Числовые алгоритмы. Матрицы. ML

@pvavilin

19 февраля 2022 г.

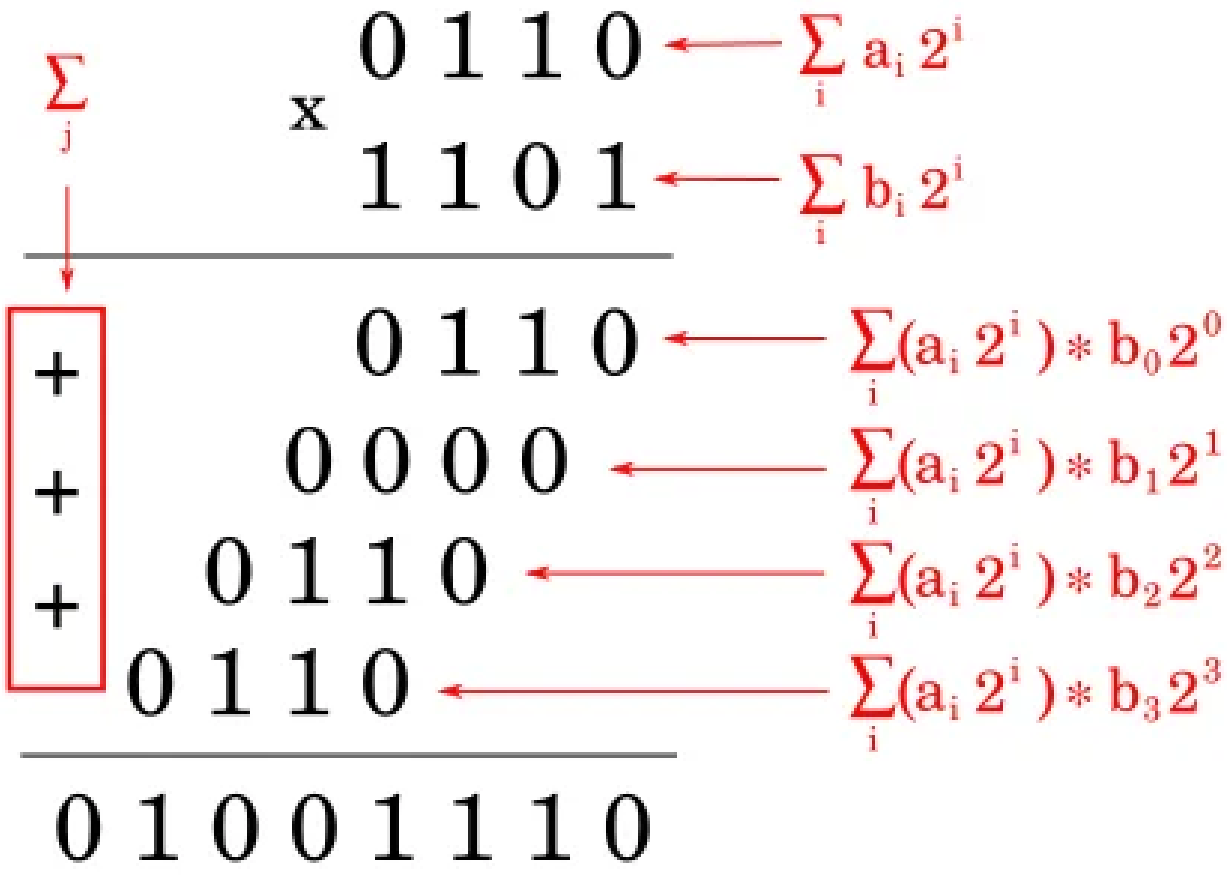
Outline

Умножение двух чисел

$$
\begin{array}{r}
\ 5\,6\,7\,8 \\
\times\ 1\,2\,3\,4 \\
\hline
\ 2\,2\,7\,1\,2 \\
\ 1\,7\,0\,3\,4 \\
1\,1\,3\,5\,6 \\
5\,6\,7\,8 \\
\hline
7\,0\,0\,6\,6\,5\,2
\end{array}
\tag{1}
$$

Можно ли лучше?

Умножение двоичных чисел



Алгоритм Каратцубы

$$
\begin{aligned}
x &= 5678 \\
y &= 1234 \\
\\
a &= 56; \; b = 78 \\
c &= 12; \; d = 34
\end{aligned}
$$

Алгоритм Каратцубы

```
# step1
step1 = a * c
# step2
step2 = b * d
# step3
a_b = a + b
c_d = c + d
step3 = a_b * c_d
# step4:
# step3 - step2 - step1
step4 = step3 - step2 - step1
```

Алгоритм Каратцубы

```
line1 = step1 * 10**4
line2 = step2
line3 = step4 * 10**2
result = (
    line1
    + line2
    + line3
)
print(result)
```

```
7006652
```

Классическое умножение

$$
\begin{array}{r}
\,5\,6\,7\,8 \\
\times\,1\,2\,3\,4 \\
\hline
2\,2\,7\,1\,2 \\
1\,7\,0\,3\,4 \\
1\,1\,3\,5\,6 \\
5\,6\,7\,8 \\
\hline
7\,0\,0\,6\,6\,5\,2
\end{array}
\qquad (2)
$$

Классическое умножение

```
print(classic(1234, 5678))
```

```
7006652
```

```
print(timeit.timeit(
    "classic(1234, 5678) == 7006652",
    globals=globals()
))
# print(timeit.timeit(
#     f"classic({big_x}, {big_y})",
#     globals=globals()
# ))
```

Алгоритм Каратцубы

```
print(timeit.timeit(
    "karatsuba(1234, 5678) == 7006652",
    globals=globals()
))
# print(timeit.timeit(
#     f"karatsuba({big_x}, {big_y})",
#     globals=globals()
# ))
```

Умножение матриц

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \times \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix} \quad (3)$$

Умножение матриц

```
def mxm(A, X):
    n = len(A)      # A: n×m
    m = len(A[0])
    p = len(X[0]) # X: m×p
    B = [[0] * p] * n
    for i in range(n):
        for j in range(p):
            for k in range(m):
                B[i][j] += A[i][k]*X[k][j]
    return B
```

Где ошибка в этом коде?

Умножение матриц

```
def mxm(A, X):
    n = len(A)      # A: n×m
    m = len(A[0])
    p = len(X[0]) # X: m×p
    B = [[0] * p for _ in range(n)]
    for i in range(n):
        for j in range(p):
            for k in range(m):
                B[i][j] += A[i][k]*X[k][j]
    return B
```

Умножение матриц

$O(n^3)$

Можно ли лучше?

Алгоритм Штрассена

$$\begin{bmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

$$\begin{bmatrix} 11 & 21 & 31 & 41 \\ 12 & 22 & 32 & 42 \\ 13 & 23 & 33 & 43 \\ 14 & 24 & 34 & 44 \end{bmatrix} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Алгоритм Штрассена

$$
\begin{aligned}
P_1 &= A(F - H), \\
P_2 &= (A + B)H, \\
P_3 &= (C + D)E, \\
P_4 &= D(G - E), \\
P_5 &= (A + D)(E + H), \\
P_6 &= (B - D)(G + H), \\
P_7 &= (A - C)(E + F)
\end{aligned}
$$

Алгоритм Штрассена

$$\begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix} = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 + P_7 \end{bmatrix}$$

векторизация

- Большинство операций процессора это SISD: Single Instruction Single Data
- Процессор может поддерживать специальные регистры для SIMD: Single Instruction Multiple Data

векторизация

0	1	2	3
a[0]=	not used	not used	not used
b[0]+	not used	not used	not used
c[0]	not used	not used	not used

векторизация

0	1	2	3
a[0]=	a[1]=	a[2]=	a[3]=
b[0]+	b[1]+	b[2]+	b[3]+
c[0]	c[1]	c[2]	c[3]

векторизация

- Без векторизованных операций

```
g++ -o novec vecexample.cpp
echo "Без векторизации"
./novec 1000
```

```
Без векторизации
Time used for norm computation=8.4000E-05
  Norm-2  = 1.5000
```

```
g++ -O3 -mavx2 -o vec vecexample.cpp
echo "Используя векторизацию"
./vec 1000
```

```
Используя векторизацию
Time used for norm computation=7.0000E-05
  Norm-2  = 1.5000
```

NumPy

```
_a = (
    np.arange(n, dtype=float) * 2
    * np.math.pi / n
)
a = s * (np.sin(_a) + np.cos(_a))
b = s * np.sin(2.0 * _a)
c = a + b
norm2 = np.sum(np.power(c, 2))
```

```
./numpy_vec.py 1000
```

```
Time used for norm computation = 0.00017
Norm-2 = 1.50000
```

Логистическая регрессия

$$z = w_0 x + w_1 x + \cdots + w_n x + b$$
$$a = \frac{1}{1+e^{-z}}$$

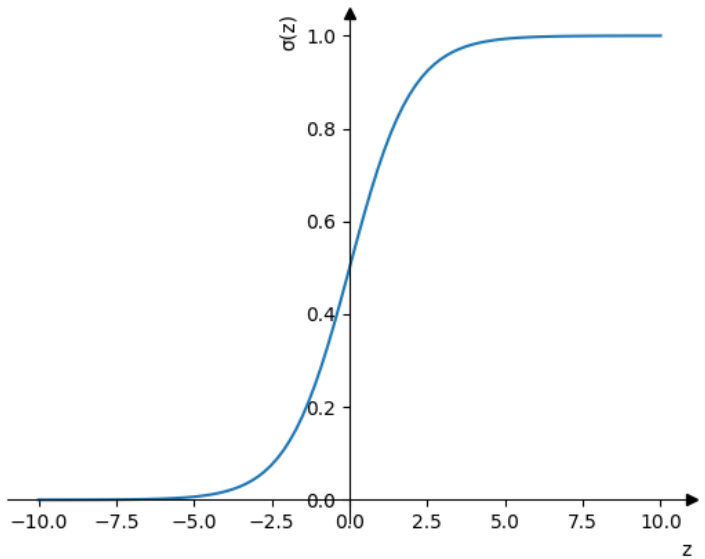


Рис.: sigmoid

Обучение

Чтобы минимизировать ошибку в ответах будем искать минимум функции, вычисляя градиент (производную) для каждой переменной.

$$w = w - \frac{\partial w}{\partial x_n}$$

Котики!

GitHub

Tensorflow

Вопросы-ответы