第 14 章

Python の高速化について

76 回生 まふまふ

14.1　はじめに

76 回生のまふまふです (￣▽￣)ノ. まだ知らないことも多いのでよろしくお願いします.

僕は主に Python を使っているので,Python について書こうと思ったのでこの内容にしました.

Python を使ったことがない人やプログラミングを知らない人でも, 面白そうだなぁと思ってくれたら嬉しいです.

特にプログラミングをしたことがない人は Python から始めることをお勧めします (￣▽￣)ノ

いきなり C 言語[*1]や Java(⇦プログラミング言語のひとつ) からなどから始めようとすると, 難しくてあきらめてしまうかもしれません.

Python なら初心者でもわかりやすいような簡単な文法なので, 簡単に始められます.

もちろん他の言語のような高度なこともできるため, 機械学習などにも生かせるため最近人気です!

今回はこの言語の唯一の欠点の『速度が遅い』ことを克服するため, 高速化について書こうと思います.

Python がどういうものか知らない方も目を通していただけたら嬉しいです.

そんなに長くないので... 汗

14.2　なぜ Python は遅いの??

Python が遅い理由はいくつかあります.

インタープリタ型言語であること

Python はインタープリタ型言語といって実行するときに一行ずつ機械語に翻訳されていきます.

なので C 言語などのコンパイル型言語[*2]とくらべてかなり速度が低下します (; ´･ω･)

変数宣言がない

Python では変数宣言 (型を明確に表記して, 変数を使用すること) が不要で, いきなりソースコードの中で新しい変数を使用することができます.

※変数とは値 (整数, 文字列など...) を保存する箱みたいなもの (?) です. 数学の変数とは意味が違います.

なので変数を使った操作で毎回型チェックが行われます. ⇦遅くなる原因

考察

これ以外にもたくさんの理由がありますが, 主な理由はここに挙げた二つです.

[*1] C 言語とは古くからあるプログラミング言語です. それの拡張版に当たるのが C++ でこれから Java,C#などの言語が作られています.Python は C 言語で作られています.

[*2] コンパイルとは書かれたコードを機械語に翻訳することです. インタプリタ型言語では一行ずつ翻訳してくれるのに対して, コンパイル型言語ではコンパイルという過程によりいっきに翻訳されます. これが速度にも関わってくるのです.

どうにかして速くできないですかねぇ...)

14.3　高速化したいのはどうして??

繰り返しの処理が遅い!!

理由の一つにループ処理 (繰り返しの処理) が遅いことがあります. 型指定を行わないせいで, ループ時にも型のチェックが必要となってすごく遅くなるんですね.

14.4　では, どうやって高速化するのか

どうすれば遅いものが速くなるの??と, 思うでしょう (多分.
しかし Python のままで高速化する方法があるんです.
※速度計測環境
[pc]:CPython&C++:Windows 10,Cython&PyPy:Ubuntu,[Python]:Python3.7
※比較に使う Python のコード

cpython.py

```
from time import time
def mf():# 関数の定義
    a=0;
    for i in range(100000):
      a+=1;
st=time();
mf();
en=time()
print("%s[sec]"%(en-st));
if __name__=="__main__":# 呼び出されたら実行する
    mf()
```

速度は **0.0047**[sec] でした.
※ CPython とは C で作られた Python のことで, これが通常の Python です.
これ以外には RPython などがあります.
(C で作られているのに遅いのは納得いかないな

NumPy を使う!!

NumPy というライブラリ (これを読み込むことでその機能を使用できる) があります.
このライブラリは C 言語でつくられています.(速い理由)
これを使用すると高度な計算なども高速に行うことができます!!

導入方法 (NumPy)

pip コマンド (Python をダウンロードすると, コマンドプロンプトから使用できます) を使って `pip install numpy` で使えるようになります.(簡単ですね!!
そして import 文 (必要なライブラリを読み込むコード) で `import numpy` とすると使えます.

計測 (NumPy)

計算に限られるため, ほかの方法との比較はできません. 次のに期待しましょう)

Numba を使う！

次は Numba ライブラリの JIT コンパイル[*3]を使う方法です.
この方法では主に NumPy を扱ったプログラムが大幅に高速化します.JIT 技術を使ってコンパイルされます. ⇦速い原因

[*3] JIT とはコンパイル技術の一種で Java(プログラミング言語) などで採用されています. コンパイル後の速度は速くなりますが実行開始までが遅いという問題があります.

NumPy を使わないプログラムではあまり効果がないことがありますが, ループ処理のあるものは速くなることがあります.

導入方法 (Numba)

pip コマンドを使って `pip install numba` で使えるようになります. デコレータ[*4]を使って

jit_.py

```
from numba import jit# import numbaで関数前で@numba.jitでもok
@jit
def mf():# mf には定義したい関数名を
    # ....
```

とすると自動でその関数がコンパイルされます.

注意！ (Numba)

Numba の JIT を使う上での注意は一回目は速くならないことです.
二回目以降の呼び出し時にはすでに一回目のときにコンパイルされているため, 速く処理を行えますけどね.

計測 (Numba)

計測しまーす!

numba_.py

```
from time import time
from numba import jit
@jit
def mf():
    a=0;
    for i in range(100000):
    a+=1;
st=time();
mf();
en=time()
print("%s[sec]"%(en-st));
if __name__=="__main__":
    mf()
```

速度は *Python*:**0.0047**[sec],*Numba*:0.0674[sec]
ということで残念ながら速くなりませんでした. やはり NumPy 用なのかなぁ...
※ファイル名の最後に"_"をつけているのは自分自身を Numba というライブラリとして読み込まないようにするためです.Python では優先度を分ける書き方がないので^^;

Cython を使う!!

今までは計算だけ!とか NumPy をつかったものだけ!などでしたが, 今回はどのようなプログラムにも適用できて速いです!!!
Cython は一種のプログラミング言語です. しかし Python との高い互換性があり, 多くのコードはそのままでも使えます.
Cython がなぜ速くなるのかというと, コンパイルするからです. また型指定 (変数宣言) もできます.
望み通りの素晴らしい言語ですねぇ (笑)
Python のライブラリも幅広く対応しており便利です.

導入方法 (Cython)

pip コマンドを使って `pip install cython` で使えるようになります. ※管理者権限が必要です
どのようにすると使えるのかというと, まず記述したコードを `.pyx` という拡張子で保存します.
その後 `cython mf.pyx` で `.c` ファイルに変換します.(mf.pyx には任意のファイルを指定)
さらに, これによって生成された `.c` ファイルを C のコンパイラ[*5]を使用して `.pyd` (Mac,Linux では `.so`) にコンパイルします.

[*4] デコレータとは関数の定義の前で `@mf` (mf には任意の関数名) とすることで, その関数に直接書き換えずに機能を編集できる機能らしいです. 簡単に言うとある関数を実行してからある関数を実行するみたいな感じです (余計にわからなくなった!?. たとえば関数 func と関数 mf があるとします. 関数 mf の定義の前で `@func` と書いたとすると, ずばり `func(mf())` と同じことになります.) 説明難しい...

[*5] C コンパイラとは C 言語や C++ で書かれた `.c` ファイルをコンパイルして実行形式の `.exe` ファイルにします. この処理は書かれたコードを一気に機械語に翻訳します. 主に GCC などのコンパイラや Visual Studio などの総合開発環境を使うことが多いです.

これにより,Python から import するだけで簡単に使える**高速なライブラリ**ができます.
このファイルは共有ライブラリといい,C,Python などから使用できるライブラリです.
(Jupyter Notebook[*6]を使って対話的に Cython を使用することもできます. この場合, 自分でコンパイルする必要はありません)
Cython 公式サイト
和訳された Cython ドキュメント (少し古いです)
で, 終わろうと思ったのですが手元の環境では上記の方法ではエラーとなったので, もう一つの方法を紹介します!

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [Extension("mf", ["mf.py"])]
setup(
  name = 'mf',
  cmdclass = {'build_ext': build_ext},
  ext_modules = ext_modules
)
```

このような内容の setup.py を用意します.
そしてコマンドプロンプトからそのフォルダに移動し (cd コマンドを使用) て, `python setup.py build_ext --inplace` と入力します.
すると, `.py` ファイルから `.pyd` または `.so` に直接変換できます!!
※ Visual Studio がインストールされている必要があります.
(;;) なんで両方できないんだぁ-(基本はできますのでご心配なく...)

計測 (Cython)

Cython でこのコードを実行してみます.

cython.pyx

```
from time import time
def mf():
    a=0;
    for i in range(100000):
        a+=1;
st=time();
mf();
en=time()
print("%s[sec]"%(en-st));
if __name__=="__main__":
    mf()
```

※残念なことに `cython mf.pyx` を実行したときに `language_level` でエラーが発生しました. 基本は `cython language_level=3` できるはずなのですができませんでした. 前の PC ではできたのに...

PyPy を使う！

PyPy は CPython(CPython とは C で作られた Python です. つまり一般的な Python のことです) より速いというのがうりです.
Numba と同じように JIT 技術を使います. ただ, 一つ問題があり **C でつくられたライブラリが使えない**ことです (RPython 特有なので仕方ないです). しかし,NumPy などの主要ライブラリは PyPy 版があり, ある程度フォローされています!!
PyPy2,PyPy3 は AtCoder[*7]の言語にもあります. 言語なのかは怪しいですが) ※ Cython はありません
単なる Python のコードでも PyPy として提出することで実際に Python では実行時間超過だったコードが正解となることもあるようです.

[*6] Jupyter Notebook とは Python のコードを対話的に実行できるものでこれを使って Cython のコードを対話的に実行することができます. 使用するには `pip install jupyter` として, 起動するには `jupyter notebook` または `ipython notebook` とします. ※ web ベースで使用できます
[*7] AtCoder とは競技プログラミングサイトの一つで, プログラミングの速度や技術を競います.

導入方法 (PyPy)

PyPy ではここが重要です.apt コマンドが使える環境では `apt install pypy` でインストールできるのですが,Windows 等の場合は公式サイトからファイルをダウンロードする必要があります.

公式サイト

Python3 の場合は PyPy3 なので PyPy3 をダウンロードしてください.

これをコマンドラインから使用するには自分でパスを通す必要があります.

使用するには `pypy mf.py` とします.mf の部分には任意のファイル名を指定してくださいね！

これで使用できます.

14.5 計測 (PyPy)

pypy.py

```
from time import time
def mf():
    a=0;
    for i in range(100000):
        a+=1;
st=time();
mf();
en=time()
print("%s[sec]"%(en-st));
if __name__=="__main__":
    mf()
```

速度は *Python*:0.0047[sec],*PyPy*:**0.0010**[sec]

速くなりましたね!

14.6 pip コマンドを使っても使えなかった場合の対処法

もし `'pip' は, 内部コマンドまたは外部コマンド, 操作可能なプログラムまたはバッチ ファイルとして認識されていません.` とでた場合は pip がインストールできていないので `apt install python-pip` でダウンロードできます. ※ Windows でのメッセージ.Mac の場合は `-bash: pip: command not found`

※ PyPy や Python2 が入っている場合になることがあります.

あとがき

速度が変わらなかったものもありましたが, 状況に合わせて使用すれば速くなりそうですね. 実際に使ってみるのもいいですね!

プログラミングを知らない人も,Python を知らない人も, 少しでも Python に興味をもってもらえれば幸いです.

最後まで読んでくれた方、ありがとうございました!!