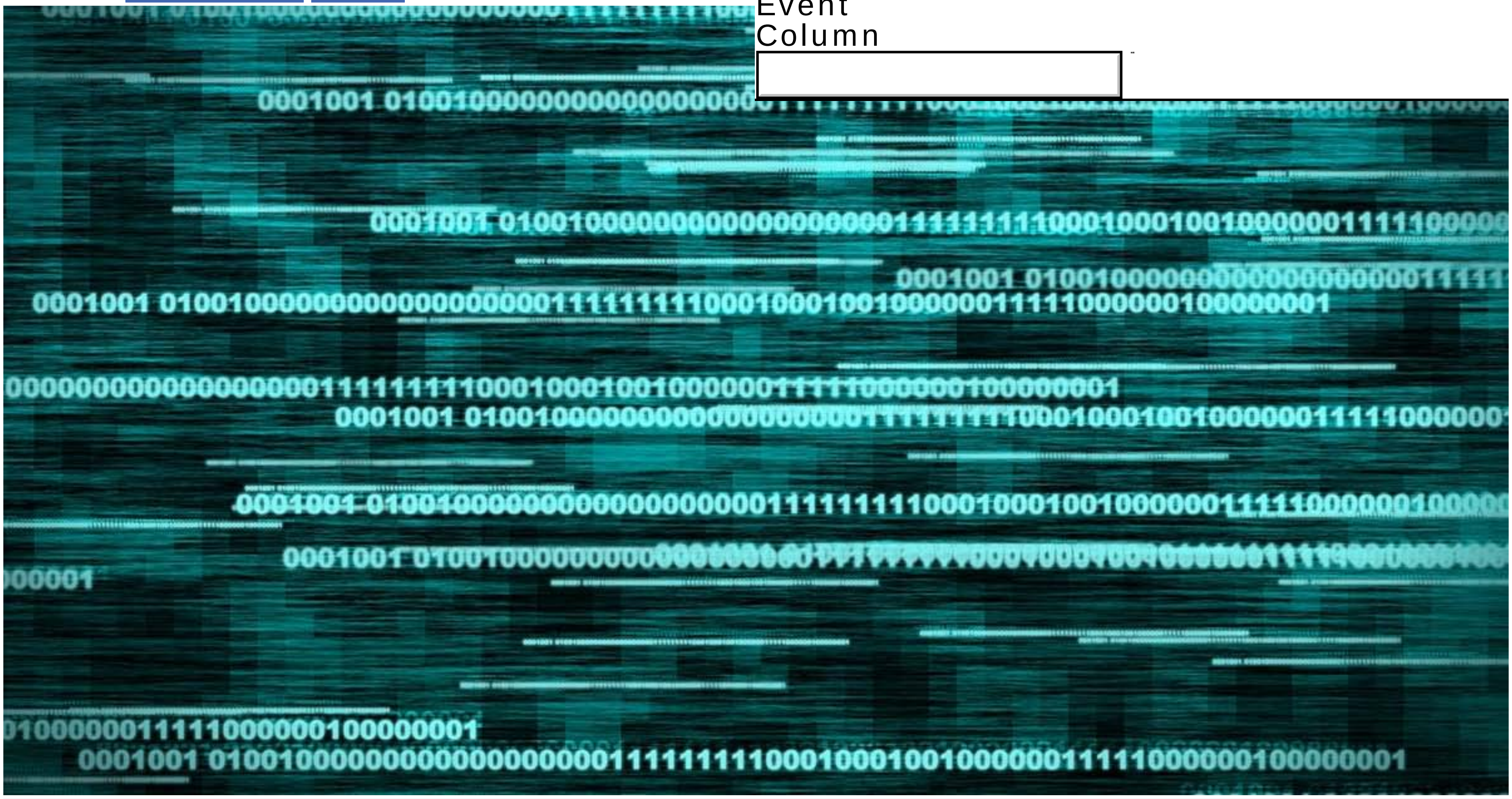
可分な畳み込みカーネルと

tmtk

NHN TECHORUS
Tech Blog
AWS
Data Science
Tech
Event
Column



Topics

こんにちは。データサイエンスチームのtmtkです。
この記事では、可分なカーネルによる畳み込みと計算量の説明をします。

畳み込みとは

はじめに畳み込みを復習します。
機械学習において、畳み込みとは以下のような処理です。いま、入力が2次元の場合を考えることにします。
畳み込みカーネルを$K \in \mathbb{R}^{n_1 \times n_2}$とするとき、入力$I \in \mathbb{R}^{N_1 \times N_2}$に対するカーネル$K$による**畳み込み**
$I * K \in \mathbb{R}^{N_1 \times N_2}$は、$1 \leq x \leq N_1, 1 \leq y \leq N_2$に対して

$$(I * K)(x, y) = \sum_{1 \leq k \leq n_1} \sum_{1 \leq l \leq n_2} I(x + k - 1, y + l - 1)K(k, l)$$

で定義されます。
畳み込みは、古典的な画像処理や最近流行りのディープラーニングでも用いられています。画像認識の本やディープラーニングの本に詳しいことが書かれています。

Pythonで書く畳み込み処理

Pythonで実際に畳み込み処理を書いてみます。コンピュータで処理するため、入力IとカーネルKは0-originなindexを持つとします。つまり、行列として表示すると

$$I = \begin{pmatrix} I(0,0) & \cdots & I(0, N_2 - 1) \\ \vdots & \ddots & \vdots \\ I(N_1 - 1, 0) & \cdots & I(N_1 - 1, N_2 - 1) \end{pmatrix}, K = \begin{pmatrix} K(0,0) & \cdots & K(0, n_2 - 1) \\ \vdots & \ddots & \vdots \\ K(n_1 - 1, 0) & \cdots & K(n_1 - 1, n_2 - 1) \end{pmatrix}$$

となります。
いま、入力のサイズを$N = N_1 = N_2 = 10$、カーネルのサイズを$n = n_1 = n_2 = 3$、入力データを$I(x, y) = 1 (0 \leq x, y \leq N - 1)$、カーネルを$K(x, y) = 1 (0 \leq x, y \leq n - 1)$とします。

```
N = 10
n = 3
image = [[1] * N for _ in range(N)]
kernel = [[1] * n for _ in range(n)]
```

すると、畳み込み$I * K$を計算するPythonの関数は以下のように書くことができます。

```
def convolution(image, kernel):
    N1 = len(image)
    N2 = len(image[0])
    n1 = len(kernel)
    n2 = len(kernel[0])
    res = [[0] * N2 for _ in range(N1)]
    for i in range(N1):
        for j in range(N2):
            for k in range(min(n1, N1 - i)):
                for l in range(min(n2, N2 - j)):
                    res[i][j] += image[i+k][j+l] * kernel[k][l]
    return res
```

実際に計算してみると、

```
convolution(image, kernel)
```

以下のような畳み込み$I * K$の計算結果を得ます。

```
[[9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [6, 6, 6, 6, 6, 6, 6, 6, 4, 2],
 [3, 3, 3, 3, 3, 3, 3, 3, 2, 1]]
```

可分な畳み込みカーネル

ところで、いま与えた畳み込みカーネル$K(x,y)=1(0\leq x,y\leq N-1)$は可分なカーネルの例になっています。2次元のカーネル$K\in\mathbb{R}^{n_1\times n_2}$が**可分**であるとは、二つのベクトル$K_1=(K_1(1),\ldots,K_1(n_1))^T, K_2=(K_2(1),\ldots,K_2(n_2))^T$によって$K=K_1K_2^T$と書けることをいいます。先ほどのカーネル$K(x,y)=1(0\leq x,y\leq N-1)$は、

$$K=\begin{pmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{pmatrix}=\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}\begin{pmatrix}1\cdots 1\end{pmatrix}=K_1K_2^T$$

（ただし$K_1=K_2=(1,\ldots,1)^T\in\mathbb{R}^n$）と書けるので、実際に可分なカーネルになっています。
可分なカーネルの例としては、他にも平均値をとるカーネル

$$K=\frac{1}{n^2}\begin{pmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{pmatrix}$$

や、ガウス関数から作られるカーネル

$$K(x,y)=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

などがあります。

可分なカーネル$K = K_1 K_2^T$に対しては、以下の等式が成り立ちます。

$$I * (K_1 K_2^T) = (I * K_1) * K_2^T$$

証明は簡単なので省略します。
それでは、この性質を使って先ほどの畳み込みを計算してみましょう。関数`convolution_separable`は可分なカーネル$K = K_1 K_2^T$での畳み込み$I * K = I * (K_1 K_2^T)$を$(I * K_1) * K_2^T$で計算する関数です。

```
kernel1 = [1]*n
kernel2 = [1]*n
def convolution_separable(image, kernel1, kernel2):
    N1 = len(image)
    N2 = len(image[0])
    n1 = len(kernel1)
    n2 = len(kernel2)
    res = [[0] * N2 for _ in range(N1)]
    tmp = [[0] * N2 for _ in range(N1)]
    for i in range(N1):
        for j in range(N2):
            for k in range(min(n1, N1 - i)):
                tmp[i][j] += image[i+k][j] * kernel1[k]
    for i in range(N1):
        for j in range(N2):
            for l in range(min(n2, N2 - j)):
                res[i][j] += tmp[i][j+l] * kernel2[l]
    return res
convolution_separable(image, kernel1, kernel2)
```

```
[[9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [9, 9, 9, 9, 9, 9, 9, 9, 6, 3],
 [6, 6, 6, 6, 6, 6, 6, 6, 4, 2],
 [3, 3, 3, 3, 3, 3, 3, 3, 2, 1]]
```

先ほどの計算方法と同じ結果を得ることができます。

それぞれの方法の計算量

それぞれの計算方法について、処理時間を見積もってみましょう。

最初の方法では、関数`convolution`に四重ループ

```
for i in range(N1):
    for j in range(N2):
        for k in range(min(n1, N1 - i)):
            for l in range(min(n2, N2 - j)):
```

があり、ループが約$N_1 \times N_2 \times n_1 \times n_2$回実行されます。いま、$N = \max(N_1, N_2), n = \max(n_1, n_2)$とおけば、ループの回数は$N^2 n^2$回以下です。そのため、この関数の処理時間は$N^2 n^2$にほぼ比例すると考えることができます。このような状況をさして、この関数は**時間計算量**が$O(N^2 n^2)$であるといいます。
それに対して、可分なカーネルに対する畳み込みでは、関数`convolution_separable`に以下の2つの三重ループがあります。

```
for i in range(N1):
    for j in range(N2):
        for k in range(min(n1, N1 - i)):
```

```
for i in range(N1):
    for j in range(N2):
        for l in range(min(n2, N2 - j)):
```

そのため、この`convolution_separable`の処理時間は$2 \times N^2 n$にほぼ比例すると考えることができ、時間計算量は$O(N^2 n)$です（$O(f(n))$と書くときは、定数倍を無視します）。
したがって、計算量がそれぞれ$O(N^2 n^2), O(N^2 n)$で、前者より後者のほうが小さいため、nが大きいとき関数`convolution`より`convolution_separable`のほうが計算時間が短くなることが予想できます。

実際の計算時間

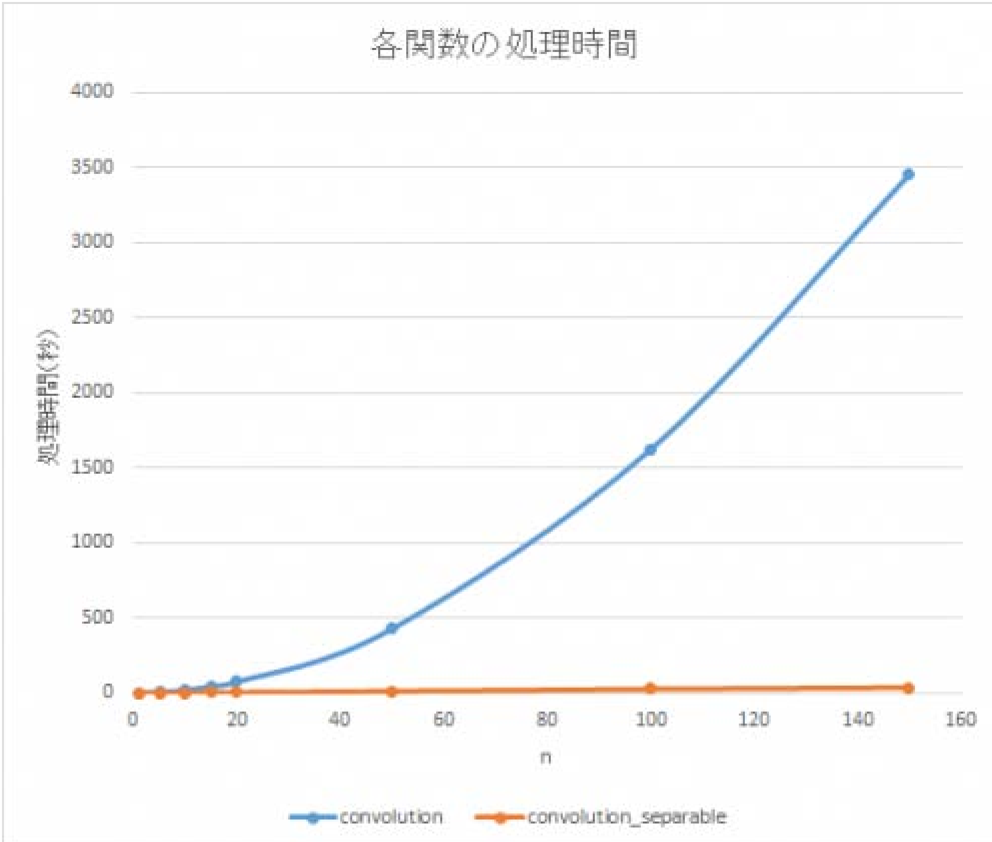
それでは、時間計算量が実際の計算時間に及ぼす影響を観察するため、どれくらいの処理時間がかかるのか計測してみます。
計算環境はCore i7が載っている普通のパソコン上の仮想マシンです。$N_1 = N_2 = N = 1000$として、$n_1 = n_2 = n$を変動させながら、畳み込み処理にかかった時間をIPythonのマジックコマンド`%time`で計測します。

```
N = 1000
image = [[1] * N for _ in range(N)]
```

たとえば、$n = 10$のとき、次のようなコードで処理時間を算出します。

```
n = 10
%time convolution(image, [[1] * n for _ in range(n)])
%time convolution_separable(image, [1]*n, [1]*n)
```

結果は以下のようになります。関数`convolution`の処理時間がn^2に比例していることと、関数`convolution_separable`の処理時間がnに比例していることが観察できます。



ここから推定すると、$n = 500$のとき、`convolution`の処理時間はおよそ10時間程度かかってしまうのにたいして、`convolution_separable`の処理時間はたったの2分程度になります。時間計算量が$O(N^2n^2), O(N^2n)$と異なるため、nが大きくなると処理時間の差も大きくなっていきます。
このように、大きいデータを処理するとき、時間計算量を考慮することは重要です。

まとめ

この記事では、可分なカーネルでの畳み込みの効率的な計算方法と、それによる時間計算量の差について説明しました。可分なカーネルでの畳み込みは、通常の畳み込みよりも高速に計算することができます。時間計算量の差は大きなデータになると実際の処理時間の差に如実に現れてくるので、時間計算量を考慮したアルゴリズムを設計することが大切です。

参考文献

- 原田達也『画像認識』
- Ian Goodfellow他『深層学習』

- Separable convolution » Steve on Image Processing – MATLAB & Simulink
- ランダウの記号 – Wikipedia
- 8.2. コンボリューション行列…
  - 畳み込みを画像処理に適用した例が載っています。

ツイート 0

#Python#深層学習#画像認識

データ分析と機械学習とソフトウェア開発をしています。 アルゴリズムとデータ構造が好きです。

Recommends

こちらもおすすめ

離散フーリエ変換と畳み込

2018.8.1

Data Science

S

Tech

GCPの利用料が安くなる|GCPの請求代行・運用代行・導入移行支援AWS

アの

2020.5.18

GCP2020.

商標について 個人情報保護方針 ISMS認証