FLOP Count Cheatsheet

Your average CS major (didnt shower for 10 years)

April 10, 2069

Summation Identities

Some summation identities for loops:

$$\begin{aligned} \sum_{k=1}^{n} 1 &= n & \sum_{k=1}^{n} k &= \frac{n(n+1)}{2} \\ \sum_{k=1}^{n} k^2 &= \frac{n(n+1)(2n+1)}{6} & \sum_{k=1}^{n} k^3 &= [\frac{n(n+1)}{2}]^2 \end{aligned} \tag{1}$$

Some algebraic properties that can be used in summations:

$$\sum_{k=1}^{n}(a+b) = \sum_{k=1}^{n}(a) + \sum_{k=1}^{n}(b) \qquad \sum_{k=1}^{n} c = c\sum_{k=1}^{n} 1 \tag{2}$$

Here is transforming limits in a summation with example:

$$\sum_{k=a}^{b} c_k = \sum_{l=1}^{b-a+1} c_{l+a-1}$$

Example:

$$\begin{aligned} &\sum_{k=3}^{27} 1 \\ =&\sum_{k=1}^{27-3+1} 1 \\ =&\sum_{k=1}^{25} 1 \end{aligned}$$

Another example:

$$\begin{aligned} &\sum_{j=2}^{i} 1 \\ =&\sum_{j=1}^{i-2+1} 1 \\ =&\sum_{j=1}^{i-1} 1 \end{aligned}$$

Counting FLOPs

How to count flops in a given algorithm. FLOPs are +, -, ÷ and ×. In summary, we can get the flop count by this:

1. Identify the loops in the algorithm.
2. Find the limits of each loop.
3. Count mathematical operations within each loop (memory access doesn't count).
4. Bring out any constants.
5. Transform the limits for any summation that doesn't start from 1.
6. Use identities to solve.
7. Simplify.

Some remarks, if your code contains any conditionals, transform your code to not include them because counting flops is very difficult with if statements. Another trick for counting FLOPs is putting a count variable in code and printing it to see a pattern. Below is some pseudocode with given python code:

Algorithm 1 Compute a matrix M to solve a system with n+1 data points

Input: $n \in \mathbb{N}$ and an $n+1 \times 1$ array of floats called $\mathbf{y}$ containing the values for the $n+1$ interpolating nodes.

let $M \in \mathbb{R}^{(n+1)\times(n+1)}$ and $\forall (M)_{ij} = 0$
for $i \leftarrow 1$ to n **do**
  $(M)_{i1} \leftarrow 1$
  for $j \leftarrow 2$ to i **do**
    $(M)_{ij} \leftarrow (M)_{i(j-1)} \cdot ((\mathbf{y})_i - (\mathbf{y})_{j-1})$
  end for
end for
return M

Output: M, an $n+1 \times n+1$ array of floats.

```
def NewtonPolyBuild(xs):
  n = len(xs)
  # print(n)
  M = np.zeros((n , n))
  M[:, 0] = 1.0

  # code for constructing the matrix M1
  for i in range(0, n):
      for j in range(1, i + 1):
          M[i, j] = (M[i, j - 1] * (xs[i] - xs[j-1]))
          # print(M[i, j])
  # print(M)
  return M
```

The "for i ← 1 to n" indicates a summation limit of $\sum_{i=1}^{n} 1$ is min and n is max. This is the outer loop, which is also indicated by range(0, n) in python but careful because python is n-1 due to zero-index. The inner loop is goes from 2 to i so the summation limit is $\sum_{j=2}^{i}$ where 2 is min and i is max, indicated by range(1, i+1). In the outer loop we are not doing any operations but in the inner loop we are doing 2 flops and accessing index doesn't count because we ignore memory access. So the setup becomes like this:

$$\sum_{i=1}^{n}\sum_{j=2}^{i} 2$$

Now we first change the boundaries:

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1} 2$$

Then we bring out any constants:

$$\sum_{i=1}^{n} 2 \sum_{j=1}^{i-1} 1 = 2 \sum_{i=1}^{n} \sum_{j=1}^{i-1} 1$$

Then using the the summation identities to solve inner loop:

$$2 \sum_{i=1}^{n} \sum_{j=1}^{i-1} 1 = 2 \sum_{i=1}^{n} i - 1$$

Then using algebraic properties, we distribute identities:

$$2 \sum_{i=1}^{n} \sum_{j=1}^{i-1} 1 = 2(\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1)$$

Then we solve again using identities:

$$2(\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1) = 2(\frac{n(n+1)}{2} - n)$$

Then we distribute any constants and simplify:

$$\begin{aligned} & 2(\tfrac{n(n+1)}{2} - n) \\ & = n^2 + n - 2n \\ & = n^2 - n \end{aligned}$$

Therefore the final flop count is $n^2 - n$ and the complexity is $O(n^2)$.

FLOP count for various algorithms

1. Computing sum of an array: $n-1$.
2. Computing a factorial: $n-2$.
3. Computing dot products: $2n-1$.
4. Computing a general matrix vector product: $2n^2-n$.
5. Computing product of two matrices: $2n^3-n^2$.
6. Gaussian Elimination: $\frac{2}{3}n^3+\frac{1}{2}n^2-\frac{7}{6}n$.
7. LU Decomposition without pivot: $\frac{2}{3}n^3+\frac{1}{2}n^2-\frac{7}{6}n$.
8. Back substitution: n^2.
9. Naive polynomial eval algo: $\frac{n^2+3n}{2}$.
10. Horner algorithm: $2n$.
11. Creating Vandermonde matrix: $\frac{1}{2}n^3+n^2+\frac{1}{2}n$.
12. Product of matrix vector using unit lower triangular matrix: n^2-3n+2.
13. Building a newton polynomial matrix: n^2-n.