

# 极客大学机器学习训练营

## Python 性能调优指南

王然

众微科技 AI Lab 负责人

二〇二一年一月九日

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献

- ▶ 虽然很多人说只要 python 写得好，python 性能可以超过 C；但是实际上，python 很容易写出很慢的代码，并且想优化还很难。
- ▶ python 优化重点不在于所谓的 python trick 而在于：
  - ▶ 识别代码瓶颈并找到原因 → Profiler 使用
  - ▶ 首先从宏观上寻找修改策略
  - ▶ 采用更接近于底层的语言（Cython 或者 C++）进行操作
  - ▶ 并行：多进程或者多线程

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献

- ▶ 大部分的代码瓶颈都在某一段上；换句话说，如果你不同代码块之间耗费几乎是一样的，说明你已经优化很好了。
- ▶ 假设有一部分代码是瓶颈所在有两种情况：
  - ▶ 这段代码你改不了（例如：其他框架）→ 换方向换工具或者提 bug
  - ▶ 这段代码你可以改 → 改完就可以收工了

## 一些基本的优化原则：Profiler-based Optimization 极客大学

- ▶ 常见错误，一顿骚操作改了 1% 的损耗部分。
  - ▶ 蚂蚁金服面试题：要求优化一个十秒钟的程序。
  - ▶ ACM 金牌大佬采用各种骚操作，优化了一秒。
  - ▶ 但是这个程序往屏幕上打印了 100 多万个数字。
  - ▶ 如果对 stream 进行优化，就可以优化掉 9 秒。
  - ▶ 结果：直接拒。
- ▶ 记住一句话：Premature Optimization Is the Root of All Evil(Stackify 2020)。

## ► 优化按照重要性排序是这样的：

- 算法本身：例如 A 算法 100 轮迭代就可以收敛，B 算法需要迭代 1000000 轮，B 算法再优化也不如 A 算法效果好。
- 算法复杂度：如果一个算法多做了一些操作（比如说循环两遍但是只需循环一遍），导致性能降低则必然可以进行优化。
- 实现细节：比如说 hash 的设计如果不合适，内存存储顺序不合适，没有利用到 simd 都会导致变慢。
- 并行：没有充分利用多线程



- ▶ 不要上来就并行。并行有很多坑（一会会讲）。
- ▶ 避免恶意抢资源：前同事开 5000 个线程，这样操作系统会分给他更多资源（有其他服务），但是其他服务基本都瘫了；该程序员已被开除。
- ▶ 一些框架可以进行扩容，例如
  - ▶ 模型推断服务可以开一个服务多线程
  - ▶ 也可以开多个服务（例如利用 docker+k8s）；后者可以进行资源动态调配，是优先方案。

- ▶ 寻找 hotspots 往往可以通过 Profiler 实现
  - ▶ Profiler 一般会分为 function profiler 和 line profiler
  - ▶ 对于 python 来说，没有特别好的 profiler。VTune 理论上是最好的，但是他最近又出现识别不了 Python 代码的 bug 了。
  - ▶ line profiler 可能需要你把 python 代码拆散，因为 python 一行可能会实现很多功能。这导致很有这个 profiler 很有侵入性。

- ▶ 寻找原因 → 极其难以做到。原因：
  - ▶ 只能看到 CPU/GPU 底层的表现，不能看到造成这种表现的原因。
  - ▶ 对于 Python 这种高级语言，控制底层常常是很困难的。
  - ▶ 很多优化必须要结合算法本身。
- ▶ 一般规则：
  - ▶ 不论是 CPU 还是 GPU，内存读取往往是瓶颈的核心。所以如果想不到原因，优先找内存操作。
  - ▶ SIMD 是提速的一个非常好的办法。

- ▶ 内存层次：内存  $\rightarrow$  Cache  $\rightarrow$  Register。
- ▶ 理解：
  - ▶ 内存：仓库；汽车
  - ▶ Cache：货架；飞机
  - ▶ Register：工作台；火箭
- ▶ 从上倒下存储量越来越小，但是速度越来越快。
- ▶ CPU 只能在 Register 当中工作。
- ▶ 我们无法控制 Cache
- ▶ 但是 Cache 往往是出事的源泉

- ▶ 不好读进来：
  - ▶ 一般来说读内存地址都是成批读临近数据进 Cache 的；
  - ▶ 但如果不在一块，就非常麻烦
- ▶ 得反复读：
  - ▶ 一般都是读一块数据进来，但如果读错了。
  - ▶ 比如说 branch prediction，如果 if 条件变了，那么就得重新清空 cache。
  - ▶ 在比如说不同线程都在改一段内存，由于 cache 必须跟内存一直，所以得一直读。
  - ▶ Cache miss 是一个很大的课题 (hazelcast 2020)
- ▶ 除去这个以外，很多 CPU 都会出各种奇怪的问题。感兴趣的强烈推荐阅读 Vtune-Cookbook (2020)

- ▶ 现代 CPU 往往可以同时做多个计算（Register 更大），但是操作必须得是一样的（比如说都是乘或加）。
- ▶ 一些编译器会自动进行优化，一些不会。
- ▶ 不同 CPU 能够并行计算的大小不一样。所以不同服务器需要重新编译。
- ▶ 实现这个要求数组排列连续并符合某些性质。
- ▶ 实现方式：
  - ▶ OpenMP 指令（只能用 C/C++）。
  - ▶ 需要 directory。
  - ▶ 或者使用 intrinsics（近乎于汇编程度）。

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
  - Profiler ■ Cython ■ 并行
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献



## 1 核心脉络

## 2 优化方法论

## 3 实践部分

■ Profiler ■ Cython ■ 并行

## 4 实践环节

## 5 要点回顾及预习

## 6 参考文献



- ▶ Function Profiler 可以采用 cProfile
- ▶ Line Profiler 选择很多，都不太好，我们采用 ????
- ▶ Vtune 我们会展示。目前只支持 C++，但理论上 python 是一致的。

- ▶ 假设我们的目标变量是 0, 1; 0 表示负样本, 1 表示正样本。
- ▶ 我们的解释变量是一个离散的变量, 并且有很多类 (比如说职业)。
- ▶ 我们如何拟合模型呢? 我们没法表示  $1 \times$  程序员. 所以我们需要把这些改为一种编码。
- ▶ 一种编码称之为 target encoding。这种方法的实现逻辑如下。

- ▶ 假设我们这里有两类：程序员和保安，目标是预测是否收入会超过 10k。
- ▶ 我们发现程序员当中 90% 超过了 10k，保安只有 10% 超过了 10k。
- ▶ 那么我们就可以用 0.9 替换程序员，用 0.1 替换保安。
- ▶ 但这里有一个问题。

- ▶ 考虑极端例子：每一个样本都是一个新的类。
- ▶ 按照上面的做法：则我们等于知道了  $y$  值。这个模型将会毫无意义。
- ▶ 解决方法：在计算比例的时候，将该值对应的  $y$  去掉再求平均。
- ▶ 这类问题称之为 target leakage。本质是  $x$  直接把对应  $y$  的信息包含了进来，而且预测的时候，这部分信息是没有的。这个问题很复杂，可以参见 Prokhorenkova et al. (2018)。
- ▶ 代码实现见 `target_encoding_v1.py`

- ▶ 需要在终端中运行命令。
- ▶ 命令为 `python -m cProfile myscript.py`
- ▶ 输出为：
  - ▶ `tottime`→ 整体运行时间（不包括调用其他函数）
  - ▶ `cumtime`→ 累计运行时间（包含调用其他函数）

- ▶ 缺乏 call-tree (可以通过其他方法补充)
- ▶ 缺乏每行代码运行时间 (存在其他 line profiler)
- ▶ 缺乏其他 profiling 信息, 尤其是底层信息
- ▶ 难以处理多语言情况
- ▶ 没有图形界面
- ▶ 很多时候时间不准确

- ▶ 我们选取的 profiler 是 line\_profiler。
- ▶ 安装命令 `pip install line_profiler`
- ▶ 需要在需要 profile 的函数加上装饰器 `@profile`
- ▶ 运行 `kernprof -l -v myscript.py`

- ▶ 付费软件，为 Intel Parallel Studio 一部分。
- ▶ 但是，有个网站是[好东西](#)。
- ▶ 运行时先要进入对应 vtune 文件夹运行 `source ./vtune-vars.sh`。
- ▶ 接下来请看展示。
- ▶ 目前有个 bug（展示不出来 python 源代码），正在官网痛骂。



## 1 核心脉络

## 2 优化方法论

## 3 实践部分

■ Profiler ■ Cython ■ 并行

## 4 实践环节

## 5 要点回顾及预习

## 6 参考文献

- ▶ 官网。
- ▶ Python 的 superset: 全部 (?) python 代码都可以运行, 但是并不是其长项。
- ▶ 更好的办法是把他想成 C
- ▶ 最大的区别: 静态类
- ▶ 一个额外的作用: 代码保护

见 hello 文件夹及注释。注意

- ▶ 运行命令 `python setup.py install`。
- ▶ 注意：Cython 的建议是每个文件是一个 module。所以建议 `install` 来配置（docker）。
- ▶ 同理：Cython 互相引用会出很多问题。
- ▶ 很多时候你可能很希望用 intel 编译器，见[这个例子](#)。但是很多时候会出很多问题。

见 `cython_example.ipynb`

见 `connect_c` 文件夹

- ▶ 所有类型均应该有 type。
- ▶ 注意 numpy 的排列。
- ▶ 尽量使用 C++ 自带数据结构。
- ▶ 通用方法：numpy 使用 view 传递给 C；C 使用 openmp 或者 Eigen。使用 Map 可以使用类似的 matlab 的语法。但是不支持 OpenMP。
- ▶ 所有内存分配和传递都应该在 python 中完成。临时变量用 C++ 类进行构建。

## 1 核心脉络

## 2 优化方法论

## 3 实践部分

■ Profiler ■ Cython ■ 并行

## 4 实践环节

## 5 要点回顾及预习

## 6 参考文献

- ▶ 由于 Python 中 GIL(Python 2020) 的存在, 使得本质上 python 只能用一个进程进行。
- ▶ 这使得并行 python 十分困难。
- ▶ python 自带多进程库滥到极点。所以建议使用 ray。
- ▶ ray 长处在于可以调用任何 python 函数, 并可以共享数据 (使得不用 copy)。
- ▶ 但是共享的数据只读。
- ▶ ray 的语法见 ray 文件夹。



- ▶ OpenMP是一套非常复杂的并行计算模块。
- ▶ 整体想法，采用注释的方式使非并行的东西变成并行。
- ▶ 最大问题：对 C++ 支持极差，所以不适合 Eigen（主流矩阵计算库）。如果要用 C++，请用oneTBB
- ▶ 建议的解决方法：使用 Intel 编译器及oneTBB的Dependency Graph+ Eigen::Map。Cython 只做数据预处理。注意：这样做编译过程将会十分痛苦。
- ▶ 如果要在 cython 中运用 openmp，只可以用 prange，并且必须满足各种需求。见具体文件。

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献

- ▶ 我们将从算法角度考虑如何优化。
- ▶ 作业中：学员将会将这段代码改为 Cython（最好加入并行），并比较速度。

我们一起来动手!!!

- ▶ 我们的核心问题在于每一次计算 groupby 都需要重新遍历，groupby 计算的是平均值。那么有没有办法不这样做呢？
- ▶ 办法：两次循环。第一次计算 sum (map-reduce) 模式，第二次我们在 sum 中减去目前观测的，就得到了具体值。
- ▶ 下面时间给大家来实现。

- ▶ 普通：请将该代码改为 cython 代码并比较速度区别（如可以实现并行可加分）。
- ▶ 附加题：完全符合真实工作需求，得分较高者会额外优先进行内推
  - ▶ 查看 [B-spline](#) 的介绍。
  - ▶ 使用 cython 实现对输入多列返回 b-spline basis 的操作。
  - ▶ 注意：禁止使用函数 recursive call。
  - ▶ 注意：必须要处理异常情况（例如缺失值，inf 等）。

可采用下面的练习。

- ▶ 使用 cython 定义如下函数：输入为三个  $3 \times 3$  的矩阵，假设为  $A, B, C$ 。
- ▶ 要求：计算  $(A + B)$  和  $AC$  矩阵乘法，并将两个结果相加。要求  $A + B$  和  $AC$  的计算必须并行。
- ▶ 必须使用 cmake 进行编译和安装。
- ▶ 建议方案：
  - ▶ 使用 Eigen::Map 将指针转化为矩阵。
  - ▶ 使用 tbb 的 Dependency Graph 实现复杂的操作。
  - ▶ 编译请根据[这个例子](#)进行修改。注意增加 numpy 的编译。
  - ▶ 需要手动先用 Cython 生成 C++ 文件。剩下只是安装问题。

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献



- ▶ Profiling 的基本原则。建议阅读参考材料。
- ▶ Cython 的各种语法及使用事项。

- ▶ 请预习 pandas 的基本操作。建议使用教材为McKinney (2012)。
- ▶ 该书也为非常常用的速查手册。建议可以常备。

- 1 核心脉络
- 2 优化方法论
- 3 实践部分
- 4 实践环节
- 5 要点回顾及预习
- 6 参考文献

-  hazelcast (2020). *What's a Cache Miss? Policies That Reduce Cache Misses*. URL: <https://hazelcast.com/glossary/cache-miss/>.
-  McKinney, Wes (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc."
-  Prokhorenkova, Liudmila et al. (2018). "CatBoost: unbiased boosting with categorical features". In: *Advances in neural information processing systems*, pp. 6638–6648.
-  Python, Real (2020). *What Is the Python Global Interpreter Lock (GIL)?* URL: <https://realpython.com/python-gil/>.
-  Stackify (2020). *Why Premature Optimization Is the Root of All Evil*. URL: <https://stackify.com/premature-optimization-evil/>.



Vtune-Cookbook (2020). *Intel® VTune™ Profiler Performance Analysis Cookbook*. URL:  
<https://software.intel.com/content/www/us/en/develop/documentation/vtune-cookbook/top.html>.

Thanks!