

# Числовые алгоритмы. Матрицы. ML

@pvavilin

20 февраля 2022 г.

# Outline

# Умножение двух чисел

$$\begin{array}{r}
 \phantom{\times} 5678 \\
 \times 1234 \\
 \hline
 22712 \\
 17034 \phantom{0} \\
 11356 \phantom{00} \\
 5678 \phantom{000} \\
 \hline
 7006652
 \end{array}$$

(1)

*Можно ли лучше?*

# Умножение двоичных чисел

$$\begin{array}{r}
 \begin{array}{c} \Sigma \\ j \end{array} \downarrow \begin{array}{c} \text{+} \\ \text{+} \\ \text{+} \end{array} \begin{array}{c} 0\ 1\ 1\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 1\ 1\ 0 \\ 0\ 1\ 1\ 0 \end{array} \begin{array}{l} \leftarrow \sum_i a_i 2^i \\ \leftarrow \sum_i (a_i 2^i) * b_0 2^0 \\ \leftarrow \sum_i (a_i 2^i) * b_1 2^1 \\ \leftarrow \sum_i (a_i 2^i) * b_2 2^2 \\ \leftarrow \sum_i (a_i 2^i) * b_3 2^3 \end{array} \\
 \hline
 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0
 \end{array}$$

$\times$ 
 $\begin{array}{r} 0\ 1\ 1\ 0 \\ 1\ 1\ 0\ 1 \end{array}$ 
 $\begin{array}{l} \leftarrow \sum_i a_i 2^i \\ \leftarrow \sum_i b_i 2^i \end{array}$

# Алгоритм Каратцубы

$x = 5678$   
 $y = 1234$

$a = 56; b = 78$   
 $c = 12; d = 34$

# 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

$$\begin{array}{r}
 \phantom{\times} 5678 \\
 \times \phantom{0000} 1234 \\
 \hline
 \phantom{00} 22712 \\
 \phantom{0} 17034 \phantom{0} \\
 11356 \phantom{00} \\
 5678 \phantom{000} \\
 \hline
 7006652
 \end{array}$$

(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

$$P_1 = A(F - H),$$

$$P_2 = (A + B)H,$$

$$P_3 = (C + D)E,$$

$$P_4 = D(G - E),$$

$$P_5 = (A + D)(E + H),$$

$$P_6 = (B - D)(G + H),$$

$$P_7 = (A - C)(E + F)$$



# 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 novex vecexample.cpp  
echo "Без векторизации"  
./novex 100000000
```

Без векторизации

Time used for norm computation=6.1561  
Norm-2 = 1.5000

```
g++ -O3 -mavx2 -o vec vecexample.cpp  
echo "Используя векторизацию"  
./vec 100000000
```

Используя векторизацию

Time used for norm computation=5.0278  
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 100000000
```

```
Time used for norm computation = 7.42046  
Norm-2 = 1.50000
```

# Логистическая регрессия

$$z = w_0x + w_1x + \dots + w_nx + b$$

$$a = \frac{1}{1+e^{-z}}$$

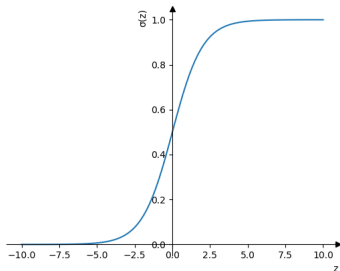


Рис.: sigmoid

# 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

Colab

# Вопросы-ответы

