以通过设计审查方法来杜绝（审查法在后面介绍）
- 并非所有同学都有求导的先验知识，所以给出所有例子方便同学们掌握问题，更好地完成任务

这10个例子都是紧贴实际深度学习应用的，涵盖的实际应用包括：

1. element-wise的乘法
2. 矩阵乘法
3. dense MTTKRP
4. 二维普通卷积
5. 转置
6. flatten
7. broadcast
8. blur

考虑到并非所有同学都接触过求导方法，我们提供了ground truth。每个测试例子在run2.cc里都会有一个对应地测试函数，在函数体以及注释里，都可以获取正确求导的结果。比如对case1，test_case1函数为

```
bool test_case1(std::mt19937 &gen, std::uniform_real_distribution<float> &dis) {
    // "C<4, 16>[i, j] = A<4, 16>[i, j] * B<4, 16>[i, j] + 1.0;"
    // "dA<4, 16>[i, j] = dC<4, 16>[i, j] * B<4, 16>[i, j];"
    float B[4][16] = {{0}};
    float dA[4][16] = {{0}};
    float dC[4][16] = {{0}};
    float golden[4][16] = {{0}};
    // initialize
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 16; ++j) {
            B[i][j] = dis(gen);
            dC[i][j] = dis(gen);
        }
    }
    // compute golden
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 16; ++j) {
            golden[i][j] = dC[i][j] * B[i][j];
        }
    }
    try {
        grad_case1(B, dC, dA);
    } catch (...) {
        std::cout << "Failed because of runtime error\n";
        return false;
    }

    // check
    for (int i = 0; i < 4; ++i) {
        for (int j = 0; j < 16; ++j) {
            if (std::abs(golden[i][j] - dA[i][j]) >= 1e-5) {
                std::cout << "Wrong answer\n";
                return false;
            }
        }
    }
    // correct
    return true;
}
```

可以看注释，或者golden的记算方法，来学习正确的求导结果。

4. 评分与要求

4.1 关键日期

project2 开始：2020年5月16日中午12:00
project2 截至：2020年6月21日中午12:00

不接受补交，请及时提交文件，并检查是否提交成功

4.2 提交途径与要求

4.2.1 途径

以小组为单位，分组沿袭第一次project的分组，小组成员最终得分一致。
提交代码有两种途径：

- 发送压缩包到邮箱compiler2020spring@163.com
- 发送github链接到邮箱compiler2020spring@163.com（推荐）

4.2.2 要求

1. 必须包含一个pdf版本的报告在project2目录下，报告内容必须涵盖小组分工，自动求导技术设计，实现流程，实验结果。其余内容可根据个人爱好添加。
2. 不可以更改/拷贝run2.h, run2.cc, clean2.cc的内容，其余内容均可自由改动
3. 不要从stdin读取内容，请从json文件读取输入
4. 使用编译器版本需要兼容C++11标准（gcc 4.8.5以上应该都满足）

如果选择发送源代码压缩包，请把必要的源文件都放在压缩包中，不要包含build文件，要保证运行代码仅仅需要以下命令：

```
mkdir build
cd build
cmake ..
make -j 4
cd project2
./test2
```

如果依赖第三方项目（以及非C++11标准自带的功能），要想办法满足这个要求（比如把第三方项目代码作为project2的子项目一起打包进压缩包）。

如果选择发送github链接，请一定保证在提交截止日期后代码仓是public的，这样助教有权限下载代码（但也要注意不要提前public了，以防有人抄代码）。助教的测试命令为：

```
git clone --recursive <提交的github链接> CompilerProject
cd CompilerProject
mkdir build
cd build
cmake ..
make -j 4
cd project2
./test2
```

4.3 评分法

本次project占总成绩20%，这20分中5分来自pdf报告，15分来自提交代码。
pdf评分标准：

- 包含小组分工，自动求导技术设计（2分）
- 包含实现流程，实验结果内容（1分）
- 通过一个具体例子解释所设计的求导技术的可行性和正确性（1分）
- 总结使用到的编译知识，讲解如何实现（1分）

代码评分标准：

- 根据通过的case数目计算分数(真实分数)，具体按照下表

通过case数目	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
得分	0	2.25	4.5	6.75	9	10.5	12	12.75	13.5	14.25	15

- 在运行./test2后会自动打出分数，符合上表的定义

4.4 审查法

审查代码是为了防止同学作弊，作弊的定义包含：

- 拷贝或修改run2.h/run2.cc/clean2.cc的内容
- 不使用任何编译分析技术直接输出字符串到kernels/目录下
- 任何两组的代码重合度过高甚至完全一致
- 完全使用第三方项目解决问题
- 报告内容与实际实现不一致

可能使用的审查法包括

1. 全面审查法，共38个队伍，3名助教平均每人审查13支队伍的代码
2. 抽样审查法，随机挑选x只队伍进行代码审查
3. 输入测试法，助教随机修改输入case的json文件，如果仍然能输出有意义的代码到kernels/下（助教会看），说明没有作弊（不测试代码正确性，只考察能生成代码）
4. 结合报告内容审查法，检查报告中的技术和实现的技术一致性。
5. 重点审查法，对于得分高于12分的组进行全面代码审查。

具体审查方法不公开，请同学们自觉使用编译课所学知识解决问题，我们的给分策略是非线性，通过4个例子的组都可以得到9分以上（60%的分数），过6个例子就可以得到12分（80%的分数），宗旨是鼓励大家侧重于自己设计编译器，而不是唯分数论。

5. 参考文献与代码

1. Halide的一个自动求导工作
   https://people.csail.mit.edu/tzumao/gradient_halide/gradient_halide.pdf
2. Halide自动求导代码
   https://github.com/halide/Halide/blob/master/src/Derivative.cpp
3. TVM自动求导代码
   https://github.com/apache/incubator-tvm/pull/2498
4. 二维卷积反向传播推导
   https://zhuanlan.zhihu.com/p/61898234

6. 讨论

Project可能潜在的bug可以在微信群、github issue上提出，有价值的issue可以为全组加分，每个bug加1分
另外，鼓励小组内部协作与讨论，也鼓励适当的小组间交流，交流方式为github issue或微信群，助教也会参与讨论，解答一些技术问题。