5. 深度学习编译器-图优化

前面介绍过，现在的深度学习框架/编译器是通过数据流图的方式表示神经网络计算。在此基础上，可以实现各种优化-将数据流图从一种形式“等价”转换为另一种行式, 提高该数据流图在硬件上的运行性能，我们将其称为图优化。本节主要讨论一些常用的图优化方法。

- **5.1 常量折叠 (Const Folding)**

常量折叠是传统编译器中的一种优化手段, 它的原理是如果一个计算所依赖的所有输入都是常量, 则在编译期间就可以得到计算结果. 比如对于下面的代码:

```
int main() {
    const int a = 100;
    int b = a + 100;
    printf(“b is %d\n”, b);
    return 0;
}
```

b 的值只依赖于 a 的值, 由于 a 为常量, 其取值在编译阶段就确定了, 因此也可以在编译阶段计算得到 b 的值, 所以上面的代码经过常量折叠优化后,等价于下面的代码:

```
int main() {
    printf(“b is %d\n”, 200);
    return 0;
}
```

深度学习编译器中的常量折叠和传统编译器是类似的, 只需将输入变为 Tensor 即可. 比如对于下面的一个网络:

```
A = const([3, 5], 1.0, fp32)
B = const([3, 5], 0.5, fp32)
C = var([5, 4], fp32)
D = dot(A + B, C)
```

通过常量折叠后, 就可以变为:

```
TMP = const([3, 5], 1.5, fp32)
C = var([5, 4], fp32)
D = dot(TMP, C)
```

常量折叠的思想比较简单, 下面讨论下常量折叠的具体实现方法。对于给定的神经网络 N, 假设其中所有 OP 的集合为 O, 对 O 进行拓扑排序后得到的 OP 列表为 TO。 对于 OP a, 假设 input(a) 返回该 OP 对应的输入 OP 列表。下面是实现常量折叠的伪代码:

```
is_const = map(Op, bool)
for o in TO:
    if o is const_op:
        is_const(o) = true
        continue
   else:
        foldable = true
        for io in input(o):
            if is_const(io) = false:
                foldable = false
                break
        if (foldable):
            is_const(o) = true
            update(o)
```

```
        else:
            is_const(o) = false
```

参考上面的代码, 逐个遍历所有 OP, 并记录该 OP 是否为常量。 如果一个 OP 是常量 OP 或者它的所有输入均为常量, 则可以推断该 OP 的结果也是常量, 因此可以直接计算得到该 OP 的值。由于遍历过程是按拓扑排序进行的, 可以保证在访问一个 OP 之前, 该 OP 的所有输入都已被访问过. 也即如果输入中有常量, 在访问该 OP 时, 这些常量的值已经被计算出来了。通过这种方式, 只需要遍历一遍 OP, 就可以计算出所有可折叠的常量, 以及他们的值。

5.2 公共子表达式消除 (Common Sub-Expression Elimination)

在一个程序中, 如果几个表达式的类型、参数和输入均相同, 则将他们称做公共子表达式。 对于公共子表达式, 只需要计算其中一个表达式的值, 其他表达式的值可以通过赋值得到。这个过程就称作公共子表达式消除, 它是一种传统编译器中常用的优化手段, 经过迁移也可以应用到深度学习编译器中。

公共子表达式实现的伪代码如下:

```
!Initialize(NN, O) // O 表示所有 OP 的集合
TopologySort(O, TO)
Map<OP_TYPE, list<OP>> map
Map<OP, OP> results
for o in TO:
    if (len(input(o)) == 0:
        results[o] = clone(o)
        continue
    candidates = map.find(o.op_type)
    if (len(candidates) == 0):
        map[o.op_type].append(o)
    else:
        find = false
        for co in candidates:
            if (! is_same_subexpr(o, co)):
                continue
            else:
                find = true
                result[o] = co
                break
        if (!find):
            map[o.op_type].append(o)
           result[o] = clone(o)
return
```

上述代码的基本思路是通过一个 MAP 表, 记录截止当前, 已处理过的同一种类型的 OP。 对于当前正在处理的 OP, 先查找该 MAP 表, 如果能找到其他和正在处理的 OP 类型相同的 OP, 则对他们进行遍历, 如果其中某个 OP 的输入和参数与当前正在处理的 OP 相同, 则它们为公共子表达式, 结果可以互相替代；如果所有 OP 都不能与当前正在处理的 OP 匹配, 则将当前 OP 复制一份返回。