Evaluating a Polynomial

Kasper Green Larsen

August 30, 2017

1 Fast Evaluation of a Polynomial

In this short note, we take a look at evaluating polynomials. A degree n polynomial P (over the real numbers $\mathbb{R}$) is given by $n+1$ coefficients $\alpha_n, \alpha_{n-1}, \dots, \alpha_1, \alpha_0 \in \mathbb{R}$. The evaluation of P at a point $x \in \mathbb{R}$ is simply:

$$P(x) = \alpha_n x^n + \alpha_{n-1}x^{n-1} + \cdots + \alpha_1 x + \alpha_0.$$

We will see two almost identical ways of evaluating P, but with dramatically different amounts of work to be done.

Naive. The first and most obvious solution is to simply evaluate each term $\alpha_i x^i$ in the sum individually and then sum them up. In pseudo code, this could look something like this:

```
1. $S \leftarrow 0$
2. For $i = 0, \dots, n$:
    (a) $B \leftarrow 1$.
    (b) For $d = 1, \dots, i$:
        i. $B \leftarrow B \cdot x$.
    (c) $S \leftarrow S + \alpha_i \cdot B$.
3. Return $S$.
```

Let us say a few words about the algorithm above. The variable S represents the sum of the terms $\alpha_i x^i$. The variable i represents which term $\alpha_i x^i$ we are about to evaluate. The inner-loop in steps (a) and (b) computes x^i and stores it as the variable B. Step (c) multiplies x^i with α_i and adds it to the sum.

How fast is the above algorithm? Let us take a look at how many times the "inner-most" instruction $B \leftarrow B \cdot x$ is executed. It is executed exactly:

$$\sum_{i=0}^{n}\sum_{d=1}^{i} 1$$

times. Notice the direct translation of a for-loop into a sum. This sum equals:

$$\begin{aligned}\sum_{i=0}^{n}\sum_{d=1}^{i} 1 &= \sum_{i=0}^{n} i \\ &= \sum_{i=1}^{n} i\end{aligned}$$

The sum of all integers from 1 to n is a well-known quantity, and equals $n(n+1)/2$. This is $n^2/2 + n/2$. The amount of work is thus roughly quadratic in the degree of the polynomial.

Re-Using Previous Work. We will now see how to speed up the evaluation significantly. Examining the naive algorithm, we see that we are computing x^i from skratch each time, even though we just computed x^{i-1} in the previous iteration. Since $x^i = x \cdot x^{i-1}$ we might as well exploit that we have computed x^{i-1}. A new algorithm implementing this idea is shown here:

```
1. $S \leftarrow 0$
2. $B \leftarrow 1$
3. For $i = 0, \ldots, n$:
    (a) $S \leftarrow S + \alpha_i \cdot B$.
    (b) $B \leftarrow B \cdot x$.
4. Return $S$.
```

As before, the idea is that B will represent B^i. For the first time we enter the for-loop with $i = 0$, we indeed have $B = 1 = x^0$. Once we've added the contribution of $\alpha_i x^i$ to the sum, we update B such that it is ready for the next iteration.

How much work are we performing this time? The loop has two steps (a) and (b), so the work is proportional to:

$$\sum_{i=0}^{n} 2 = 2(n+1).$$

So amount of work went from something quadratic in n to linear in n. Let us see how much that matters in practice. Assume we have modern computer that can execute about 1 billion instructions per second. The following table shows for how long we have to wait for the computer to finish evaluating a polynomial P of degree n for various values of n:

Degree n:	10^2	10^4	10^6	10^8
Naive ($n^2/2 + n/2$ work):	5 microseconds	50 milliseconds	8 minutes	2 months
Re-use ($2(n+1)$ work):	0.1 microseconds	20 microseconds	2 milliseconds	0.2 seconds

Table 1: Time for a modern computer to evaluate a polynomial of degree n using the two different approaches. The time is computed as time=(work/10^9) seconds. Recall that 1 microsecond is $1/10^6$ seconds and 1 millisecond is $1/10^3$ seconds.

Observe that as the input size (degree) n grows, the difference becomes more and more apparent. Once the input size reaches 10^8, we are talking a difference of less than one second compared to two months!