Rýchla Fourierova transformácia

Daniel Kyselica

10. apríla 2019

Problém

Pre efektívne používanie Fourierovej transformácie, je dôležité mať rýchly algoritmus na jej výpočet. Diskrétnu Fourierovu transformáciu popisuje vzorec pre vstup veľkosti N a $0 \leq k < N$ [**?**]

$$F(k) = \sum_{n=0}^{N-1} f(n) e^{-2i\pi \frac{n}{N} k} \tag{1}$$

Každá hodnota $F(k)$ je vypočitaná z N hodnôt vstupu. Priamočiarou implementáciou získame vzorec s časovou zložitosťou $O(N^2)$.

Cooley–Tukey algoritmus

Tento algoritmus rýchlej Fourierovej transformácie alebo fast Fourier transform, **FFT**, je najpoužívanejším algoritmom na výpočet Fourierovej transformácie. Funguje na princípe *rozdeľuj a panuj*. Pôvodnú rovnicu upravíme tak, aby sme jej výpočet rozdelili na dva alebo viac menších častí.

$$\begin{aligned} F(k) &= \sum_{n=0}^{N-1} f(n) e^{-2i\pi \frac{n}{N} k} && (2) \\ &= \sum_{n=0}^{N-1} f(n) \omega_N^{nk} && (3) \end{aligned}$$

Sumu rozdelíme na dve. Prvá suma pre párne hosnoty n, $n = 2x$, a druhá suma pre nepárne n, teda pre $n = 2x + 1$.

$$= \sum_{x=0}^{N/2-1} f(2x)e^{-2i\pi\frac{2x}{N}k} + \sum_{x=0}^{N/2-1} f(2x+1)e^{-2i\pi\frac{2x+1}{N}k} \quad (4)$$

$$= \sum_{x=0}^{N/2-1} f(2x)e^{-2i\pi\frac{x}{N/2}k} + \sum_{x=0}^{N/2-1} f(2x+1)e^{-2i\pi\frac{1}{N}k}e^{-2i\pi\frac{x}{N/2}k} \quad (5)$$

V indexe exponenta sme presunuly dvojku z čitateľa do menovateľa. Ďalej využijeme nasledujúcu vlastnosť.

$$e^{-2i\pi\frac{x}{N/2}k} = \omega_{N/2}^{x}k \quad (6)$$

Rovnica sa dá následne upraviť do tvaru

$$= \sum_{x=0}^{N/2-1} f(2x)\omega_{N/2}^{xk} + \sum_{x=0}^{N/2-1} f(2x+1)\omega_{N/2}^{k}\omega_{N/2}^{xk} \quad (7)$$

$$= \sum_{x=0}^{N/2-1} f(2x)\omega_{N/2}^{xk} + \omega_{N/2}^{k}\sum_{x=0}^{N/2-1} f(2x+1)\omega_{N/2}^{xk} \quad (8)$$

Po úprave nám vznikly dve sumy, pričom jasne vidieť, že ide o Fourierove transformácie pre párne a nepárne prvky vstupu. Tieto transformácie rovnakým spôsobom vypočítame z ich menších transformácií. Menšie transformácie pracujú s $0 \leq k < N/2$ avšak vo vzorci počítame aj pre hodnoty $N/2 \leq k < N$. Vďaka cyklickej vlastnosti ω_Y to nie je problém. Príklad pre $N/2 = 4$

$$\omega_4^{6x} = \omega_4^{2x} \quad (9)$$

Pre $N/2 \leq k < N$ sa použije už vypočítaná hodnota pre $0 \leq k < N/2$. Čas trvania algoritmu pre vstup veľkosti N je

$$T(N) = 2T(N/2) + O(N) \quad (10)$$

Tento vzťah rozpíšeme ďalej podľa rekurzívnej formuly a vzťahu $T(1) = O(1)$

$$\begin{aligned} T(N) &= 2T(N/2) + O(N) && (11)\\ &= 4T(N/4) + O(N) + 2O(N/2) && (12)\\ &= 8T(N/8) + O(N) + 2O(N/2) + 4O(N/4) && (13)\\ &= .... && (14)\\ &= 2^{\log N}T(1) + O(N)\log N && (15)\\ &= NO(1) + O(N)\log N && (16)\\ &= O(N) + O(N)\log N && (17) \end{aligned}$$

$$O(N) + O(N) \log N \in O(NlogN) \tag{18}$$

Časová zložitosť algoritmu je $O(NlogN)$.Tento algoritmus však funguje len pre vstupy veľkosti $N = 2^a, a \in \mathbb{N}$. Miernymi úpravami sa dá docieliť rozdelenie v rôznych pomeroch ak $N = OP, O, P \in \mathbb{N}$. Pre ilustráciu demonštráciu myšlienky postačuje riešenie pre vstupy veľkosti $N = 2^a, a \in \mathbb{N}$.

Implementácia v Pythone

Tu je uvedená ilustračná implementácia v Pythone. FFT je implementovaná vo väčšine matematických knižniciach ako napríklad Numpy, SciPy alebo knižniciach na spracovanie obrazu OpenCV. Tieto implementácie sú efektívnejšie ako dole uvedená kôli pomalej interpretácií python kódu oproti kompilovaným jazykom napríklad C, v ktorom sú napísané ostatné implementácie.

```
def omega(base):
    power = np.linspace(0,base-1, base)
    return np.exp(-2*np.pi*1j*power/base)

def fft(x):

    N = len(x)

    if N == 1:
        return x

    even = fft(x[::2])
    even = np.append(even,even) # cirkulacia

    odd = fft(x[1::2])
    odd = np.append(odd,odd)

    return even + omega(N)* odd
```

Literatúra

[1] Introduction to Algorithms, 2rd Edition (The MIT Press) 2001 - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein