

Machine learning for vision

Hiver 2015

Roland Memisevic

Lecture 1

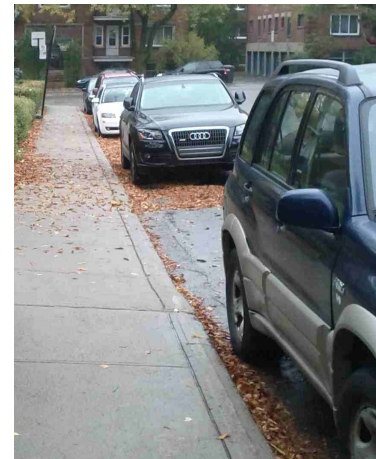


Ways to represent an image

- ▶ as a *matrix* (or *tensor* for multi-channel images such as color images)
 - ▶ as a *sequence* of gray-values or colors (“vectorization”, “serialization”)
 - ▶ as a *vector* equipped with operations from linear algebra (a point in “pixel space”)
 - ▶ as a *sequence of features* computed from the image
-
- ▶ It is common to jump back-and-forth between these representations.
 - ▶ For matrix representations, it can be convenient to use negative indexes, so that the origin is in the center of the image.



Translation invariance and locality



- ▶ Almost all structure in natural images is *position-invariant* and *local*. This has several practical consequences:
 - ▶ Almost all low-level vision operations will be based on small patches.
 - ▶ The universal mathematical framework for understanding the structure in images is the Fourier transform.



Linear features

- ▶ One of the simplest and most common operations on a vectorized image is linear projection:

$$\mathbf{w}^T \mathbf{x} = \sum_i w_i x_i$$

or, if images are matrices:

$$= \sum_{x_*=-\infty}^{\infty} \sum_{y_*=-\infty}^{\infty} W(x_*, y_*) I(x_*, y_*)$$

- ▶ The resulting scalar will be large if \mathbf{x} is similar to \mathbf{w}
- ▶ \mathbf{w} is called “feature”, “filter”, or “mask”
- ▶ Features can be defined by hand or learned from data.



Filtering

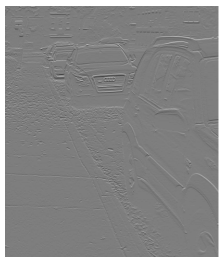
- ▶ We define “filtering” as scanning a linear feature W across the image:

$$O(x, y) = \sum_{x_*=-\infty}^{\infty} \sum_{y_*=-\infty}^{\infty} W(x_*, y_*) I(x + x_*, y + y_*)$$

which yields *feature map* where structure similar to W is enhanced.

- ▶ Almost every recognition system makes use of this.
- ▶ Feature maps may be multidimensional if we apply multiple filters on the same image.
- ▶ Usually the features are a much smaller than the image, and both can be thought of as padded with zeros on all sides.
- ▶ Of course, one can do the same for 1-d signals.

Example: two feature maps from edge filters



linear filter

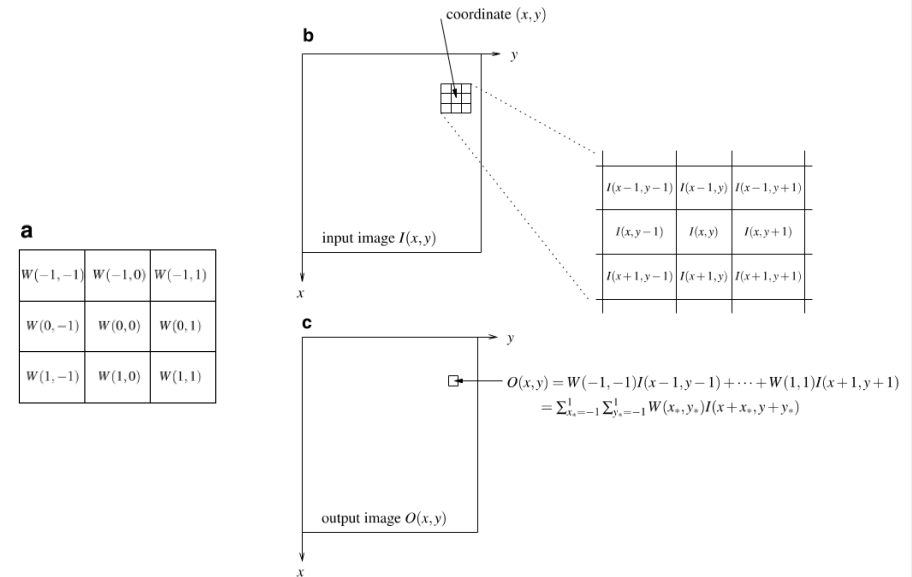


linear filter



(a non-linear combination)

Filtering



How to deal with the boundaries

- ▶ There are several things one can do at the boundaries:
 1. Make the output smaller than the original image
 2. Assume that the image is padded with zeros
 3. “wrap-around”: Think of image as being periodic
- ▶ All of these can be found in practice, depending on application, preferences, etc.

Filtering the impulse

- ▶ What happens if we filter the impulse image

$$\delta(x, y) = \begin{cases} 1, & \text{if } x = 0 \text{ and } y = 0, \\ 0, & \text{otherwise} \end{cases}$$

with some filter mask $W(x, y)$?



Filtering the impulse

- ▶ What happens if we filter the impulse image

$$\delta(x, y) = \begin{cases} 1, & \text{if } x = 0 \text{ and } y = 0, \\ 0, & \text{otherwise} \end{cases}$$

with some filter mask $W(x, y)$?

- ▶ The output image will show the *flipped feature in the center*:

$$O(x, y) = W(-x, -y)$$

- ▶ hmmm... can we get rid of the flipping?
- ▶ Yes, flip the image or the mask when computing the inner product...:



Convolution

- ▶ The **convolution** of two images is defined by

$$I_1(x, y) * I_2(x, y) = \sum_{x_*=-\infty}^{\infty} \sum_{y_*=-\infty}^{\infty} I_1(x-x_*, y-y_*) I_2(x_*, y_*)$$

- ▶ Typically (not always), one image will be a feature vector $h(x, y)$:

$$I(x, y) * h(x, y) = \sum_{x_*=-\infty}^{\infty} \sum_{y_*=-\infty}^{\infty} I(x-x_*, y-y_*) h(x_*, y_*)$$

- ▶ In 1-d:

$$s(y) * h(y) = \sum_{y_*=-\infty}^{\infty} s(y-y_*) h(y_*)$$



Properties of convolution

- ▶ commutative:

$$s(x, y) * h(x, y) = h(x, y) * s(x, y)$$

- ▶ associative:

$$(s(x, y) * h_1(x, y)) * h_2(x, y) = s(x, y) * (h_1(x, y) * h_2(x, y))$$

- ▶ distributive:

$$s(x, y) * (h_1(x, y) + h_2(x, y)) = s(x, y) * h_1(x, y) + s(x, y) * h_2(x, y)$$

- ▶ These are properties of *products*.



Convolution as superposition, impulse response

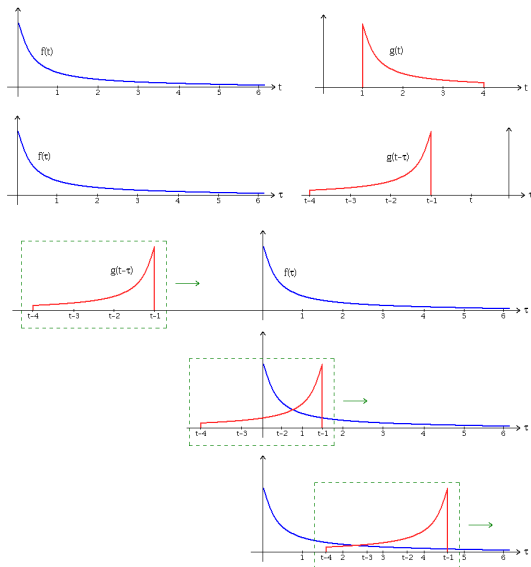
- ▶ One usually thinks of convolution as scanning a (flipped) filter across the image.
- ▶ One can also think of convolution as computing a weighted sum over *translated copies of the image*.
- ▶ The (flipped) filter entries are then the weights in the combination.
- ▶ The **impulse response** of a filter is defined as the result of convolving the filter with the impulse image.

◀ ▶ ⏪ ⏩ 🔍

Roland Memisevic

Machine learning for vision

One-d convolution according to Wikipedia



◀ ▶ ⏪ ⏩ 🔍

Roland Memisevic

Machine learning for vision

Continuous convolution

- ▶ In continuous domains, replace the sum with an integral:

$$s(x, y) * h(x, y) = \iint_{-\infty}^{\infty} s(x^*, y^*) \cdot h(x - x^*, y - y^*) dx^* dy^*$$

- ▶ *Caveat:* To get a sensible impulse response, the impulse signal needs to integrate to one. But at the same time it has to be equal to zero everywhere except at the origin.
- ▶ Solution: The Dirac delta.

◀ ▶ ⏪ ⏩ 🔍

Roland Memisevic

Machine learning for vision

Finite convolution

- ▶ In practice, almost all filters that we need to deal with are finite.
- ▶ So we may write

$$(I * h)_{m,n} = \sum_{k=-\frac{K}{2}}^{\frac{K}{2}} \sum_{r=-\frac{R}{2}}^{\frac{R}{2}} h(k, r) I(m - k, n - r)$$

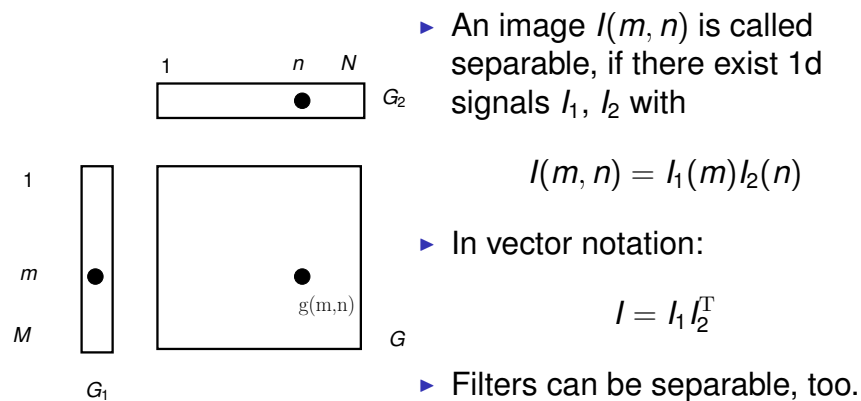
- ▶ For images of size $M \times N$ this takes $MNKR$ operations.
- ▶ K and R are usually much smaller than M and N .

◀ ▶ ⏪ ⏩ 🔍

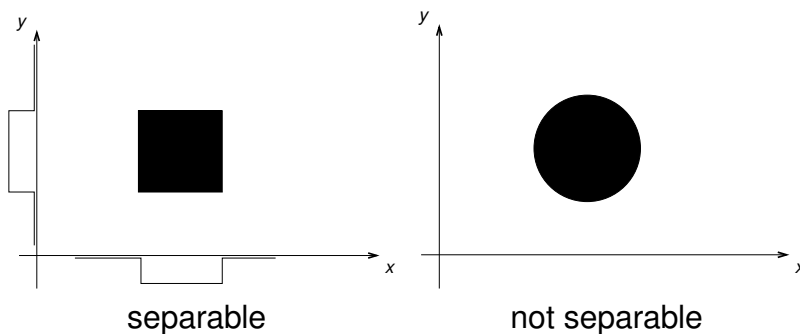
Roland Memisevic

Machine learning for vision

Separability



Separable/non separable example



Convolution and separability

- ▶ If the filter h is separable, we have:

$$\begin{aligned} (I * h)_{m,n} &= \sum_k \sum_r h(k, r) s(m - k, n - r) \\ &= \sum_k \sum_r h_1(k) h_2(r) s(m - k, n - r) \\ &= \sum_k h_1(k) \underbrace{\sum_r h_2(r) s(m - k, n - r)}_{=: a(m-k, n)} \end{aligned}$$

- ▶ We can pre-compute $a(\cdot, \cdot)$ using MNR operations.
- ▶ Given $a(\cdot, \cdot)$ the result takes only MNK operations.
- ▶ This makes $MN(R + K)$ in total.

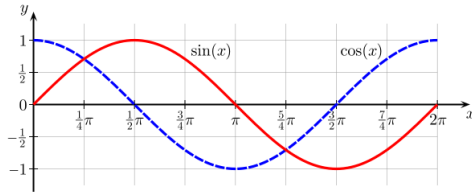


LTI Systems

- ▶ Convolution amounts to applying a single linear transformation everywhere. This makes it a **Linear Time-Invariant system (LTI System)**
- ▶ (note: “time”, for us, usually means space)
- ▶ LTI system is aka *Linear Shift-Invariant system (LSI System)*
- ▶ A system is a function that takes a signal as input and outputs a new signal.
- ▶ There are many dimensions along which one can characterize systems: causality, finite vs. infinite impulse response (FIR vs. IIR filter), stability (eg. “BIBO”), ...



Sine and cosine



- ▶ Two signals which (if combined into one) are intimately related to LTI systems are the sine and the cosine-signal:

$$s_c(x; A, \omega, \varphi) = A \cos(\omega x + \varphi)$$

$$s_s(x; A, \omega, \varphi) = A \sin(\omega x + \varphi)$$

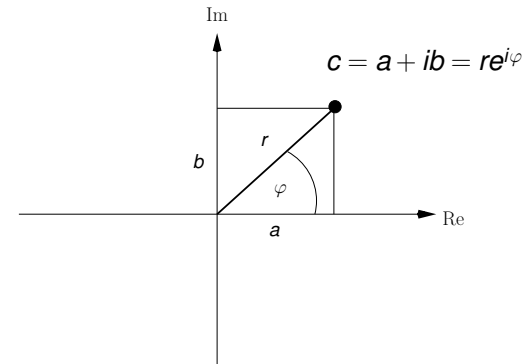
- ▶ ω is called (“angular”) frequency; its reciprocal is called wavelength; φ is called phase.

Digression: Complex numbers

- ▶ Complex numbers are “2d-vectors” with some special arithmetic, most of which related to *Euler’s formula*:

$$e^{i\varphi} = \cos \varphi + i \sin \varphi$$

- ▶ Most applications rely on jumping back-and-forth between cartesian and polar coordinates:



$$a = r \cos(\varphi)$$

$$b = r \sin(\varphi)$$

$$r = |c| = \sqrt{a^2 + b^2}$$

$$\varphi = \arg(c) = \text{atan}\left(\frac{b}{a}\right)$$

Digression: Complex numbers

- ▶ Addition is the same as for 2d vectors.
- ▶ Multiplication is standard arithmetic in the polar representation:

$$c_1 \cdot c_2 = r_1 e^{i(\varphi_1)} \cdot r_2 e^{i(\varphi_2)} = r_1 \cdot r_2 \cdot e^{i(\varphi_1 + \varphi_2)}$$

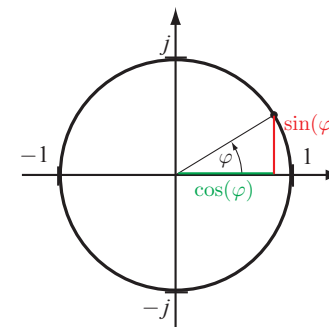
Thus, multiplication is *stretching* + *rotation*.

- ▶ Multiplying a number by a complex number c of length 1.0,

$$c = e^{i\alpha},$$

amounts to rotating the number by α degrees counter clock-wise around the origin.

Digression: Complex numbers



- ▶ The signal $\exp(i\omega t)$ is therefore a unit-vector running counter clock-wise around the unit circle.

Digression: Complex numbers

- ▶ Other useful equations:
 - ▶ Conjugation is reflection at the real axis:

$$\bar{c} = a - ib = r \exp(-i\varphi)$$

- ▶ It follows that $\bar{c}c = |c|^2$ and $\frac{1}{2}(\bar{c} + c) = \text{real}(c)$
- ▶ The standard inner product uses conjugation:

$$\langle \vec{c}, \vec{d} \rangle = \sum_i \bar{c}_i d_i$$

- ▶ Why? Because now $\langle \vec{c}, \vec{c} \rangle = \|\vec{c}\|^2$
- ▶ In practice, use the function `atan2()` to compute the `atan` for polar representations.



The phasor

The **phasor** is the complex valued signal

$$p(t) = \exp(i\omega t)$$

- ▶ The phasor is the central object of interest in signal processing. It combines sine and cosine into a single, complex valued signal.
- ▶ t can be continuous or discrete.



Digression: Complex numbers

$$e^{i\pi} + 1 = 0$$

— End of digression —



Sine and cosine from two phasors

- ▶ Euler's formula allows us to express sine and cosine with phasors:

$$\cos \varphi = \frac{1}{2}e^{i\varphi} + \frac{1}{2}e^{-i\varphi}$$

$$\sin \varphi = \frac{1}{2i}e^{i\varphi} - \frac{1}{2i}e^{-i\varphi} = -\frac{i}{2}e^{i\varphi} + \frac{i}{2}e^{-i\varphi}$$

- ▶ To derive these formulas one can use:

$$e^{i\varphi} + e^{-i\varphi} = 2 \cos \varphi$$

and

$$e^{i\varphi} - e^{-i\varphi} = 2i \sin \varphi$$

- ▶ One can think of this as a way to transform to Cartesian coordinates.



Phasors and shift (1d)

Phasors are eigenfunctions of translation

$$p(t - z) = e^{i\omega(t-z)} = e^{i\omega t} e^{-i\omega z} = e^{-i\omega z} p(t)$$

- ▶ Thus, translating a (1-d) phasor is equal to multiplying it by the constant: $e^{-i\omega z}$
- ▶ This constant depends on (i) the frequency of the phasor, and (ii) the amount of shift.
- ▶ Now recall that we can think of convolution as computing a linear combination of translated copies of the signal:



Phasors and convolution (1d)

- ▶ The complex number $H(\omega)$ is called *frequency response* of the filter.
- ▶ Its absolute value $|H(\omega)|$ is called *amplitude response*.
- ▶ Its phase $\arg H(\omega)$ is called *phase response*.



Phasors and convolution (1d)

Phasors are eigenfunctions of convolution

$$\begin{aligned} p * h &= \sum_{z=-\infty}^{\infty} h(z) p(t - z) \\ &= \left(\sum_{z=-\infty}^{\infty} h(z) e^{-i\omega z} \right) e^{i\omega t} \\ &=: H(\omega) p \end{aligned}$$

- ▶ The constant depends on the frequency of the phasor.



Phasors and convolution (1d)

- ▶ Recall that multiplication for complex numbers is stretching + rotation.
- ▶ It follows that convolution can change *phase* and *amplitude* of a phasor but not its frequency.
- ▶ Convolution with the filter $h(t)$ turns the phasor $p(t) = \exp(i\omega t)$ into:

$$H(\omega) \exp(i\omega t) = |H(\omega)| \exp(i\omega t + \arg H(\omega))$$



Signals as superpositions of phasors (1d)

- ▶ The fact that phasors are well-behaved wrt. convolution suggests writing a signal as a *superposition of phasors*.
- ▶ That way we can characterize the effect of a filter by just looking at the phasors (which differ by frequency).
- ▶ For signals that are *finite* (in other words, usual vectors) this is straightforward, because we can construct an orthonormal basis from phasors as follows:



An orthogonal basis from phasors (1d)

- ▶ Define T discrete phasor signals of length T with frequencies $k = 0, \dots, T - 1$ as:

$$p_k(t) = e^{\frac{2\pi i}{T}kt} \quad t = 0, \dots, T - 1$$

- ▶ These phasors form an orthonormal basis wrt. the complex inner product:

$$\langle p_k(t), p_l(t) \rangle = \sum_t e^{\frac{2\pi i}{T}kt} e^{-\frac{2\pi i}{T}lt} = T\delta_{kl}$$

- ▶ So the coefficients that allow us to represent a signal $s(t)$ in this basis are simply

$$S(k) = \frac{1}{T} \langle s(t), p_k(t) \rangle$$



Discrete Fourier Transform (1d)

Discrete Fourier Transform (DFT) 1d

$$S(k) = \sum_{t=0}^{T-1} s(t) e^{-i\frac{2\pi}{T}kt} \quad k = 0, \dots, T - 1$$

Inverse discrete Fourier Transform 1d

$$s(t) = \frac{1}{T} \sum_{k=0}^{T-1} S(k) e^{i\frac{2\pi}{T}tk} \quad t = 0, \dots, T - 1$$



Discrete Fourier Transform (1d)

- ▶ Most practical implementations of the DFT use the Fast Fourier Transform (FFT) to compute the coefficients. Accordingly, most libraries are named “FFT” not “DFT”.
- ▶ One can generalize the DFT to continuous periodic functions (“Fourier Series”).
- ▶ One can generalize further to (non-periodic) square-integrable functions (“Fourier Transform”).



Spectrum

- ▶ The complex function $S(\omega)$ is called the *spectrum* of the signal.
 - ▶ $|S(\omega)|$ is called *amplitude spectrum*.
 - ▶ $\arg S(\omega)$ is called *phase spectrum*.
-
- ▶ It follows that the frequency response of a filter is the spectrum of the impulse response.
 - ▶ The spectrum $S(\omega)$ is a set of coefficients that we need to apply to the phasors of corresponding frequency to get back the signal.
 - ▶ It determines the strength of each phasor (via $|S(\omega)|$) and the phase of each phasor (via $\arg S(\omega)$).

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍

Discrete = finite = periodic

- ▶ For discrete time signals, we have

$$\exp(i\omega t) = \exp(i(\omega + 2\pi)t)$$

and thus

$$S(\omega + 2\pi) = S(\omega)$$

- ▶ **Discrete signals have a periodic spectrum.**
- ▶ We need to know $S(\omega)$ only for $-\pi \leq \omega \leq \pi$.
- ▶ The converse is true, too: periodic signals have discrete spectrum.
- ▶ In practice, it is common to think of *periodic* as *finite*.

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍

Fourier analysis zoo

| | signal continuous | signal discrete |
|---------------------|---|---|
| spectrum continuous | Fourier transform | Discrete Time Fourier Transform (DTFT): spectrum is periodic |
| spectrum discrete | Fourier series: signal is periodic | Discrete Fourier Transform (DFT/FFT): spectrum and signal are both periodic/ discrete / finite |

“finite” == “periodic”

◀ ▶ ⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍