Fast 1D Convolution Algorithms

P. Giannakeris, P. Bassia, N. Kilis, R.T. Chadoulis
Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr
www.aiia.csd.auth.gr
Version 2.6
Outline

• Signals and systems
• 1D convolutions
  Linear & Cyclic 1D convolutions
  Discrete Fourier Transform, Fast Fourier Transform
  Winograd algorithm
  Nested convolutions
  Block convolutions.
• Machine learning
  Convolutional neural networks.
Motivation

• Machine learning
  Fast implementation of 1D/2D/3D convolutions in Convolutional Neural Networks (CNNs).

• Fast implementation of 1D digital filters
  1D signal filtering (e.g., audio/music, ECG, EEG)
  1D Signal feature calculation

• Fast implementation of 1D correlation
  1D template matching
  Time-of-flight (distance) calculation (e.g., sonar)
Motivation

- Fast implementation of 2D/3D convolutions
  - Image/video filtering
  - Image/video feature calculation
    - Gabor filters
    - Spatiotemporal feature calculation

- Fast implementation of 2D correlation
  - Template matching
  - Correlation tracking
1D Linear Systems

- Linearity:
  \[ T[ax_1 + bx_2] = aT[x_1] + bT[x_2]. \]
- Shift-Invariance:
  \[ y(n) = T[x(n)] \Rightarrow y(n - m) = T[x(n - m)]. \]

LSI system convolution: \[ y(k) = h(k) * x(k). \]
1D Linear Systems

\[ x[n] \]

\[ x[n-2] \]
Linear 1D convolution

- The one-dimensional (linear) convolution of:
  - an input signal $x$ of length $L$ and
  - a convolution kernel $h$ (filter mask, finite impulse response) of length $M$:

$$y(k) = h(k) \ast x(k) = \sum_{i=0}^{M-1} h(i)x(k - i).$$

- For a convolution kernel centered around 0 and $M = 2v + 1$, convolution takes the form:

$$y(k) = h(k) \ast x(k) = \sum_{i=-v}^{v} h(i)x(k - i).$$
Linear 1D convolution

• The one-dimensional (linear) convolution is a linear operator.
• Signals $x, h$ can be interchanged:
  \[ y(k) = h(k) * x(k) = x(k) * h(k). \]
• It can model any linear shift-invariant system.
• It has a geometrical interpretation:
  • Signal is $x(i)$ flipped around $i = 0$ producing $x(-i)$.
  • Flipped signal $x(-i)$ is shifted by $k$ samples, producing $x(k - i)$.
  • Products $h(i)x(k - i), i = 0, \ldots, M - 1$ are calculated and summed.
  • This is repeated for other temporal shifts $k$. 
Linear 1D convolution – Example
Linear 1D convolution – Example
Linear 1D convolution – Example
Linear 1D convolution – Example

$x[m]$

$h[m]$

$x[m]$

$h[m-10]$

$x[m]h[m-10]$

$y[n]$
Linear 1D correlation

• Correlation of template signal $h$ and input signal $x(k)$ (inner product):

$$r_{hx}(k) = \sum_{i=0}^{M-1} h(i)x(k + i) = h^Tx(k).$$

• $h = [h(0), ..., h(M - 1)]^T$: template vector.

• $x(k) = [x(k), ..., x(k + M - 1)]^T$: local signal vector.
Linear 1D correlation

Differences from convolution:

• $x(k)$ is not flipped around $i = 0$.

• **It is often confused with convolution**: they are identical only if $h$ is centered at and is symmetric about $i = 0$.

• It is used for:
  • 1D template matching;
  • Time-delay estimation.
Linear 1D convolution: matrix notation

- Vectorial convolution input/output, kernel representation:
  \[ \mathbf{x} = [x(0), \ldots, x(L - 1)]^T : \text{the first input vector.} \]
  \[ \mathbf{h} = [h(0), \ldots, h(M - 1)]^T : \text{the second input vector.} \]
  \[ \mathbf{y} = [y(0), \ldots, y(N - 1)]^T : \text{the output vector, with } N = L + M - 1. \]

- 1D linear convolution between two discrete signals \( \mathbf{x}, \mathbf{h} \) can be expressed as the matrix-vector product:
  \[ \mathbf{y} = \mathbf{H}\mathbf{x}, \]
  where \( \mathbf{H} \) is a \( N \times L \) matrix.
Linear 1D convolution: matrix notation

- \( \mathbf{H} \): a \( N \times L \) band matrix of the form:

\[
\mathbf{H} = \begin{bmatrix}
 h(0) & 0 & \cdots & 0 \\
 h(1) & h(0) & \cdots & \cdots \\
 \vdots & \vdots & \ddots & \vdots \\
 h(M-1) & h(M-1) & \cdots & h(0) \\
 0 & 0 & \cdots & h(1) \\
 \vdots & \vdots & \ddots & \vdots \\
 0 & 0 & \cdots & h(M-1)
\end{bmatrix}
\]

- Alternative matrix notation: \( \mathbf{y} = \mathbf{X}\mathbf{h} \), where \( \mathbf{X} \) is an \( N \times M \) matrix.
- Fast calculation of the product \( \mathbf{y} = \mathbf{H}\mathbf{x} \) using BLAS/cuBLAS.
Optimal algorithms: minimal multiplicative complexity

- As multiplications are more expensive than additions, optimal algorithms having minimal multiplicative complexity have been designed.
- Example: Complex number multiplication normally involves four multiplications and two additions:
  \[ R + Ii = (a + bi)(c + di) = (ac - bd) + (bc + ad)i. \]
- Gauss trick: a way to reduce the number of multiplications to three (optimal algorithm): \( ac, bd, (a + b)(c + d) \), as: \( I = (a + b)(c + d) - ac - bd \).
- Other formulation:
  \[ k_1 = c(a + b), \quad k_2 = a(d - c), \quad k_3 = b(c + d), \quad R = k_1 - k_3, \quad I = k_1 + k_2. \]
Optimal Linear Convolution Algorithms

• Winograd theory for optimal linear convolution algorithms:
  • Minimal number of multiplications.

• The computation of the first $m$ outputs $y(0), \ldots, y(m-1)$ of an $n$-tap filter can be written as:

$$
\begin{bmatrix}
y(0) \\
y(1) \\
\vdots \\
y(m-1)
\end{bmatrix}
= 
\begin{bmatrix}
x(0) & x(1) & \cdots & x(n-1) \\
x(1) & x(2) & \cdots & x(n) \\
\vdots & \vdots & \ddots & \vdots \\
x(m-1) & x(m) & \cdots & x(m+n-2)
\end{bmatrix}
\begin{bmatrix}
h(0) \\
h(1) \\
\vdots \\
h(n-1)
\end{bmatrix}.
$$

• Notation consistent with literature.
Optimal Linear Convolution Algorithms

- Linear convolution can be written as a product $y = Xh$, where $X$ is an $m \times n$ matrix:
  \[ X_{ij} = x(i + j) , \quad 0 \leq i \leq m - 1, 0 \leq j \leq n - 1. \]

- Example: for input $x(k)$ with size $n = 4$, kernel $h(k)$ with size $m = 3$ and output $y(k)$ with size $r = 2$, the matrix vector product:
  \[
  \begin{bmatrix}
  y(0) \\
  y(1)
  \end{bmatrix} =
  \begin{bmatrix}
  x(0) & x(1) & x(2) \\
  x(1) & x(2) & x(3)
  \end{bmatrix}
  \begin{bmatrix}
  h(0) \\
  h(1) \\
  h(2)
  \end{bmatrix},
  \]
  requires $2 \times 3 = 6$ multiplications and 4 additions.
Optimal Linear Convolution Algorithms

- Example: optimal Winograd linear convolution algorithm.
- 8 additions and 4 multiplications.
- 3 additions on $h$ can be pre-calculated:

$$\begin{bmatrix} y(0) \\ y(1) \end{bmatrix} = \begin{bmatrix} x(0) & x(1) & x(2) \\ x(1) & x(2) & x(3) \end{bmatrix} \begin{bmatrix} h(0) \\ h(1) \\ h(2) \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix},$$

where:

$$m_1 = (x(0) - x(2))h(0), \quad m_2 = (x(1) + x(2))\frac{h(0)+h(1)+h(2)}{2},$$

$$m_4 = (x(1) - x(3))h(2), \quad m_3 = (x(2) - x(1))\frac{h(0)-h(1)+h(2)}{2}.$$
Optimal Linear Convolution Algorithms

Intermediate addition result is used 2 times.
Optimal Linear Convolution Algorithms

• Winograd linear convolution algorithm requires $m + r - 1$ multiplications, $m$ and $r$: lengths of $y$ and $h$, respectively.

• General form of optimal Winograd linear convolution algorithms:

$$y = A^T[(Hh) \otimes (B^Tx)],$$

$\otimes$ indicates element-wise $m + r - 1$ multiplications.

$x, h, y$: input signal, filter coefficient and output signal vectors.
Optimal Linear Convolution Algorithms-Example

Input signal, filter coefficient, output signal vectors for \( m = 3 \) and \( r = 2 \):

\[
\mathbf{x} = [x(0) \ x(1) \ x(2) \ x(3)]^T, \quad \mathbf{h} = [h(0) \ h(1) \ h(2)]^T, \quad \mathbf{y} = [y(0) \ y(1)]^T.
\]

The involved matrices are:

\[
\mathbf{B}^T = \begin{bmatrix}
1 & 0 & -1 & 0 \\
0 & 1 & 1 & 0 \\
0 & -1 & 1 & 0 \\
0 & 1 & 0 & -1 \\
\end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix}
1 & 0 & 0 \\
1/2 & 1/2 & 1/2 \\
1/2 & -1/2 & 1/2 \\
0 & 0 & 1 \\
\end{bmatrix},
\]

\[
\mathbf{A}^T = \begin{bmatrix}
1 & 1 & 1 & 0 \\
0 & 1 & -1 & -1 \\
\end{bmatrix}.
\]
Cyclic 1D convolution

• One-dimensional cyclic convolution of length $N$:

$$y(k) = x(k) \ast h(k) = \sum_{i=0}^{N-1} h(i)x((k - i)_N),$$

$$(k)_N = k \mod N.$$

• It is of no use in modeling linear systems.

• Important use: Embedding linear convolution in a **fast** cyclic convolution $y(n) = x(n) \ast h(n)$ of length $N \geq L + M - 1$ and then performing a cyclic convolution of length $N$. 
Linear convolution embedding

- Zero-padding signal vector $\mathbf{x}$ to length $N: \mathbf{x}_p = [x(0), \ldots, x(M - 1), 0, \ldots, 0]^T$.
- Zero-padding convolution kernel vector $\mathbf{h}$ to length $N: \mathbf{h}_p = [h(0), \ldots, h(L - 1), 0, \ldots, 0]^T$.
- Linear convolution embedding in a cyclic convolution of length $N = L + M - 1$:

![Diagram showing convolution process with zero-padding and cyclic convolution.](image)
Cyclic 1D convolution - Example

Cyclic convolution of $x(n) = [1 \ 2 \ 0]$ and $h(n) = [3 \ 5 \ 4]$.

$y(0) = 1 \times 3 + 2 \times 4 + 0 \times 5$  $y(1) = 1 \times 5 + 2 \times 3 + 0 \times 4$  $y(2) = 1 \times 4 + 2 \times 5 + 0 \times 3$
Cyclic 1D convolution: matrix notation

- Cyclic convolution definition as matrix-vector product:
  \[ y = Hx, \]

where:

- \( x = [x(0), ..., x(N - 1)]^T \): the input vector.
- \( y = [y(0), ..., y(N - 1)]^T \): the output vector.
- \( H \): a \( N \times N \) Toeplitz matrix of the form:

\[
H = \begin{bmatrix}
  h(0) & h(1) & h(2) & \cdots & h(N - 1) \\
  h(N - 1) & h(0) & h(1) & \cdots & h(N - 2) \\
  h(N - 2) & h(N - 1) & h(0) & \cdots & h(N - 3) \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  h(1) & h(2) & h(3) & \cdots & h(0)
\end{bmatrix}
\]
Cyclic Convolution via DFT

• Cyclic convolution calculation using 1D Discrete Fourier Transform (DFT):

\[ y = IDFT\left(DFT(x) \otimes DFT(h)\right). \]

• Fast calculation of DFT, IDFT through an FFT algorithm.
1D FFT

- There are various FFT algorithms to speed up the calculation of DFT.
- The best known is the radix-2 decimation-in-time (DIT) Fast Fourier Transform (FFT) (Cooley-Tuckey).

1. DFT of a sequence $x(n)$ of length $N$ ($n = 0, ..., N - 1$):

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi i}{N} nk}, \quad k = 0, ..., N - 1.$$ 

$N$-th complex roots of unity: $W_N^n = e^{-\frac{2\pi i}{N} n}, \quad n = 0, ..., N - 1$. 
2. The Radix-2 DIT algorithm rearranges the DFT:

\[ X(k) = \sum_{m=0}^{\frac{N}{2}-1} x(2m)e^{-\frac{2\pi i m k}{N}} + e^{-\frac{2\pi i k}{N}} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1)e^{-\frac{2\pi i m k}{N}} = E(k) + e^{-\frac{2\pi i k}{N}} O(k). \]

\[ \text{DFT of even-indexed part of } x(n) \quad \text{DFT of odd-indexed part of } x(n) \]

3. The complex exponential is periodic which means that:

\[ X(k + \frac{N}{2}) = \sum_{m=0}^{\frac{N}{2}-1} x(2m)e^{-\frac{2\pi i m (k+\frac{N}{2})}{N}} + e^{-\frac{2\pi i (k+\frac{N}{2})}{N}} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1)e^{-\frac{2\pi i m (k+\frac{N}{2})}{N}} \]

\[ = \sum_{m=0}^{\frac{N}{2}-1} x(2m)e^{-\frac{2\pi i m k}{N}} e^{-2\pi i m} + e^{-\frac{2\pi i k}{N}} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1)e^{-\frac{2\pi i m k}{N}} e^{-2\pi i m} \]

\[ = \sum_{m=0}^{\frac{N}{2}-1} x(2m)e^{-\frac{2\pi i m k}{N}} - e^{-\frac{2\pi i k}{N}} \sum_{m=0}^{\frac{N}{2}-1} x(2m+1)e^{-\frac{2\pi i m k}{N}} = E(k) - e^{-\frac{2\pi i k}{N}} O(k). \]
4. We can rewrite $X$ as:

$$X(k) = E(k) + e^{-\frac{2\pi i k}{N}} O(k).$$

$$X\left(k + \frac{N}{2}\right) = E(k) - e^{-\frac{2\pi i k}{N}} O(k).$$

5. By inserting a recursion, we can compute in that way $X(k)$ and $X(k + \frac{N}{2})$ as well. The algorithm gains its speed by re-using the results of intermediate computations to compute multiple DFT outputs.
1D FFT

- radix-2 FFT breaks a length-$N$ DFT into many size-2 DFTs called "butterfly" operations.
- There are $\log_2 N$ FFT stages.
Radix-2 1D FFT properties

- DFT length $N$ should be a power of 2.
- The nice FFT structure is based on the properties of the $N$-th complex roots of unity $W_N^n = e^{-\frac{2\pi i}{N}n}$, $n = 0, ..., N - 1$.
- Computational complexity of the 1D FFT is $O(N \log_2 N)$.
- Computational complexity of the 1D FFT is $O(N^2)$.
- Other FFT algorithms: Radix-4 FFT ($N$ is power of 4), Prime Factor Algorithm (PFA) $N = PQ$, $P, Q$ co-prime numbers.
Cyclic 1D convolution computation

• Its computation by definition requires:
  • Two nested for loops
  • Computational complexity (number of multiplications) is $O(N^2)$.

• Computational complexity through two 1D FFTs and one inverse 1D FFT is $O(N \log_2 N)$.

• Fast linear convolution calculation through cyclic convolution embedding.
The $Z$ transform of a discrete signal $x(n)$ having domain $[0, ..., N]$ is given by:

$$X(z) = \sum_{n=0}^{N-1} x(n)z^{-n}.$$ 

The domain of $Z$ transform is the complex plane, since $z$ is a complex number.

Convolution property of the $Z$ transform (polynomial product $X(z)H(z)$):

$$y(n) = x(n) \ast h(n) \iff Y(z) = X(z)H(z).$$
Cyclic convolution and $Z$ transform

Polynomial product form of the 1D cyclic convolution:

\[ y(k) = x(k) \ast h(k) = \sum_{i=0}^{N-1} h(i)x((k - i)_N), \]

where: \( (k)_N = k \mod N. \)

\[ y(k) = x(k) \ast h(k) \iff Y(z) = X(z)H(z) \mod z^N - 1. \]
Convolutions as polynomial products-examples

• 1D linear convolution $y(k) = h(k) \ast x(k)$ of length $L = M = 3$:

$$Y(z) = X(z)H(z) = [x(0) + x(1)z + x(2)z^2][h(0) + h(1)z + h(2)z^2].$$

• 1D cyclic convolution $y(k) = x(k) \bigotimes h(k)$ of length $N = 3$:

$$Y(z) = X(z)H(z) \mod z^3 - 1$$

$$= [x(0) + x(1)z + x(2)z^2][h(0) + h(1)z + h(2)z^2] \mod z^3 - 1.$$

Factorization: $z^3 - 1 = P_1(z)P_2(z) = (z - 1)(z^2 + z + 1)$ ($\nu = 2$ factors).
Chinese Remainder Theorem (CRT)

If we have a polynomial $Y(z)$ and:

$P(z) = z^N - 1$, where the Greatest Common Diviser $GCD[P_i(z), P_j(z)] = 1$ for $i \neq j$.

$P(z) = \prod_{i=1}^{\nu} P_i(z)$.

$Y_i(z) = Y(z) \mod P_i(z)$.

$0 < \text{deg}[Y(z)] \leq \sum_{i=1}^{\nu} \text{deg}[P_i(z)]$, where $\text{deg}$ is the degree of a polynomial.

$R_i(z) = d_{ij} \mod P_i(z), 1 \leq i \leq \nu$.

Then the polynomial $Y(z)$ is given by: $Y(z) = \sum_{i=1}^{\nu} R_i(z)Y_i(z) \mod z^N - 1$
From CRT to Winograd convolution algorithm

CRT helps us to create fast convolution algorithms by embedding linear convolutions in cyclic ones:

\[ Y(z) = H(z)X(z) \mod z^N - 1. \]

A 'divide-and-conquer' approach can be used for creating fast convolution algorithms, by factorizing polynomial \( z^N - 1 \) to its prime factors \( P_i(z) \).

Then, we calculate first the items:

\[ X_i(z) = X(z) \mod P_i(z), \quad i = 1, \ldots, \nu \]
\[ H_i(z) = H(z) \mod P_i(z), \quad i = 1, \ldots, \nu \]

and their products:

\[ Y_i(z) = X_i(z)H_i(z) \mod P_i(z), \quad i = 1, \ldots, \nu. \]
Winograd algorithm
Fast 1D cyclic convolution with minimal complexity

• Winograd convolution algorithms or fast filtering algorithms:
  \[ y = C(AX \otimes Bh). \]

• They require only \( 2N - \nu \) multiplications in their middle vector product, thus having minimal complexity.

• \( \nu \): number of cyclotomic polynomial factors of polynomial \( z^N - 1 \) over the rational numbers \( \mathbb{Q} \).
Example: Winograd Cyclic convolution Algorithm for N=3

I. Definitions:

\[ X(z) = \sum_{n=0}^{N-1} x_n z^n \Rightarrow X(z) = \sum_{m=0}^{2} x_m z^m \Rightarrow X(z) = x_0 + x_1 z + x_2 z^2 \]

\[ H(z) = \sum_{n=0}^{N-1} h_n z^n \Rightarrow H(z) = \sum_{m=0}^{2} h_m z^m \Rightarrow H(z) = h_0 + h_1 z + h_2 z^2 \]

\[ Y(z) = H(z) \cdot X(z) \mod (z^N - 1) \Rightarrow Y(z) = H(z) \cdot X(z) \mod (z^3 - 1) \]

\[ z^N - 1 = \prod_{c \mid N} c_d(z) \Rightarrow z^3 - 1 = c_1(z) \cdot c_3(z) \Rightarrow z^3 - 1 = (z - 1)(z^2 + z + 1) \]

\[ Y(z) = H(z) \cdot X(z) \mod (z - 1)(z^2 + z + 1) \]
Example: Winograd Cyclic convolution Algorithm for N=3

II. Computing the reduced Polynomials $X_i, H_i$:

$X_i = X(z) \mod P_i \Rightarrow X_1 = (x_0 + x_1z + x_2z^2) \mod (z - 1) \Rightarrow \boxed{X_1 = x_0 + x_1 + x_2}$

$\Rightarrow X_2 = (x_0 + x_1z + x_2z^2) \mod (z^2 + z + 1) \Rightarrow \boxed{X_2 = (x_0 - x_2) + (x_1 - x_2)z}$

Let $a_0 = x_1 + x_2 + x_3$, $a_1 = x_0 - x_2$ and $a_2 = x_1 - x_2$ :

$X_1 = a_0$ and $X_2 = a_1 + a_2z$

Similarly for $H_i$ : $H_1 = h_0 + h_1 + h_2 = b_0$ and $H_2 = (h_0 - h_2) + (h_1 - h_2)z = b_1 + b_2z$
Example: Winograd Cyclic convolution Algorithm for N=3

III. Computing the products $Y_i$:

$Y_i = X_i(z)H_i(z) \mod P_i(z) \Rightarrow Y_1 = (a_0 \cdot b_0) \mod (z-1) \Rightarrow Y_1 = a_0 \cdot b_0$

$Y_2 = X_2(z)H_2(z) \mod P_2(z) \Rightarrow$

$Y_2 = [(a_1 + a_2z)(b_1 + b_2z)] \mod (z^2 + z + 1) \Rightarrow$

$Y_2 = (a_1b_2 - a_2b_1) + z \cdot (a_1b_2 + a_2b_1 - a_2b_2)$

One can see, that: $a_1b_2 + a_2b_1 - a_2b_2 = -(a_1 - a_2)(b_1 - b_2) + a_1b_1$, thus, $Y_2$ becomes:

$Y_2 = (a_1 \cdot b_1 - a_2 \cdot b_2) + z \cdot [- (a_1 - a_2)(b_1 - b_2) + a_1b_1]$

Let $c_0 = a_0b_0$, $c_1 = a_1b_1 - a_2b_2$ and $c_2 = -(a_1 - a_2)(b_1 - b_2) + a_1b_1$

$Y_1 = c_0$ and $Y_2 = c_1 + c_2 \cdot z$
Example: Winograd Cyclic convolution Algorithm for N=3

IV. Using the Chinese Remainder Theorem (CRT) to Compute $Y$:

$$R_1 = \frac{p - c_p(z)}{p} \Rightarrow R_1 = \frac{3 - c_3(z)}{3} \Rightarrow R_1 = \frac{3 - z^2 - z - 1}{3}$$

$$R_2 = \frac{c_p(z)}{p} \Rightarrow R_2 = \frac{c_3(z)}{3} \Rightarrow R_2 = \frac{z^2 + z + 1}{3}$$

$$Y = \sum_{i=1}^{2} R_i Y_i \mod (z^N - 1) \Rightarrow Y = \ldots = \frac{1}{3} [ (c_0 + 2c_1 - c_2) + (c_0 - c_1 + 2c_2) \cdot z + (c_0 - c_1 - c_2) \cdot z^2 ]$$
Example: Winograd Cyclic convolution Algorithm for $N=3$

V. Finding Matrices $A$ and $B$:

\[
\begin{bmatrix}
  a_0 \\
  a_1 \\
  a_2 \\
  a_1 - a_2
\end{bmatrix}
= A \cdot
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2 \\
  x_3
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
  x_0 + x_1 + x_2 \\
  x_0 - x_2 \\
  x_1 - x_2 \\
  x_0 - x_1
\end{bmatrix}
= A \cdot
\begin{bmatrix}
  x_0 \\
  x_1 \\
  x_2
\end{bmatrix}
\Rightarrow
A =
\begin{bmatrix}
  1 & 1 & 1 \\
  1 & 0 & -1 \\
  0 & 1 & -1 \\
  1 & -1 & 0
\end{bmatrix}
\]

Similarly, we get exactly the same result for matrix $B$!
Example: Winograd Cyclic convolution Algorithm for N=3

VI. Finding matrix $C$:

$$
\begin{bmatrix}
Y_1 \\
Y_2 \\
Y_3
\end{bmatrix}
=\begin{bmatrix}
c_0 + 2c_1 - c_2 \\
c_0 - c_1 + 2c_2 \\
c_0 - c_1 - c_2
\end{bmatrix}
=\begin{bmatrix}
a_0b_0 + a_1b_1 - 2a_2b_2 + (a_1 - a_2) \cdot (b_1 - b_2) \\
a_0b_0 + a_1b_1 + a_2b_2 - 2(a_1 - a_2) \cdot (b_1 - b_2) \\
a_0b_0 - 2a_1b_1 + a_2b_2 + (a_1 - a_2) \cdot (b_1 - b_2)
\end{bmatrix}
$$

$$
M_{(4 \times 1)} = (A \cdot x) \otimes (B \cdot h) = \begin{bmatrix}
1 & 1 & 1 \\
1 & 0 & -1 \\
0 & 1 & -1 \\
1 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
x_0 \\
x_1 \\
x_2
\end{bmatrix}
\otimes
\begin{bmatrix}
1 & 1 & 1 \\
1 & 0 & -1 \\
0 & 1 & -1 \\
1 & -1 & 0
\end{bmatrix}
\begin{bmatrix}
h_0 \\
h_1 \\
h_2
\end{bmatrix}
= \begin{bmatrix}
a_0b_0 \\
a_1b_1 \\
a_2b_2 \\
(a_1 - a_2)(b_1 - b_2)
\end{bmatrix}
$$
Example: Winograd Cyclic convolution Algorithm for N=3

VI. Finding matrix C:

\[ Y_{(3x1)} = C_{(3x4)} \cdot M_{(4x1)} \Rightarrow \]

\[
\begin{bmatrix}
    a_0b_0 + a_1b_1 - 2a_2b_2 + (a_1 - a_2) \cdot (b_1 - b_2) \\
    a_0b_0 + a_1b_1 + a_2b_2 - 2(a_1 - a_2) \cdot (b_1 - b_2) \\
    a_0b_0 - 2a_1b_1 + a_2b_2 + (a_1 - a_2) \cdot (b_1 - b_2)
\end{bmatrix}
\]

\[
\begin{bmatrix}
    C_{00} & C_{01} & C_{02} & C_{03} \\
    C_{10} & C_{11} & C_{12} & C_{13} \\
    C_{20} & C_{21} & C_{22} & C_{23}
\end{bmatrix}
\cdot
\begin{bmatrix}
    a_0b_0 \\
    a_1b_1 \\
    a_2b_2 \\
    (a_1 - a_2)(b_1 - b_2)
\end{bmatrix}
\Rightarrow
\]

\[
\begin{bmatrix}
    1 & 1 & -2 & 1 \\
    1 & 1 & 1 & -2 \\
    1 & -2 & 1 & 1
\end{bmatrix}
\Rightarrow C
\]

Theoretically minimal number of multiplications: 4!
Winograd algorithm version with minimal computational complexity

- Can be equivalently expressed as:
  \[ y = RB^T(Ax \otimes C^T Rh) \].
- Matrices \( A, B \) typically have elements \( 0, +1, -1 \).
- Multiplications \( C^T Rh, RB^T y' \) are done only by additions/subtractions.
- \( R \) is an \( N \times N \) permutation matrix.
- \( Rh \) can be precomputed.
Block diagram: Winograd Cyclic convolution Algorithm for N=3
Block diagram: Winograd Cyclic convolution Algorithm for $N=3$
Winograd algorithm
Fast 1D cyclic convolution with minimal complexity

• Winograd algorithm works on small blocks of the input signal.
• The input block and filter are transformed.
• The outputs of the transform are multiplied together in an element-wise fashion.
• The result is transformed back to obtain the outputs of the convolution.
• GEneral Matrix Multiplication (GEMM) BLAS or CUBLAS routines can be used.
Nested convolutions

- Winograd algorithms exist for relatively short convolution lengths, e.g.: $N = 3, 5, 7$.
- Use of efficient short-length convolution algorithms iteratively to build long convolutions.
- Does not achieve minimal multiplication complexity.
- Good balance between multiplications and additions.

Decomposition of 1D convolution into a 2D convolution:
- 1D convolution of length: $N = N_1 N_2$
- with $N_1, N_2$ co-prime integers, $(N_1, N_2) = 1$
- results into a 2D $N_1 \times N_2$ convolution.
Nested convolutions

- Remember circular convolution definition:

\[ y(l) = \sum_{n=0}^{N-1} h(n)x((l - n)N), \quad l = 0, \ldots, N - 1. \]

- If \( N = N_1 N_2 \), \( l \) and \( n \) can be analyzed as:

\[
\begin{align*}
    l &\equiv N_1 l_2 + N_2 l_1 \quad \text{mod} \ N_1 N_2 \quad l_1, n_1 = 0, \ldots, N_1 - 1. \\
    n &\equiv N_1 n_2 + N_2 n_1 \quad \text{mod} \ N_1 N_2 \quad l_2, n_2 = 0, \ldots, N_2 - 1.
\end{align*}
\]

- 1D circular convolution becomes now a 2D \( N_1 \times N_2 \) convolution:

\[
    y(N_1 l_2 + N_2 l_1) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} h(N_1 n_2 + N_2 n_1)x(N_1 (l_2 - n_2) + N_2 (l_1 - n_1)).
\]
Nested convolutions

- The number of multiplications required are $M_1M_2$, where, $M_1$ and $M_2$ is the number of multiplications for convolutions $N_1$ and $N_2$ accordingly.
- The same applies for the reverse decomposition: $N_2 \times N_1$.
- However, the number of additions varies over the nesting order:
  - $N_1 \times N_2$: $A_1N_2 + M_1A_2$
  - $N_2 \times N_1$: $A_2N_1 + M_2A_1$,
with $A_1$ and $A_2$ being the number of additions required for $N_1$ and $N_2$ accordingly.
- Additions must be calculated beforehand, so as to choose the nesting order.
Nested convolutions

For example, let's assume we have the circular convolution of length $N = 12$ and $N_1 = 3$, $N_2 = 4$:

then,

$$l_1, n_1 = [0, 1, 2]$$
$$l_2, n_2 = [0, 1, 2, 3]$$

and

$$X_0(z) = x_0 + x_3z + x_6z^2 + x_9z^3$$
$$X_1(z) = x_4 + x_7z + x_{10}z^2 + x_1z^3$$
$$X_2(z) = x_8 + x_{11}z + x_2z^2 + x_5z^3$$

$$H_0(z) = h_0 + h_3z + h_6z^2 + h_9z^3$$
$$H_1(z) = h_4 + h_7z + h_{10}z^2 + h_1z^3$$
$$H_2(z) = h_8 + h_{11}z + h_2z^2 + h_5z^3$$
Nested convolutions

$X$ and $H$ can be defined now as $3\times 4$ matrices with the polynomial terms as elements:

$$X = \begin{bmatrix} x_0 & x_3 & x_6 & x_9 \\ x_4 & x_7 & x_{10} & x_1 \\ x_8 & x_{11} & x_2 & x_5 \end{bmatrix}$$

$$H = \begin{bmatrix} h_0 & h_3 & h_6 & h_9 \\ h_4 & h_7 & h_{10} & h_1 \\ h_8 & h_{11} & h_2 & h_5 \end{bmatrix}$$
Nested convolutions

• We use Winograd 3-point circular convolution $C$, $A$, and $B$ matrices at the outer nesting layer:
  \[ y = C(AX \otimes BH) \]

• The matrix-vector products $Ax$ and $Bh$, are replaced by matrix multiplication $AX$ and $BH$ respectively, each one resulting to a 4×4 matrix.

• We use a fast 4-point Winograd routine to calculate the 4 circular convolutions $AX \otimes BH$ (one per row). The 4-point routine is “nested” inside the 3-point routine.

• The correct order of the elements in the result can be found based on the nesting formulation:
  \[ y_{N_1 l_2 + N_2 l_1} \]
Block Based convolutions

- Input signal $x(n)$ is split in overlapping/non-overlapping blocks.
- Blocks are convolved independently.
- Great parallelism is achieved.

- Two block-based convolution methods:
  - Overlap-add method
  - Overlap-save method.
Overlap-add method

It uses the following signal processing principles:

• The linear convolution of two discrete-time signals of length $L$ and $M$ results in a discrete-time convolved signal of length $N = L + M - 1$.

• Additivity:

\[
[x_1(n) + x_2(n)] * h(n) = x_1(n) * h(n) + x_2(n) * h(n).
\]
1. Break the input signal $x(n)$ into **non-overlapping blocks** $x_m(n)$ of length $L$.
2. Zero pad $h(n)$ to be of length $N = L + M - 1$.
   Calculate the DFT $H(k), k = 0, 1, \ldots, N - 1$.
3. For **each block** $m$:
   - Zero pad $x_m(n)$ to be of length $N = L + M - 1$. Calculate the DFT $X_m(k)$.
   - Use the IDFT of $H(k)X_m(k)$ for block output $y_m(n), n = 0, 1, \ldots, N - 1$.
4. Form **overall output** $y(n)$ by overlapping the last $M - 1$ samples of $y_m(n)$ with the first $M - 1$ samples $y_{m+1}(n)$ and adding the result.
Overlap-add method

\[ L \quad \text{zeros} \quad L \quad \text{zeros} \quad L \]

\[ x_1(n) \quad (M - 1) \text{ zeros} \quad x_2(n) \quad (M - 1) \text{ zeros} \quad x_3(n) \]

\[ y_1(n) \quad (M - 1) \text{ points} \quad y_2(n) \quad (M - 1) \text{ points} \quad y_3(n) \]

\[ (M - 1) \text{ points} \quad (M - 1) \text{ points} \quad (M - 1) \text{ points} \]
Overlap-add method

Block x1

Block y1

Block x2

Block y2

Block x3

Block y3

Block x4

Block y4
Overlap-add method

Input sequence

Impulse response sequence

Fast Convolution using Overlap-Add method
Overlap-save method

It uses the following signal processing principles:

• The linear convolution of two discrete-time signals of length $N$ and $M$ results in a discrete-time convolved signal of length $N = L + M - 1$.

• Time-Domain Aliasing:

$$x_c(n) = \sum_{l=-\infty}^{\infty} x_L(n - lN), \quad n = 0, 1, \ldots, N - 1.$$
Overlap-save method

Overlap-Save method:
1. Pad $M - 1$ zeros at the beginning of the input sequence $x(n)$.
2. Break the padded input signal into overlapping blocks $x_m(n)$ of length $N = L + M - 1$, where the overlap length is $M - 1$.
3. Zero-pad $h(n)$ to be of length $N = L + M - 1$.
   Calculate the $DFT H(k), k = 0,1, \ldots, N - 1$.
4. For each block $m$:
   • Calculate the $DFT X_m(k), k = 0,1, \ldots, N - 1$.
   • Calculate the IDFT of: $Y_m(k) = X_m(k) H(k), k = 0,1, \ldots, N - 1$ to get the block output $y_m(n), n = 0,1, \ldots, N - 1$.
5. Discard the first $M - 1$ points of each output block $y_m(n)$. Concatenate the remaining (i.e., last) $L$ samples of each block $y_m(n)$ to form the output $y(n)$. 
Overlap-save method

Input signal blocks

Output signal blocks

- $L \rightarrow L \rightarrow L$
- $(M - 1)$ zeros
- $(M - 1)$ point overlap
- $(M - 1)$ point overlap
- Discard $(M - 1)$ points
Overlap-save method
Overlap-save method

Input sequence

Impulse response sequence

Fast Convolution using Overlap-Save method
Applications

- Convolutional neural networks
- Signal processing
  - Signal filtering
  - Signal restoration
  - Signal deconvolution
- Signal analysis
  - Time delay estimation
  - Distance calculation (e.g., sonar)
  - 1D template matching
Convolutional Neural Networks

Convergence of machine learning and signal processing

- Two step architecture:
  - First layers with sparse NN connections: convolutions.
  - Fully connected final layers.
- Need for fast convolution calculations.

![Diagram of a convolutional neural network](image)
# Deep Learning Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>User Interface</th>
<th>Data Parallelism</th>
<th>Model Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>protobuf, C++, Python</td>
<td>Yes</td>
<td>Limited</td>
</tr>
<tr>
<td>CNTK</td>
<td>BrainScript, C++, C#</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>Python, C++</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Theano</td>
<td>Python</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Torch</td>
<td>LuaJIT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Image Source: Heehoon Kim, Hyoungwook Nam, Wookeun Jung, and Jaejin Le - Performance Analysis of CNN Frameworks for GPUs
Convolutions in DL Frameworks

- All 5 frameworks work with cuDNN as backend.
- cuDNN unfortunately not open source.
- cuDNN supports FFT and Winograd convolutions.

<table>
<thead>
<tr>
<th>Framework</th>
<th>User Selectable</th>
<th>Heuristic-based</th>
<th>Profile-based</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Heuristic-based</td>
</tr>
<tr>
<td>CNTK</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Profile-based</td>
</tr>
<tr>
<td>TensorFlow</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Heuristic-based†</td>
</tr>
<tr>
<td>Theano</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>GEMM</td>
</tr>
<tr>
<td>Torch</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>GEMM</td>
</tr>
</tbody>
</table>

†TensorFlow uses its own heuristic algorithm

Image Source: Heehoon Kim, Hyoungwook Nam, Wookeun Jung, and Jaejin Le - Performance Analysis of CNN Frameworks for GPUs
Basic Exercises

1. Implement the definition of the 1D linear convolution algorithm in C/C++ or Python.
2. Implement the definition of the 1D DFT algorithm in C/C++ or Python.
3. Implement the 1D cyclic convolution algorithm for length $N = 2^n$ in C/C++ using FFT.
4. Derive analytically the 1D Winograd cyclic convolution algorithm for length $N = 3$.
5. Implement the 1D Winograd cyclic convolution algorithm for length $N = 3$ in C/C++ or Python.
Advanced Exercises

1. Implement the 1D linear convolution for filter length $M$ and signal length $L$ in C/C++ using your own myFFT routine.

2. Implement a nested convolution algorithm for length $N = N_1 N_2$ in C/C++.

3. Study the properties of cyclotomic polynomials. Derive analytically the 1D Winograd cyclic convolution algorithm for length $N = 4, N = 6$ or $N = 9$.

4. Implement the 1D linear convolution algorithm for filter length $M$ and a large signal length $L$ using overlap-add in a) C/C++ or b) CUDA.
References

Q & A

Thank you very much for your attention!

Contact: Prof. I. Pitas
pitas@csd.auth.gr