Fast Convolution

Michael Werman

School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
werman@cs.huji.ac.il

Abstract

We present a very simple and fast algorithm to compute the convolution of an arbitrary sequence x with a sequence of a specific type, a. The sequence a is any linear combination of polynomials, exponentials and trigonometric terms. The number of steps for computing the convolution depends on a certain *complexity* of a and not on its length, thus making it feasible to convolve a sequence with very large kernels fast.

Computing the convolution (correlation, filtering) of a sequence x together with a fixed sequence a is one of the ubiquitous operations in graphics, image and signal processing. Often the sequence a is a polynomial, exponential or trigonometric function sampled at discrete points or a piecewise sum of such terms, such as, splines, or else the sequence can can be well approximated with a few such terms. The computation of these convolutions is usually computed straight from the definition taking $O(|x||a|)$ time or using a more complicated FFT based $O(|x| \log |a|)$ time algorithm.

Here we present a simple and fast algorithm to compute the convolution of $x_1, x_2, \ldots, x_n$ with $a_m, a_{m-1}, \ldots, a_1$, namely $y_1, y_2, \ldots, y_{n-m}$ where $y_i = \sum_{k=1}^{m} a_k x_{i+k-1}$. The number of steps of the algorithm depends on a measure of complexity of a and not on m, its length. The number of steps to compute the convolution is $O(dn)$ $(m < n)$ where the sequence a satisfies a linear homogeneous equation, (LHE), $\sum_{i=0}^{d} \beta_i a_{r+i} = 0$ (where the β do not depend on r), or equivalently, $a_r = \sum_{i=1}^{d} \alpha_i a_{r+i}$. For d smaller than $\log |m|$ this is faster and much simpler than using FFT.

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972 WSCG(92)2003, February 3-7, 2003, Plzen, Czech Republic. Copyright UNION Agency (96) Science Press

Examples of such sequences are;

• polynomials of degree $d-1$, $a_i = \sum_{j=0}^{d-1} \lambda_j i^j$, the LHE is
$\sum_{j=0}^{d} (-1)^j \binom{d}{j} a_{i+j} = 0$, this is of complexity d.

• $a_i = \beta \lambda^i$, the LHE is $\lambda a_i - a_{i+1} = 0$, this is of complexity 2.

• $a_i = \lambda^i \sum_{j=0}^{d-1} \alpha_j i^j$, the LHE is
$\sum_{j=0}^{d} (-1)^j \binom{d}{j} a_{i+j} \lambda^{d-j} = 0$, this is of complexity d.

• $a_i = \alpha \sin(i\theta) + \beta \cos(i\theta)$, the LHE is $a_i - 2\cos(\theta) a_{i+1} + a_{i+2} = 0$, this is of complexity 3.

• $a_i = \lambda^i(\alpha \sin(i\theta) + \beta \cos(i\theta))$, the LHE is $\lambda^2 a_i - 2\lambda \cos(\theta) a_{i+1} + a_{i+2} = 0$, this is of complexity 3.

Sums of above like terms also satisfy a linear homogeneous equation with a complexity that is additive, such as $a_i = 3\sin(21\pi i/4) + (-2)^i + i^3 - 4$, this is of complexity $3 + 2 + 4 = 9$.

The complete algorithm, consists of two steps; initialization and the running computation:

```
• for i=1 ... $d$
    – $y_i = \sum_{k=1}^{m} a_k x_{i+k-1}$
    – $F_{d+1}^{d+1-i} = y_i - \sum_{k=1}^{d-i+1} a_k x_{i+k-1}$
      $+ \sum_{k=m}^{m+d-i+1} a_k x_{i+k-1}$.
• for i = $d+1$ ... $n-m$
    – $y_i = F_i^0 = \sum_{j=1}^{d} \alpha_j F_i^j$
    – for j = 1 ... $d$
        * $F_{i+1}^j = F_i^{j-1} - a_{j+1} x_i + a_{m+j+1} x_{m+i}$
```

The invariant is $F_i^j = \sum_{k=1}^{m} a_{j+k} x_{i+k-1}$.

The correctness of the algorithm follows from,
$\sum_{j=1}^{d} \alpha_j F_i^j = \sum_{j=1}^{d} \alpha_j \sum_{k=1}^{m} a_{j+k} x_{i+k-1}$
$= \sum_{k=1}^{m} (\sum_{j=1}^{d} \alpha_j a_{j+k}) x_{i+k-1}$
$= \sum_{k=1}^{m} a_k x_{i+k-1} = y_i$.

There is no reason to save all the previous F's so that the extra memory requirement over the input/output is just $O(d)$.

Notes:

• This is a special case of a recursive filter[Smi97], where it is easy to define the sequence a.
$y_i = \sum_{j=1}^{d} \alpha_j y_{i-j} - \sum_{k=1}^{d} (\sum_{l=1}^{d+1-k} a_l) x_{i-k}$
$+ \sum_{k=1}^{d} (\sum_{l=1}^{d+1-k} a_l) x_{m+i-k}$

• Special cases of LHE's with a fast algorithm such as the constant function (order 0 polynomial), box filtering, [SBHC88], and exponential functions have appeared before [Smi97, SBW02].

• The case of polynomials where the filter can be space variant was treated in [SBHC88, Hec86] using repeated integration and differentiation. They gave slightly different formula as they used a continuous set up.

Formulas equivalent to ours can be derived using finite differences [MT33] and the summation by parts formula: $\sum_{k=1}^{m} x(r+k)a(k)$
$= \sum_{l=0}^{d} (-1)^l x_{-l}(r+m+l) \Delta^l a(m)$.

x_{-l} (sum) is defined by, $x_0(i) = x(i)$ and $x_p(i) = \sum_{j=0}^{i} x_{p-1}(j)$, and $\Delta^l a(m)$ by $\Delta^0 a(m) = a(m)$ and $\Delta^{p+1} a(m) = \Delta^p a(m+1) - \Delta^p a(m)$.

$\Delta^{d+1} a(m) = 0$ whenever $a(m)$ is a sequence of equally spaced samples of a polynomial of degree d. So that as in [SBHC88, Hec86] once the sequences x_l and $\Delta^l a(m)$ for $l = 0..d$ are precomputed in $O(d)$ per element the computation of $\sum_{k=1}^{m} x(r+k)a(k)$ takes only $O(d)$ time.

$\Delta^l a(m)$ is especially easy to compute using that $\Delta^l \binom{x}{n} = \binom{x}{n-l}$ and that any polynomial can be written as $\sum \beta_j \binom{x}{j}$.

• The complexity of convolving with a sequence that is a piecewise sum of simple functions such as a spline can be computed by adding up the contributions of each of the pieces.

• Almost exactly the same same formulas can be used to compute the convolution for sequences defined by linear equations that are not homogeneous. This saves a few arithmetic operations but does not add any new functions, for if $\sum_{i=0}^{d} \beta_i a_{r+i} = c$ then subtracting two consecutive sums gives $\beta_0 a_r - \beta_d a_{r+d+1} + \sum_{i=1}^{d-1} (\beta_{i+1} - \beta i) a_{r+i} = 0$ a LHS of order $d+1$.

• Of course the algorithm can be used successively on rows then on columns for convolving 2-d[1], signals, images, whenever the function is separable in x and y and if the 1-d functions are simple, for example, $a(i,j) = 3\sin(\frac{2\pi i}{10})(2j^3 - j^2 + \cos(\frac{2\pi j}{20}))$. Then the amount of work for each output is just the sum of the two complexities.

The family of 2-d functions that satisfy a linear homogeneous equation is much larger than what can be computed with the method of separable functions, as it is possible to add arbitrary 1-d functions of linear combinations of x and y for free. The reason that we do not propose using a straight forward generalization of the the 1-d algorithm is that the updating of the F's (in the algorithm) now takes time proportional to the the perimeter of the signal which is now very large, $(O(\sqrt{m})$ in 2-d).

[1]or any dimension

Experiments

In order to test the algorithm we compared different convolutions with different algorithms. The algorithms were convolution using a straight forward implementation of the definition, convolution based on FFT and our proposed fast convolution. The code was written in JAVA and only the FFT based algorithm was optimized, time is in ms.

N	M	d	Conv	FFT	Fast
16384	16	3	160	611	110
16384	128	3	1212	751	110
16384	1024	3	9273	731	81
16384	2048	3	18066	751	101
8186	16	5	140	120	70
8186	128	5	991	101	70
8186	1024	5	8242	100	60
8186	2048	5	15332	90	50

References

[Hec86] P. Heckbert. Filtering by repeated integration. In *Proceedings of SIGGRAPH*, pages 317–321, 1986.

[MT33] L. M. Milne-Thomson. *The Calculus of Finite Differences.* Macmillan, 1933.

[SBHC88] P. Y. Simard, L. Bottou, P. Haffner, and Y. Le Cun. Boxlets: a fast convolution algorithm for signal processing and neural networks. *Neural Information Processing Systems, NIPS 11*, 1988.

[SBW02] H. Schweitzer, W. Bell, and F. Wu. Very fast template matching. In *ECCV*, pages 358–372, 2002.

[Smi97] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing, http://www.dspguide.com/, 1997.