编译技术第二次 project 报告

谷文睿 叶思炜 凌钰明 龚子清

一. 自动求导总体技术设计：

首先，我们希望结合数学公式以及给定的例子来找到这些求导之间的共性。以 C=A*B 为例，$\mathrm{dA}=\frac{\partial \mathrm{loss}}{\partial \mathrm{A}}=\frac{\partial \mathrm{loss}}{\partial \mathrm{C}}\frac{\partial \mathrm{C}}{\partial \mathrm{A}}=dC*B$，这里的 B 并不一定只是一个矩阵，也可能是其他与 AC 无关的矩阵的运算，这并不影响结果。这样，对于一个矩阵的赋值 L=R，可以将其写为$\mathrm{L}=\mathrm{A}*\mathrm{X}+\mathrm{B}$的形式,其中 A*X 表示赋值右面含有要求导的 X 矩阵的项，而 B 则表示不含有 X 的项，这样，dX 即可表示为 dL*A（若 A 中没有 X）。进一步地，若 A 中有 X，则

L=A+B*X+C*X*X+…，A,B,C…中均不含有 X，此时$\mathrm{dX}=\frac{\partial \mathrm{loss}}{\partial \mathrm{X}}=\frac{\partial \mathrm{loss}}{\partial \mathrm{L}}\frac{\partial \mathrm{L}}{\partial \mathrm{X}}=$

$$\frac{\partial \mathrm{loss}}{\partial \mathrm{L}}\frac{\partial \mathrm{A}}{\partial \mathrm{X}}+\frac{\partial \mathrm{loss}}{\partial \mathrm{L}}\frac{\partial (\mathrm{BX})}{\partial \mathrm{X}}+\frac{\partial \mathrm{loss}}{\partial \mathrm{L}}\frac{\partial (\mathrm{C*X*X})}{\partial \mathrm{X}}+\cdots=dL*B+dL*C*X+dL*C*X+\cdots$$

由以上的推导，我们发现一项有趣的事实，若我们等式左端为 L，要求导的矩阵为 X，那么，取出等式右边与 X 相关的部分并逐个将 X 替换为 dL 之后相加的结果即为 dX。这就是我们本次 project 中自动求导的总体技术设计。

二. 实现流程：

根据上面的总体技术设计，我们需要记录下来各个变量的名字，后面所跟循环变量的表达式以及范围，这个变量的位置，这个变量所处的乘法运算式的头部位置（因为求导可以看作以乘法为单位）以及这个变量所处的括号的首位。我们扫描一遍原 kernel 表达式并记录这些信息。

```
while (pos < len) {
        if (resource[pos] >= 'A' && resource[pos] <= 'Z') {
            arr[++cnt].name = resource[pos];
            arr[cnt].bracket = flag;
            arr[cnt].pos = pos;
            arr[cnt].last = last;
            char temp[10];
            pos += 2;
            cpos = 0;
            int num = 0;
            while (1) {
                if (resource[pos] == ',') {
                    temp[cpos] = 0;
                    arr[cnt].range[++num] = atoi(temp);
                    pos++;
                    cpos = 0;
                }
                else if (resource[pos] == '>') {
                    temp[cpos] = 0;
                    arr[cnt].range[++num] = atoi(temp);
                    arr[cnt].range[0] = num;
```

```
                    pos++;
                    break;
                }
                else temp[cpos++] = resource[pos++];
            }
            pos++;
            cpos = 0;
            num = 0;
            while (1) {
                if (resource[pos] == ',') {
                    temp[cpos] = 0;
                    memcpy(arr[cnt].var[++num], temp, 10);
                    pos++;
                    cpos = 0;
                }
                else if (resource[pos] == ']') {
                    temp[cpos] = 0;
                    memcpy(arr[cnt].var[++num], temp, 10);
                    pos++;
                    break;
                }
                else temp[cpos++] = resource[pos++];
            }
        }
        else {
            if (resource[pos] == '(') {
                flag = pos;
                last=cnt+1;
            }
            else if (resource[pos] == ')') {
                flag = -1;
                last=cnt+1;
            }
            else if (resource[pos] == '+' || resource[pos] == '=') last
= cnt + 1;
            pos++;
        }
    }
```

接下来需要进行求导变换。我们根据 json 信息中获得的 gradto 也就是要求导的变量名称来寻找名称相同的变量。然后对于找到的同名变量，我们获得这个变量的乘法区间，并将这个变量替换为表达式左面的变量，接下来对于下标也需要进行替换。我们采取的方式是这样的：为了方便我们将要求导的矩阵的各个下标参数的名字设为 w,x,y,z，例如原来是 A[i+1,j]，我们将其改为

dA[w,x]，那么这样，对于其他变量中如果下标参数含有 i 或 j 的，就需要将 i 变为 w-1，j 换为 x，再根据 w，x 的范围计算出变化后的范围。同时，我们还有计算出这个式子可以被计算的条件 cond，计算方式是比如这个变量的下标参数被改成了类似于 w-1 的表达式，那么这个表达式的取值要落在原下标的取值范围内，即(w-1 >=0 && w-1<range(i))?value[w-1]:0 。这样，我们就获得了一个乘法项要被改成的表达式，最后将所有含有求导项的乘法项得到的这些表达式用“+”或“-”连起来，即得到了我们核心的计算表达式。

随后是对于循环变量的处理，我们取出最终的计算表达式中涉及到的变量，对于这些变量下标中含有的下标参数逐层进行 0-range 的循环，最后在循环的最内层加入计算表达式即可。对于函数签名的获取，首先是最终计算需要的非 d 变量，然后是计算需要的 d 变量（一般是原表达式左值的 d 值），然后是需要计算的 d 值，每种内部按字母序排列。最终实验结果：

```
gurrypku@master:~/CompilerProject-2020Spring-master/build/bin$ ./test2
Random distribution ready
Case 1 Success!
Case 2 Success!
Case 3 Success!
Case 4 Success!
Case 5 Success!
Case 6 Success!
Case 7 Success!
Case 8 Success!
Case 9 Success!
Case 10 Success!
Totally pass 10 out of 10 cases.
Score is 15.
```

三. 具体例子解释：

为了解释我们的整个方法，我们以 case6 为例

// "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];"

// "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);"

上面是给的原表达式，下面是正确结果的表达式。以下是我们的生成代码：

```
#include "../run2.h"
void grad_case6(float (&C)[8][16][3][3],float (&dA)[2][8][5][5],float (&dB)[2][16][7][7]){
        for (int w=0;w<2;++w)
                for (int x=0;x<16;++x)
                        for (int y=0;y<7;++y)
                                for (int z=0;z<7;++z)
                                        for (int k=0;k<8;++k)
                                                for (int r=0;r<3;++r)
                                                        for (int s=0;s<3;++s)
                                                                dB[w][x][y][z]+=(w>=0 && w<2 && y-r>=0 && y-r<5 && z-s>=0
&& z-s<5 && x>=0 && x<16?dA[w][k][y-r][z-s]*C[k][x][r][s]:0.0);
}
```

扫描一遍后，根据上面的方法，在右面找到了要求导的 B，将 dB 下标变为 w,x,y,z，将左面的 A 加 d 替换到原来 B 的位置。此时是 dA[n][k][p][q]*C[k][c][r][s]，接下来，因为 w=n,x=c,y=p+r,z=q+s，故右面的变量额下标要被替换为 dA[w][k][y-r][z-s]*C[k][x][r][s]，其中，w，y-r，z-s 和 x 因为是替换后的下标，需要按照原来的 n,p,q 和 c 检查范围。最终生成的计算表达式中含有的下标参数有 w,x,y,z,k,r,s，按其单独出现的范围生成循环范围。最终生成的代码就如图所示。

四.　用到的编译知识：

具体用到的知识包括词法分析和语法分析生成符号表的过程，生成的变量节点和计算符可以看成是语法树，进行转化的过程可以看成 SDT，只不过这里没有用到中间代码而是直接处理成了最终代码而已。这些知识的实现在上面的实现中都有体现，这里不详细指出。

五.　小组成员分工：

叶思炜：表达式转化及代码生成

龚子清，凌钰明：词法分析和语法分析，获取变量表，debug 帮助

谷文睿：理论部分以及报告撰写