An Introduction to Convolution

What is Convolution? An Intuitive View I

At its heart, convolution is a mathematical operation that describes how the shape of one function is modified by another. It's best understood as a **sliding, weighted average**.
Imagine a signal or function $\mathbf{f}$ and a "kernel" or filter $\mathbf{g}$. The convolution $(\mathbf{f} * \mathbf{g})$ at a point t is the integral of the product of the two functions after one is reversed and shifted.

$$(\mathbf{f} * \mathbf{g})(t) = \int_{-\infty}^{\infty} \mathbf{f}(\tau)\mathbf{g}(t-\tau)d\tau$$

What is Convolution? An Intuitive View II

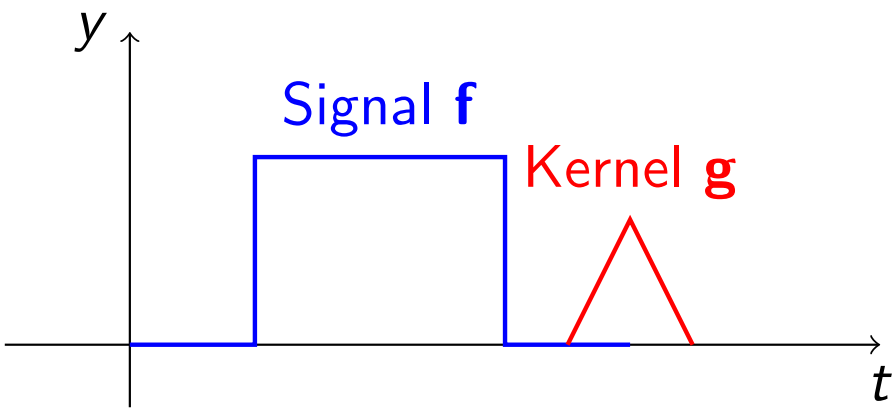

Figure: Convolution smooths or modifies the original signal based on the shape of the kernel.

▸ Link

Importance: Convolution is the cornerstone of signal processing, image filtering (blur, sharpen, edge detection), and modern deep learning (Convolutional Neural Networks).

From Convolution to Matrices I

In the discrete world, convolution can be expressed as a matrix-vector product. This reveals special, highly structured matrices.

Toeplitz Matrices: Have constant values along each descending diagonal. They represent linear, time-invariant systems.

$$\mathbf{T} = \begin{pmatrix} a & b & c & d \\ e & a & b & c \\ f & e & a & b \\ g & f & e & a \end{pmatrix}$$

From Convolution to Matrices II

Circulant Matrices: A special case of Toeplitz where each row is a **cyclic shift** of the row above it. The entire matrix is defined by its first row $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$.

$$\mathbf{C} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ c_3 & c_0 & c_1 & c_2 \\ c_2 & c_3 & c_0 & c_1 \\ c_1 & c_2 & c_3 & c_0 \end{pmatrix}$$

Key Property: The product of two circulant matrices is circulant, and they commute ($\mathbf{CD} = \mathbf{DC}$).

Circulant Matrices as Polynomials I

A circulant matrix can be represented as a polynomial in the basic **cyclic shift matrix P**.

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{P}^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{P}^3 = \ldots, \quad \mathbf{P}^4 = \mathbf{I}$$

Any circulant matrix $\mathbf{C}$ with first row $(c_0, c_1, \ldots, c_{n-1})$ can be written as:

$$\mathbf{C} = c_0\mathbf{I} + c_1\mathbf{P} + c_2\mathbf{P}^2 + \cdots + c_{n-1}\mathbf{P}^{n-1}$$

This is the polynomial $p(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$ evaluated at the matrix $\mathbf{P}$, so $\mathbf{C} = p(\mathbf{P})$.

Circulant Matrices as Polynomials II

Link with Polynomial Product: Multiplying two circulant matrices $\mathbf{C} = p(\mathbf{P})$ and $\mathbf{D} = q(\mathbf{P})$ corresponds to multiplying their polynomials:

$$\mathbf{CD} = p(\mathbf{P})q(\mathbf{P}) = r(\mathbf{P})$$

where $r(x) = p(x)q(x) \pmod{x^n - 1}$. This product performs **cyclic convolution** on the first rows of the matrices.

Example: Cyclic vs. Non-Cyclic Convolution

Let $\mathbf{p} = (1, 2, 3)$ and $\mathbf{q} = (4, 5, 0)$. The associated polynomials are $p(x) = 1 + 2x + 3x^2$ and $q(x) = 4 + 5x$.

Non-Cyclic (Standard) Product:

$$p(x)q(x) = (1 + 2x + 3x^2)(4 + 5x) = 4 + 13x + 22x^2 + 15x^3$$

The resulting coefficient vector is $(4, 13, 22, 15)$.

Cyclic Product (for $n = 3$): We compute the product modulo $(x^3 - 1)$. Since $x^3 \equiv 1 \pmod{x^3 - 1}$:

$$p(x)q(x) = 4 + 13x + 22x^2 + 15(1) = (4 + 15) + 13x + 22x^2$$

$$\implies r(x) = 19 + 13x + 22x^2$$

The resulting coefficient vector is $(19, 13, 22)$. This is the first row of the product of the circulant matrices for $\mathbf{p}$ and $\mathbf{q}$.

Eigenvalues and Eigenvectors of Circulant Matrices

T

he eigenvectors of **all** $n \times n$ circulant matrices are the same! They are the columns of the **Fourier Matrix F**.

Let $\omega = e^{2\pi i/n}$ be the n-th primitive root of unity. The k-th eigenvector $\mathbf{v}_k$ has components $v_{j,k} = \omega^{jk}$ for $j = 0, \ldots, n-1$.

$$\mathbf{v}_k = \begin{pmatrix} 1 \\ \omega^k \\ \omega^{2k} \\ \vdots \\ \omega^{(n-1)k} \end{pmatrix}$$

The eigenvalue λ_k corresponding to $\mathbf{v}_k$ is the k-th component of the Discrete Fourier Transform (DFT) of the matrix's first row, $\mathbf{c}$:

$$\lambda_k = c_0 + c_1\omega^{-k} + c_2\omega^{-2k} + \cdots + c_{n-1}\omega^{-(n-1)k} = \sum_{j=0}^{n-1} c_j\omega^{-jk}$$

Proof for a 4x4 Matrix

Let $\mathbf{C}$ be a 4x4 circulant matrix and let's check if $\mathbf{v}_k$ is an eigenvector. We need to compute the j-th element of the product $\mathbf{C}\mathbf{v}_k$. The j-th row of $\mathbf{C}$ is $(c_{j-0}, c_{j-1}, \dots)$ with indices taken modulo 4.
$(\mathbf{C}\mathbf{v}_k)_j = \sum_{l=0}^{3} C_{jl}(\mathbf{v}_k)_l = \sum_{l=0}^{3} c_{j-l \pmod 4}\omega^{lk}$
Let $m = j - l \pmod 4$, so $l = j - m \pmod 4$.
$(\mathbf{C}\mathbf{v}_k)_j = \sum_{m=0}^{3} c_m\omega^{(j-m)k} = \sum_{m=0}^{3} c_m\omega^{jk}\omega^{-mk}$
We can factor out ω^{jk} as it doesn't depend on the summation variable m:

$$(\mathbf{C}\mathbf{v}_k)_j = \omega^{jk}\left(\sum_{m=0}^{3} c_m\omega^{-mk}\right)$$

The term in the parenthesis is the eigenvalue λ_k. The term outside is the j-th component of the original vector, $(\mathbf{v}_k)_j$.

$$(\mathbf{C}\mathbf{v}_k)_j = \lambda_k(\mathbf{v}_k)_j$$

This holds for all j, so $\mathbf{C}\mathbf{v}_k = \lambda_k\mathbf{v}_k$. The proof is complete.

Example: 4x4 Fourier Matrix and Eigenvalues

For $n = 4$, the root of unity is $\omega = e^{2\pi i/4} = i$. The eigenvectors of any 4x4 circulant matrix are the columns of the 4x4 Fourier Matrix $\mathbf{F}_4$:

$$\mathbf{F}_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

Let $\mathbf{C}$ have first row $\mathbf{c} = (c_0, c_1, c_2, c_3)$. The four eigenvalues are:

- $\lambda_0 = c_0 + c_1 + c_2 + c_3$
- $\lambda_1 = c_0 - ic_1 - c_2 + ic_3$
- $\lambda_2 = c_0 - c_1 + c_2 - c_3$
- $\lambda_3 = c_0 + ic_1 - c_2 - ic_3$

These are the components of DFT($\mathbf{c}$).

The Convolution Rule I

The fact that all circulant matrices share the same eigenvectors (the Fourier basis) leads to a profound result.
Any circulant matrix $\mathbf{C}$ can be diagonalized by the Fourier matrix $\mathbf{F}$:

$$\mathbf{C} = \mathbf{F}^{-1}\Lambda\mathbf{F}$$

where Λ is a diagonal matrix whose entries are the eigenvalues of $\mathbf{C}$ (the DFT of its first row).

The Convolution Rule II

The Connection:

- A (cyclic) convolution of two vectors $\mathbf{c}$ and $\mathbf{d}$ can be written as the matrix-vector product $\mathbf{Cd}$.
- Using the diagonalization: $\mathbf{Cd} = (\mathbf{F}^{-1}\Lambda_C\mathbf{F})\mathbf{d}$.
- This means: to convolve $\mathbf{c}$ and $\mathbf{d}$, we can:
  1. Compute the DFT of $\mathbf{d}$: $\hat{\mathbf{d}} = \mathbf{Fd}$.
  2. Compute the DFT of $\mathbf{c}$ to get the eigenvalues: $\hat{\mathbf{c}} = \text{diag}(\Lambda_C)$.
  3. Multiply them element-wise: $\hat{\mathbf{r}} = \hat{\mathbf{c}} \odot \hat{\mathbf{d}}$.
  4. Compute the Inverse DFT of the result: $\mathbf{r} = \mathbf{F}^{-1}\hat{\mathbf{r}}$.

Convolution in the time/spatial domain is just element-wise multiplication in the frequency domain.

Example: Multiplying Circulant Matrices C and D I

Let's compare the number of operations for computing the product $\mathbf{E} = \mathbf{CD}$. Let the matrices be $n \times n$.

Approach 1: Direct Computation

- We only need to compute the first row of $\mathbf{E}$, which is the cyclic convolution of the first rows of $\mathbf{C}$ and $\mathbf{D}$.
- This requires a sum of n products for each of the n elements in the first row.
- Total operations: $\mathcal{O}(n^2)$.

Example: Multiplying Circulant Matrices C and D II

Approach 2: Using the Convolution Rule

- The eigenvalues of **E** are the element-wise product of the eigenvalues of **C** and **D**.
- **Step 1:** Find eigenvalues of **C** (DFT of its 1st row). Using the Fast Fourier Transform (FFT) algorithm, this takes $\mathcal{O}(n \log n)$.
- **Step 2:** Find eigenvalues of **D** (DFT of its 1st row): $\mathcal{O}(n \log n)$.
- **Step 3:** Multiply the eigenvalues element-wise: $\mathcal{O}(n)$.
- **Step 4:** Find the first row of **E** by taking the Inverse FFT of the resulting eigenvalues: $\mathcal{O}(n \log n)$.

Conclusion: For large n, the Fourier approach with its $\mathcal{O}(n \log n)$ complexity is dramatically faster than the direct $\mathcal{O}(n^2)$ convolution. This is why the FFT is one of the most important algorithms ever developed.

Convolution in 2D: Image Filtering

The concept extends naturally to 2D signals like images.

- An image is a matrix of pixel values (e.g., from 0 to 255).
- The filter, called a **kernel**, is a small matrix.
- The kernel slides over every position of the input image.
- At each position, we perform an **element-wise multiplication** between the kernel and the overlapping patch of the image.
- All the resulting products are **summed up** to get the value for a single output pixel.

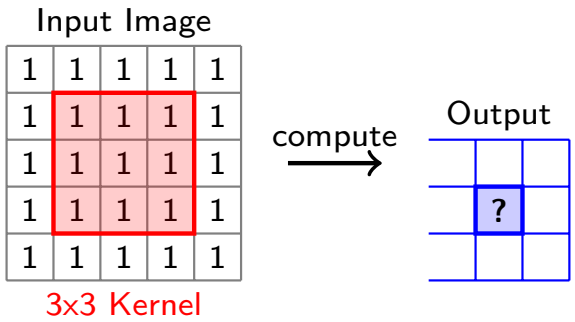

Figure: The 3x3 kernel slides over the input image matrix to produce a single pixel value in the output (feature map).

Example: The Blurring (Averaging) Filter I

Goal: Soften an image by averaging each pixel's value with its neighbors. This reduces sharp variations and noise.

The Kernel: A common blurring kernel is a 3x3 matrix where every element has the same value. The sum of the elements should be 1 to preserve brightness, so we normalize by dividing by 9.

$$\text{Blur Kernel} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Calculation for one pixel:

Image Patch $\quad * \quad$ **Kernel**

$$\begin{pmatrix} 10 & 10 & 10 \\ 10 & \mathbf{90} & 10 \\ 10 & 10 & 10 \end{pmatrix} \quad * \quad \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Example: The Blurring (Averaging) Filter II

Output pixel value $=$
$\frac{1}{9}(10 \cdot 1 + 10 \cdot 1 + \cdots + 10 \cdot 1 + 90 \cdot 1) = \frac{1}{9}(80 + 90) = \frac{170}{9} \approx 19.$
The high central value of 90 has been "smoothed out" to 19 by its neighbors.

Example: Edge Detection Filter I

Goal: Highlight sharp changes in intensity, which correspond to edges in the image.

The Kernel: An edge detection kernel is designed to have a strong response where pixel values change, and a zero response in uniform regions. The **Sobel operator** is a classic example.

$$\text{Vertical Edge Kernel } (G_x) = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

How it works: This kernel subtracts the pixel values on the left from the pixel values on the right, emphasizing vertical lines.

Calculation across an edge:

Image Patch

$$\begin{pmatrix} 10 & 10 & 90 \\ 10 & 10 & 90 \\ 10 & 10 & 90 \end{pmatrix}$$

Example: Edge Detection Filter II

Output value $= (-1 \cdot 10) + (0 \cdot 10) + (1 \cdot 90) + (-2 \cdot 10) + (0 \cdot 10) + (2 \cdot 90) + (-1 \cdot 10) + (0 \cdot 10) + (1 \cdot 90)$
$= -10 + 90 - 20 + 180 - 10 + 90 = 320$. A very high value, indicating a strong vertical edge.

- Link
- Link