4. 深度学习编译器-自动微分

本节介绍自动微分，之所以先介绍自动微分，一是因为这是深度学习训练过程中必不可少的一步，二是在 Tensorflow等深度学习框架中，使用自动微分之后，反向计算梯度的过程也是以 OP 的形式集成到整个计算图里，因此后续优化就可以将推理和训练统一起来考虑，不用再做单独处理。

背景介绍

在许多应用中需要计算函数的微分，流体动力学、最优控制、物理建模等。比如通过梯度下降法求解优化问题，

通过计算机求微分值的方法包括以下几种：（1）通过公式推导，获取微分的显示表达式，编写程序计算；（2）数值计算，函数在某个位置的微分可以通过计算函数在相邻两个位置或者多个位置的值获得；（3）符号微分，通过计算机系统推导微分的表达式，比如 Mathematical; (4) 自动微分（Automatic Differention）, 也叫算法微分，利用链式法则，根据从输入变量到输出函数的计算路径，逐步计算输出函数相对于输入变量的微分值。

上述方法各有优缺点，比如公式推导法需要人工参与，对于任何一个新的函数，都需要手动推导。数值计算法虽然自动化程度较高，但容易受数值稳定性的影响，而且为了计算某个变量的微分，至少要计算两次原函数的值，随变量个数的扩展性较差。符号微分相比公式推导，可以自动推导微分公式，不需要人工参与，但由于符号微分的难度很大，该方法能处理的问题范围受限。另外，公式推导法和符号微分法都只能处理可以用表达式显示表示的问题，而实际应用中，很多函数的计算是通过程序表示的，里面包含涉及复杂的控制逻辑，难以用表达式描述。

不论一个函数多复杂, 最终都可以分解为一些基本函数的复合函数。因此,只要知道这些基本函数的梯度计算方法, 就可以通过链式法则, 逐步计算得到目标函数相对输入变量的梯度. 这也是自动微分方法的基本思想. 不同与公式推导法和符号微分法求解导数的显示表达式，自动微分法的目的是计算函数在给定位置的导数值，最明显的优势是可以直接应用在程序上，而不仅仅是显示表达式上。被应用的程序可以包括分支、循环甚至递归等控制逻辑。

基于上述思想, 自动微分有不同的实现方法. 根据应用链式法则的顺序, 可以分为正序模式和逆序模式. 根据记录计算路径的方法, 可以分为源代码翻译和操作符重载两种, 下面分别介绍.

正序模式

所谓正序模式，是指计算函数值和计算梯度同步进行，在记录中间计算结果的同时，也记录该结果相对于函数变量的梯度。例如，我们希望求 f(x) = x^2 + log(x) 在某个位置相对于 x 的梯度值。这个表达式的计算可以分成三步：

t0 = x^2

t1 = log(x)

t2 = t0 + t1

最终 f 相对于 x 的梯度值就等于 t2 相对于 x 的梯度值，而 t2 相对于 x 的梯度值依赖于 t0 和 t1 相对于 x 的梯度值。所以为了计算 t2 相对于 x 的梯度值，我们在计算 t0 和 t1 的同时也计算他们相对于 x 的梯度值。如果分解成步骤则是：

STEP1： t0 = x^2, dt0/dx = 2 * x

STEP2: t1 = log(x), dt1/dx = 1/x

STEP3: t2 = t0 + t1, dt2/dx = dt0/dx + dt1/dx

根据上面的流程，在计算过程中，除了保存当前结果外也需要保存当前结果相对于输入变量的梯度值。

输入变量个数为 N, 输出变量为 M 时空间复杂度: O(NM)

反序模式

反序模式也叫逆序模式。顾明思议，是从结果向后推得到最终的梯度值。它的理论依据仍然是链式法则：df(x)/dx =sum(df / dt_i * dt_i/dx), 其中 t_i 为和 x 相关的中间结果。根据这一法则，只要从后往前计算输出相对于每个中间结果的梯度值，最终就可以得到输出相对于输入变量的梯度值。仍以正序模式的例子说明，在反序模式下，其计算流程是这样的，先进行前向计算

t0 = x^2,

t1 = log(x),

t2 = t0 + t1

f = t2

然后反向计算梯度：

df(x)/df(x) = 1

df(x)/dt2 = 1

df(x)/dt0 = df(x)/dt2 * dt2/dt0 = 1

df(x)/dt1 = df(x)/dt2 * dt2/dt1 = 1

d(fx)/dx = df(x)/dt0 * dt0/dx + df(x) / dt1 * dt1/dx = 2 * x + 1/x

输入变量个数为 N, 输出变量为 M 时空间复杂度: O(N + M)

对于深度学习而言，输出为 Loss 值，通常只有一维，且输入和中间变量很多，逆序模式有明显的空间复杂度优势， 因此实现时通常选逆序模式。

源代码翻译

源代码翻译的思路是在编译前或者编译阶段，根据对原函数的语法分析和链式法则，直接生成计算梯度的代码，并将两者整合在一起进行优化。

因为是在编译阶段完成的，源代码翻译产生的程序在运行时具有较高的性能。但缺点是针对复杂的控制结构，比如条件跳转、循环和递归等，源代码翻译实现的复杂度比较高。

对于深度学习而言，目前大部分应用的仍然是不带控制的静态图，此时使用源代码翻译实现的复杂度较低，比如 Tensorflow 内部就是通过源代码翻译实现自动微分的。

操作符重载

该方案的思路是对函数和运算符进行重载，在调用的时候将对应的操作和输入变量记录下来，通常大家将记录的结构称作 Tape, 然后在计算梯度时，逆序回放 “Tape”, 根据记录下来的输入变量和操作计算梯度。

操作符重载方案实现起来比较简单，另外，由于是运行时记录，Tape 上记录都是实际发生的操作，是一个线性的序列，不受控制逻辑的影响，因此该方案能够支持原函数中有比较复杂的控制逻辑。缺点是由于在运行时记录，计算梯度的过程需要有一个专门的程序负责执行。性能通常不如源代码翻译好。