# 1 为什么是 a fully differentiable programming language?

在描述能力上，理论上（1）“imperative language” 加上（2）“if” and “goto” statements, 或者 “branch if zero” 就足以描述所有能够被计算机处理的计算逻辑，high-level programming 会设计出很多 semantics 和语言特性来方便用户来使用。

- 导数计算是机器学习任务的核心，并且是一个非常规则化和流程化的计算过程，因此非常适合用计算机程序自动化完成导数的计算。
- 但另一方面，high-level programming language 提供的一些语言特性远远超过了“描述任计算过程” 所需要的。
- 在自动实现 AD 的过程中，通过尽可能的保留 programming language 的语言特性，同时应用足够的程序优化技术，来改善用户体验。
  - imperative programming language 的灵活性和表达能力 + compiled programming language 的性能和优化。
  - 解决 two language contexts 带来的用户体验痛点。

# 2 为什么一定要把 AD 实现在 language/compiler level?

更多是为了优化，在编译器做更多的事情。

例如，对 JPL 这种有 staged programming 思想和（一定的）能力的语言，结合独特的类型系统，通过 value types + generated function：（1）可以在编译期特化出针对具体的矩阵大小的实现；（2）在用户真正写程序的那一刻很多 shape 信息都已经是确定的，可以实现编译期的各种 shape 合法性检查。

通常有两种实现 AD 的方式：

1. 动态的 Tracing / operator overloading
   - 不需要显示地处理 control-flow，运行时会得到一个**线性的** function call traces（概念上的 Tape），通过反向 replay Tape 来实现自动的反向。
   - 实现很简单，并且有足够的灵活性。缺点是失去了很多静态分析和全局优化的机会。
2. 静态的 STC（source code transoformation）
   - 需要显示地处理 control-flow。程序被表示成 basic block 为节点，block 之间控制流跳转为边的 control flow graph。
   - 生成反向计算时需要解决 control-flow reverse 的问题。
   - 对一些动态类型语言需要解决 Type inference 问题。
   - 数学函数 $f(x)$ 是可导的，但 $f(x)$ 的实现不一定能直接生成反向计算，需要静态检查。

机器学习任务的输入通常是高维向量，输出（loss function）是一个 scalar。反向 AD 的复杂度与输出维度线性相关，反向 AD 更适合机器学习任务，不论是 Tracing 还是 STC 都需要引入 runtime data structure 来记录前向计算的中间结果，在通常的实现中，会引入 stack 来保存前向计算一个 basic block 中的 context。

# 3 为什么一层 IR 对 language-level 的 AD 是必须的?

编译器的后端最终会负责最终的 binary code optimization，而反向的计算在构建 AST，做完 type inference，parser 之后就可以拿到所需要的全部信息。在进入编译器后端的代码生成之前这一层 “language IR” 上做更方便。但并不是每一个 language 在 parser 之后都有这样一层暴露给开发者的 IR。

在 STC 实现的 AD 中，遇到过两种形式 IR。（1）源自 lamba 演算的 functional reprensentation。这层 IR 非常接近 LISP 的语法；（2）SSA。

1. 把语言的各种表面语法归一化。
2. 构建 CFG，方便进行各种 data flow analysis。
3. 控制 side effects （从这一点上看，Python 非常不适合 SCT：变量类型运行时决定并且可变；imperative language 大量利用了 side effects）
   - side effects 会产生复杂的读写依赖，让 data dependency analysis 和 code optimization 复杂化。
   - reverse mode AD 的计算逻辑对保留前向计算结果有一定的要求，在反向计算完之前需要被保留，不可以被修改。需要控制用户能够使用的语言特性，保证反向计算可以被静态地生成出来。
   - 控制 side effects 的一层 IR 对生成 high-order gradient function 很重要。使得可以对生成出来的反向，重复的应用 SCT 去生成高阶导数的计算。