Intelligent Systems for Pattern Recognition

Federico Matteoni

A.A. 2021/22

Index

0.1 Introduction

Prof.s: Davide Bacciu and Antonio Carta

Objectives Train ML specialists capable of: designing novel learning models, developing pattern recognition applications using ML, developing intelligent agents using **Reinforcement Learning**.
We're referring to images and signals, but not limited to that: practical applications.
Focusing on challenging and complex data: **machine vision** (noisy, hard to interpret, semantically rich...) and **structured data** (relational information: sequences, trees, graphs...)
Natural Language Processing will be used as an example, but will not be the focus of this course.

Methodology-Oriented Outcomes Gain in-depth knowledge of advanced machine learning models, understanding the underlying theory. This gives the ability to read and understand and discuss research works in the field.

Application-Oriented Outcomes Learn to address modern pattern recognition applications, gain knowledge of ML, PR and RL libraries and be able to develop an application using ML and RL models.

Prerequisites Knowledge of ML fundamentals, mathematical tools for ML and Python.

0.2 Pattern Recognition

Automated recognition of meaningful patterns in noisy data.

Origins

Viola-Jones Algorithm Framework for face recognition. Sum pixel in white area and subtract those in the black portion. The VJ algorithm positions the masks on the image and combines the responses (training set of $\simeq$5k images with hand-aligned filters)

An historical view

1. Identification of distinguishing features of the object/entity (**feature detection**)
2. Extraction of features for the defining attributes (**feature extraction**)
3. Comparison with known patterns (**matching**)

Basically, lots of time spent hand-engineering the best data features.

A modern view Data is thrown into a neural network. A single stage process with a data crushing-and-munching neural network spitting out prediction, which encapsulates the three historical steps. But the time is now spent in fine-tuning the neural network.

The deep learning Lego Creating applications by putting together various combinations of CNN and LSTM modules.

0.2.1 Signals

Signals are time series: a sequence of measurements in time. Examples of sources are: medicine, finance, geology, IoT, biometrics...

Formalization A time series x is a sequence of measurements in time t

$$\text{x} = x_0, \dots, x_N$$

where x_t or $x(t)$ is the measurement at time t.

Observation can be at **irregular** time intervals.

We assume **weakly stationary** (or second-order stationary) data

$\forall\, t \;\; E[x_t] = \mu$
$\forall\, t \; \text{Cov}(x_{t+\tau}, x_t) = \gamma_\tau$ with γ depending only on the lag τ

Goals

Description

Analysis: identify and describe dependencies in data

Prediction: forecast next values given information up to t

Control: adjust parameters of the generative process to make the time series fit a target

Key Methods

Time domain analysis: assesses how a signal changes over time (correlation, convolution, auto-regressive models)

Spectral domain analysis: assesses the distribution of the signal over a range of frequencies (Fourier analysis, wavelets)

Time Domain Analysis

Mean

$$\hat{\mu} = \frac{1}{N}\sum_{t=1}^{N} x_t$$

Can be used to subtract mean from values and "standardize" the two series.

Autocovariance For lag $-N \leq \tau \leq N$

$$\hat{\gamma}_x(\tau) = \frac{1}{N}\sum_{t=1}^{N-|\tau|}(x_{t+|\tau|} - \hat{\mu})(x_t - \hat{\mu})$$

Autocorrelation The correlation of a signal with itself.

$$\hat{\rho}_x(\tau) = \frac{\hat{\gamma}_x(\tau)}{\hat{\gamma}_x(0)}$$

We can compute this with every possible τ, finding the max/min which gives the τ where the autocorrelation is max/min, which means the lag where the signal starts repeating itself. The lags near zero typically dominates, so we want the maximum lag reasonably far from 0.

Autocorrelation plot

Cross-Correlation A measure of similarity of x_1 and x_2 as a function of a time lag τ

$$\phi_{\mathrm{x}_1\,\mathrm{x}_2}(\tau) = \sum_{t=\max\{0,\tau\}}^{\min\{(T_1-1+\tau),(T_2-1)\}} \mathrm{x}_1(t-\tau)\cdot \mathrm{x}_2(t)$$

Normalized cross-correlation Returns an amplitude independent value

$$\overline{\phi}_{\mathrm{x}_1\,\mathrm{x}:2}(\tau) = \frac{\phi_{\mathrm{x}_1\,\mathrm{x}:2}}{\texttt{todo}} \in [-1,+1]$$

With $\overline{\phi}_{\mathrm{x}_1\,\mathrm{x}:2}(\tau) = +1$ mean that the two time series have the exact same shape if aligned at time τ. Nearing -1 we get the maximum anticorrelation, same shape but opposite sign. Near 0 we get that the two signals are completely **linearly** uncorrelated.
Note that we measure **linear correlation**.

Cross correlation looks like the convolution

$$(f*g)[n] = \sum_{t=-M}^{M} f(n-t)g(t)$$

but we have a flipped sign ($n-t$ instead of $t-\tau$).
Cross-correlation is not symmetric, whereas convolution is ($f*g = g*f$).

Autoregressive Process A timeseries autoregressive process (AR) of order K is the linear system

$$x_t = \sum_{k=1}^{K} \alpha_k x_{t-k} + \epsilon_t$$

Autoregressive means x_t regresses on itself

$\alpha_k \Rightarrow$ linear coefficients $| \, |\alpha| < 1$

$\epsilon_t \Rightarrow$ sequence of independent and identically distributed values with mean 0 and fixed variance.

We look backward K steps, so limited memory.

ARMA Autoregressive with Moving Average process

$$x_t = \sum_{k=1}^{K} \alpha_k x_{t-k} + \sum_{q=1}^{Q} \beta_q \epsilon_{t-1} + \epsilon_t$$

With ϵ_t Random white noise (again)

The current time series values is the result of a regression on its past values plus a term that depends on a combination of stochastically uncorrelated information

Estimating Autoregressive Models Need to estimate: the values of the linear coefficients α_t and β_t and the order of the autoregressor K and Q
Estimation of the α, β is performed with the Levinson-Durbin Recursion (`levinson(x, K)` in matlab, and included in several Python modules).
The order is often estimated with a Bayesian model selection criterion, choosing the largest K and Q possible. E.g.: BIC, AIC...
The set of autoregressive parameters $\alpha_{i,1}, \ldots, \alpha_{i,K}$ fitted to a specific time series x_i is used to confront it with other time series. Same thing for β so we can use α for both sets.

Comparing time series by AR

timeseries clustering

novelty/anomaly detection

Spectral Domain Analysis

Analyze the time series in the frequency domain. Key idea: decomposing the time series into a linear combination of sines and cosines with random and uncorrelated coefficients. So a **regression on sinusoids** with Fourier analysis.

Fourier Transform Discrete Fourier Transform (DFT): transform a time series from the time domain to the frequency domain. Can be easily inverted back to the time domain.
Useful to handle periodicity in the time series: seasonal trends, cyclic processes...

Representing functions We know that, given an orthonormal system for E we can use linear combinations of the basis $\{e_1, \ldots\}$ to represent any function $f \in E$

$$\sum_{k=1}^{\infty} \langle f, e_k \rangle e_k$$

Given the orthonormal system

$$\{\frac{1}{\sqrt{2}}, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots\}$$

The linear combination above becomes the Fourier series `todo`

Representing function in Complex space Using $\cos(kx) - i\ \sin(kx) = e^{-ikx}$ with $i = \sqrt{-1}$ we can rewrite the Fourier series as

$$\sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

on the orthonormal system

$$\{1, e^{ix}, e^{-ix}, e^{2ix}, e^{-2ix}, \ldots\}$$

Representing Discrete Time Series Consider x of length N and $x_n \in R$. Using the exponential formulation, the orthonormal system is finite, from e_0 to e_{N-1} each $\in C^N$
The n-th component of the k-th vector is

$$[e_k]n = e^{\frac{-2\pi ink}{2}}$$

Discrete Fourier Transform Given a time series x = $x_0, \ldots, x_{N-1}$ its DFT is the sequence

$$\text{Spectral domain} \;\; X_k = \sum_{n=1}^{N-1} x_n e^{\frac{-2\pi ink}{N}} \;\; \text{Time domain}$$

And can be inverted

$$x_k = \frac{1}{N} \sum_{k=1}^{N-1} X_k e^{\frac{2\pi ink}{N}}$$

Basic Spectral Quantities in SFT

Amplitude $A_k = |X_k| = \sqrt{Re^2(X_k) + Im^2(X_k)}$

Power $P_k = \frac{|X_k|^2}{N}$, more used in reality and under some conditions this is a reasonable estimate of

DFT in Action We use the DFT elements $X_1, \ldots, X_K$ as representation of the signal to train the predictor/classifier. This representation can reveal patterns that are not clear in the time domain.

0.2.2 Image Processing

Bidimentional series. Basically same approach to signals.

Descriptors

An image is a matrix of pixel intensities or color values (RGB). There are other representations, not interesting for this course. CIE-LUV often used in image processing due to perceptual linearity (image difference is more coherent)

Machine Vision Applications For example region of interest, or object classification.
Even pixel-level tasks, for example image segmentation (regions of the image) or semantic segmentation (classifying regions of the image).
Up one level of abstraction: automated image captioning, requiring identifying objects, generating sentences and ranking those sentences.

Key Questions

How to represent visual information? It has to be:

Informative, carrying all the information

Invariant to photometric (different illuminations) and geometric transformation (position in the picture, rotation...)

Efficient for indexing and querying

How to identify informative parts?

Whole image is generally not a good idea

Must lead to good representations

Image Histograms One of the first answer. Describes the distribution of some visual information on the whole image: colors, edges, corners. . . depending on the goals.

> **Color Histograms**, one of the earliest image descriptors.
> Count the number of pixels of a given color (normalize!). We need to discretize and group the RGB colors.
> Any information concerning shapes and position is lost. Two image with a random permutation of the same pixels produce the same color histograms.
> Images can be compared, indexed and classified based on their color histogram representation.
> Can be computed with OpenCV in Python.

Describing Local Image Properties We need something less global, on a local level. Capturing information on image regions, extract **multiple local descriptors**: different location, different scale. . .
Several approaches, typically performing convolution between a filter and the image region. Using filters sensitive to specific features we can extract many kind of information.

Localized Descriptors

> **Intensity Vector** The simplest form of localized descriptor: a vector $n \cdot m$ of the pixels of a single patch of the image with dimensions n, m. The vector can be normalized to make it invariant to intensity variations.
> But rotating gives a different vector. A more robust representation is an histogram of this vector.

> **Distribution-Based Descriptors** Represent local patches by histograms describing properties of the pixels in the patch. The simplest is an histogram of intensity values, but it's not invariant enough even if normalized.
> We want a descriptor invariant to illumination (normalization), scale (captured at multiple scale) and geometric transformations (rotation invariant). We want locality, histogram based and invariant to geometric transformation.

SIFT Scale Invariant Feature Transform

1. Center the image patch on a pixel x, y of the image I
2. Represent image at scale σ (controls how close to look at the image)

   Convolve the image with a Gaussian filter with standard variation σ, basically computing average of pixels with the coefficient taken from a Gaussian distribution. With a smooth Gaussian, we artificially smooth the object, and vice versa. We can compute different versions of the image.

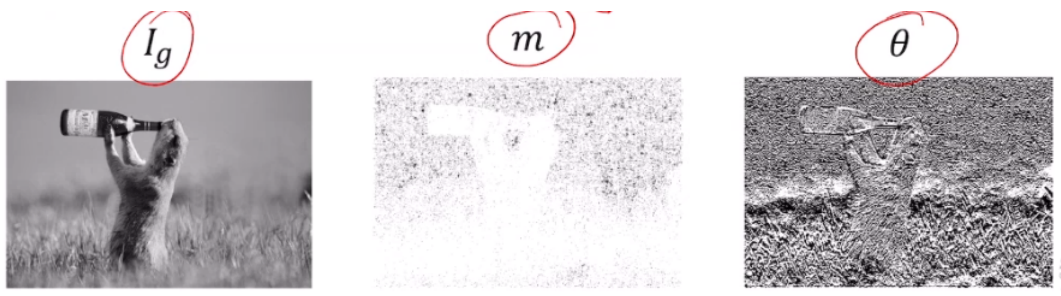
   $$L_\sigma(x, y) = G(x, y, \sigma) * I(x, y)$$

   $$G(x, y, \sigma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

3. Compute the **gradient of intensity** in the patch, extracting magnitude m and orientation Θ using finite differences.

Gaussian Filter of an Image

```
Iscale = imgaussfilt (I, sigma);
```



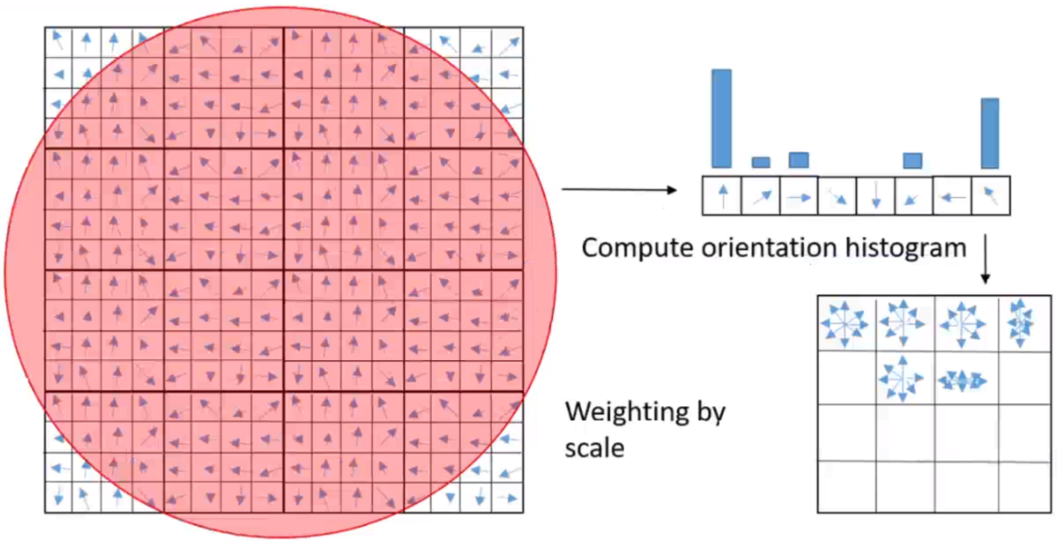
4. Create **gradient histogram**

   4×4 gradient window

   Histogram of 4×4 per window on 8 orientation bins

   Gaussian weighting on center keypoint (width = 1.5σ)

   4×4×8 = 128 descriptor size

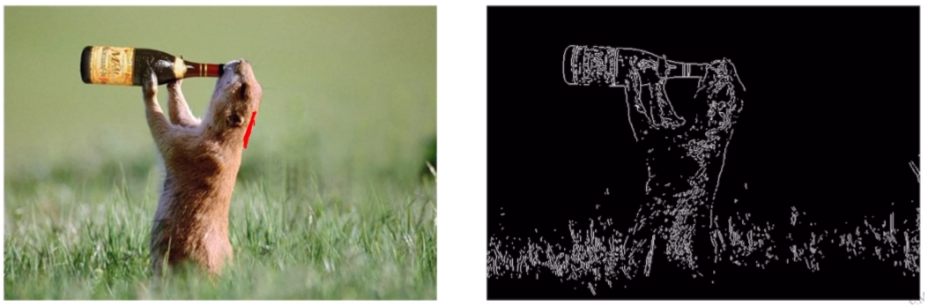


Detectors

Visual Feature Detector Properties

Repeatability Detect the same feature in different image portions and different images, under different conditions (color, luminance...). So with respect to translation, photometric changes, rotation, scaling and affine transformations (non-isotropic changes, for example the relative position of the camera)...

Edge Detection We need to find interesting points, talking about fundamental elements, basic components. One possible example are the edges of the image.
Reasoning in changes of intensity: edges are those points where the intensity changes.



Typically using an edge detector filter on each pixel and turning pixels white or black by thresholding

Edges and Gradients The image gradient (graylevel) is

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

which is basically two images, gradient in both x and y directions. Edge are pixel regions where intensity gradient changes abruptly. The return of finite difference methods:

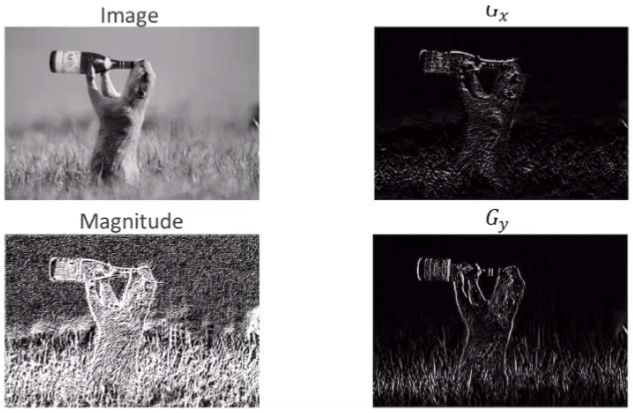
$G_x = \frac{\partial I}{\partial x} \simeq I(x+1, y) - I(x-1, y)$

$G_y = \frac{\partial I}{\partial y} \simeq I(x, y+1) - I(x, y-1)$

Edge detectors build on this idea combining with some smoothing: average on multiple pixels.

Prewitt operators

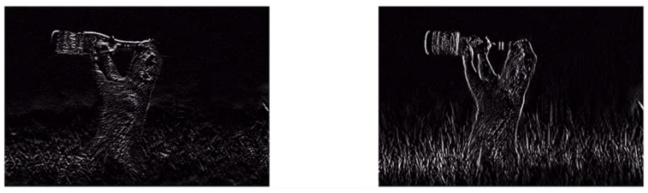
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$



Sobel Operator

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Often with a constant $c \simeq \frac{1}{8}$ for scaling.



Blob Detection Pixel regions with little gradient variability.
$g_\sigma(x, y)$ has maximum response when centered on a circle of radius $\sqrt{2}\sigma$, with σ being the scale of the gaussian.
Laplace of Gaussian (LoG):

$$\nabla^2 g_\sigma(x, y) = \frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2}$$

Typically using a scale normalized response

$$\nabla^2_{norm} g_\sigma(x, y) = \sigma^2 \left(\frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2} \right)$$

1. Convolve image with a LoG filter at different scales $\sigma = k\sigma_0$ by varying k with a starting σ_0
2. Find maxima of squared LoG responses:

    Find maxima on space-scale: focus on a scale and find maxima

    Find maxima between scales: do the same for all the scales and pick the maxima

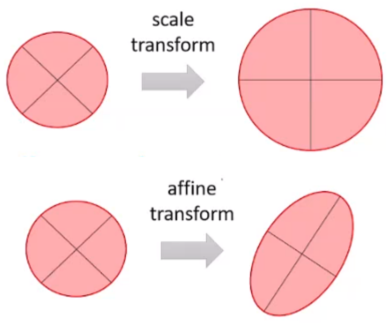
    Threshold

The LoG can be approximated by the Difference of Gaussians (DoG) for efficiency, so to reuse part of the computations.

$$g_{k\sigma_0}(x, y) - g_{\sigma_0}(x, y) \simeq (k-1)\sigma_0^2 \nabla^2 g_{(k-1)\sigma_0}$$

SIFT uses LoG.

Affine Detectors Laplacia-based detectors are invariant to scale thanks to the maximization in scale-space. Still not invariant to affine-transformation.

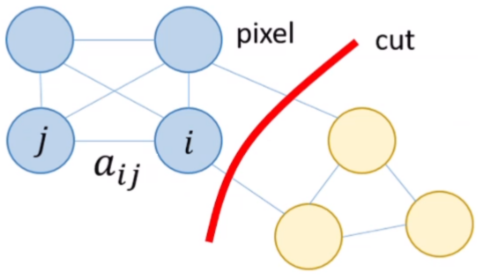


MSER Maximally Stable Extremal Regions
Extract covariant regions (blobs) that are stable connected components of intensity sets of the image. Interesting areas stay the same at different threshold: stable with respect to variations in luminance, not scale dependent and doesn't assume circular regions. Key idea... Blobs generated (locally) by binarizing the image over a large number of thresholds

Intuitions on MSER Generate frames from the image by thresholding it on all graylevels.
Capture those regions that from a small seed of pixel grow to a stably connected region. Stability is assessed by looking at derivatives of region masks in time (most stable $\Rightarrow$ minima of connected region variation).

Image Segmentation The process of partitioning an image into a set of homogeneous pixels, hoping to match objects or their subparts.
A naive approach: straighten the image in a $N \cdot M$ vector and use it as a dataset for K-means.

Ncut Normalized cuts



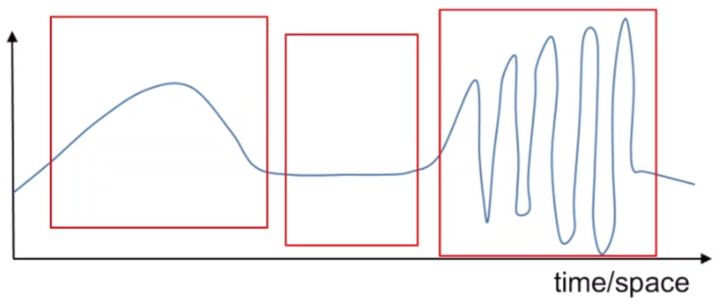
With each node being a pixel: an image is a graph. a_{ij} is the affinity between pixels at a certain scale σ. A cut of G is the set of edges such whose removal makes G a disconnected graph. Breaking the graph into pieces by cutting edges of low affinity.
The normalized cut problem is NP-hard, approximate solution as an eigenvalue problem. But the eigenvalue decomposition it's really intractable with big images. We need to reduce the number of pixels. We can use **superpixels**: clustering the pixels with K-means (perhaps with different K) and using the clusters as nodes for segmentation algorithms (Ncut, Markov Random Fields...). We can do multiscale superpixeling and segmenting at different scales, different policies...

Conclusion

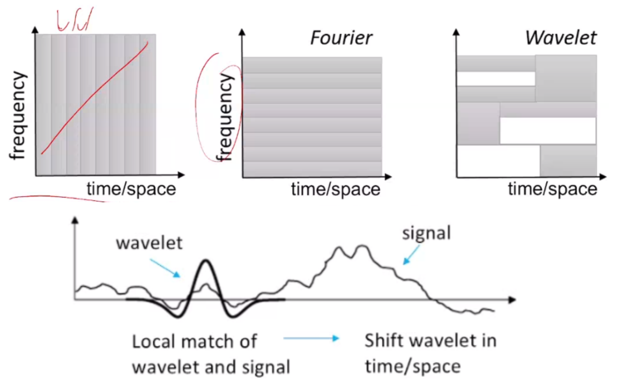
Image processing is a lot about convolutions: linear masks to perform gradient operations, gaussian functions to apply scale changes (zooming in and out). Computational efficiency is a driving factor: convolution in Fourier domain, superpixel, lightweight feature detectors...

0.2.3 Wavelets

Limitations of DFT Sometimes we might need localized frequencies rather than global frequency analysis.



We slice the signal in "time slots" in time analysis and "frequency slots" in frequency analysis. In wavelet analysis you do both.



1. Scale and shift original signal
2. Compare signal to a wavelet
3. Compute coefficient of similarity

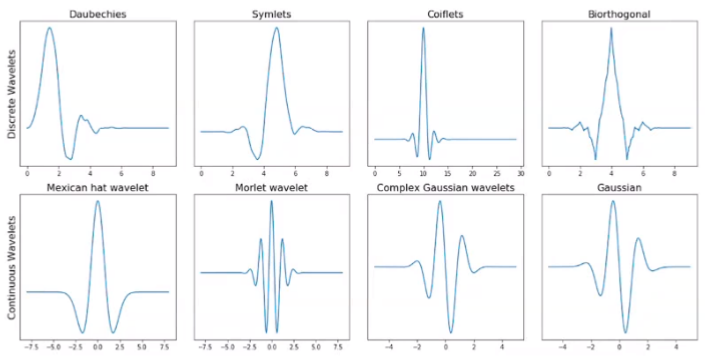
Split the signal with an orthonormal basis generate by translation and dilation of a mother wavelet

$$\sum_t \mathrm{x}(t)\phi_{j,k}(t)$$

Terms k, j regulate scaling and shifting of the wavelet

$$\phi_{t,k}(\mathrm{x}) = 2^{\frac{k}{2}}\phi\left(\frac{t - j2^k}{2^k}\right)$$

With many different options for the mother wavelet ϕ



Scaling and dilation is akin to a sort of frequency: high scale mean stretched wavelet with slowly changing coarse feature and low frequency, while low scale compressed wavelet with rapidly changing details and high frequency.

DWT Discrete Wavelet Transform: uses a finite set of scales and shifts rather than ”any possible value” as in the continuous wavelet transform.

0.3 Generative and Graphical Models

Generative referring to the probability we learn: if we know the distribution probability of data we can generate new data.
Graphical referring to graphical formalisms that describe in a syntetic way the structures we’ll see.

Generative Learning ML models that represent knowledge inferred from data under the form of probabilities:

Probabilities can be sample: new data can be generated

Supervised, unsupervised, weakly supervised tasks

More easily incorporate prior knowledge on data and tasks

Interpretable knowledge (how data is generated)

The majority of modern tasks comprises large number of variables

Modeling the joint distribution of all variables can become impractical

Exponential size of the parameter space

Computationally impractical to train and predict

Representation Graphical models are a compact way to represent exponentially large probability distributions. Encode conditional independence assumptions, and different classes of graph structures imply different assumptions/-capabilities.

Inference How to query (predict with) a graphical model? Probability of unknown X given observations d, $P(X \mid d)$, the **most likely hypothesis** (parameters) X.

Learning Find the right model parameters.

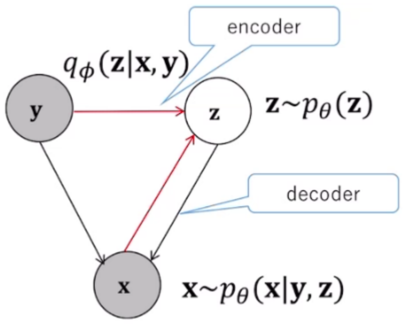
Representation A graph whose nodes are random variables and edges represent probabilistic relationships between the variables.
Different classes of graphs:

Directed edges express **causal relationships**

Undirected edges express **soft constraints**, values cannot change independently

Dynamic models, graphs subject to structure changes to reflect dynamic processes. For example RNNs: recurrent neural networks are unfolded using weight sharing, producing a dynamic model.

In Deep Learning Bayesian learning necessary to understand Variational Deep Learning.


encoder
$q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$
y
z
$\mathbf{z} \sim p_\theta(\mathbf{z})$
decoder
x
$\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z})$

Generate new knowledge Complex data can be generated if the model is powerful enough to capture its distribution.

0.3.1 Probability Refresher

`todo`

Inference

Bayesian: consider all hypothesis weighted by their probabilities

$$P(X \mid d) = \sum_i P(X \mid h_i) P(h_i \mid d)$$

MAP (Maximum a-Posteriori): infer X from $P(X \mid h_{MAP})$ where h_{MAP} is the maximum a-posteriori hypothesys given d

$$h_{MAP} = \arg\max_{h \in H} P(h \mid d) = \arg\max_{h \in H} P(d \mid h) P(h)$$

ML assuming uniform prioris $P(h_i) = P(h_j)$ yields the maximum likelihood (ML) estimate $P(X \mid h_{ML})$

$$h_{ML} = \arg\max_{h \in H} P(d \mid h)$$

Any probability can be obtained from the Joint Probability Distribution $P(X_1, \ldots, X_n)$ by marginalization but at an exponential cost (e.g. 2^{n-1} for a marginal distribution from binary RV)

0.3.2 Graphical Models

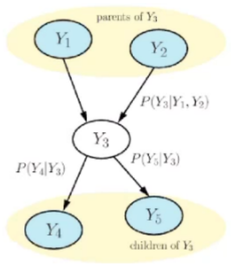
Compact graphical representation for exponentially large joint distribution: simplifies marginalization and inference algorithms

Bayesian Networks

Directed Acyclic Graphs (DAG) $G = (V, E)$

Nodes $v \in V$ represent random variables
Shaded $\Rightarrow$ observed, empty (like Y_3) $\Rightarrow$ unobserved

Edges $e \in E$ describe the conditional independence relationships

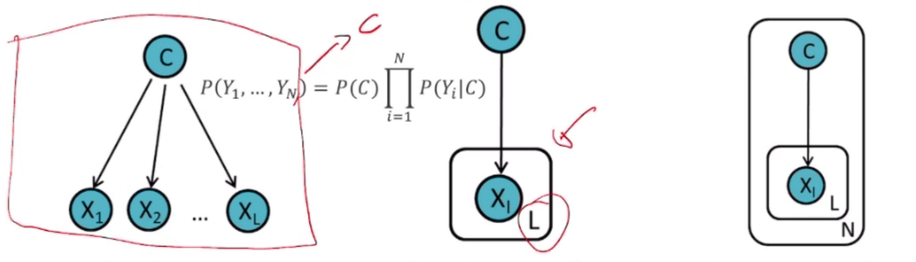


Conditional Probability Tables CPTs are local to each node and describe the probability distribution **given its parents**.

$$P(Y_1, \ldots, Y_n) = \prod_{i=1}^{N} P(Y_i \mid \text{Parents}(Y_i))$$

Plate notation If the same causal relationship is replicated for a number of variables, we can compactly represent it with plate notation.

$$P(Y_1, \ldots, Y_N, C) = P(C) \prod_{i=1}^{N} P(Y_i \mid C)$$

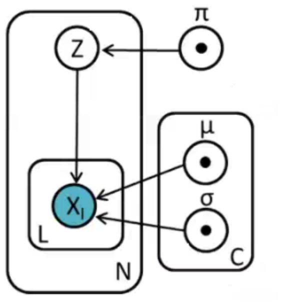


The Naive Bayes Classifier

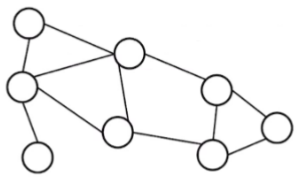
Replication for L attributes

Replication for N data samples

Full-Plate Notation Boxes denote replication for a number of times (denoted by the letter in the corner). Shaded nodes are observed variables, empty nodes are unobserved latent variables.
Black dots (optional) identify model parameters.



Markov Random Fields



Undirected graph $G = (V, E)$ (a.k.a. Markov Networks). Also with shaded/empty nodes to denote observed/unobserved variables.
Edges $e \in E$ represent bidirectional dependencies between variables (constraints).
Often arranged in a structure that is coherent with the data/constraint we want to model.
Often used in image processing to impose spatial constraints (e.g. smoothness)

0.3.3 Conditional Independence and Causality

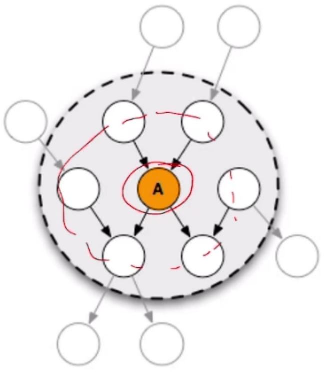
Can we reason on the structure of the graph to infer direct/indirect relationships between random variables?

Local Markov Property Each node (random variable) is conditionally independent of all its non-descendants given a joint state of its parents.

$$Y_v \perp Y_{V \setminus \text{Children(v)}} \text{ given } Y_{\text{Parent}(v)} \; \forall \, v \in V$$

There are substructures in the Bayesian networks with which we can build everything.

Markov Blanket A Markov blanket $Mb(A)$ of a node A is the minimal set of vertices that isolates/shields the node from the rest of the Bayesian network. If I know the variables in $Mb(A)$ then I know everything I need to know about A



Taking only the parents it's not sufficient, we need also the children and the co-parents (nodes that are parents of one of my children). So it contains parents, children and children's parents.

$$P(A \mid Mb(A), Z) = P(A \mid Mb(A)) \; \forall \, Z \notin Mb(A)$$

Joint Probability Factorization An application of the chain rule and local Markov property.

1. Pick a topological ordering of the nodes
2. Apply chain rule following the order

Sampling of a Bayesian Network A BN describes a generative process for observations.

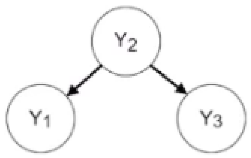
1. Pick a topological ordering of the nodes
2. Generate data by sampling from the local condition probabilities following this order

Generate ith sample for each variable, example $s_i \simeq P(S)$, $h_i \simeq P(H \mid S = s_i)$

0.3.4 Fundamental Bayesian Network Structures

Three fundamental substructures that determine the conditional independence relationships in a Bayesian network.

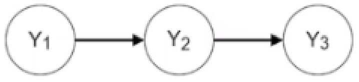
Tail to Tail Common cause



$$P(Y_1, Y_3 \mid Y_2) = P(Y_1 \mid Y_2)P(Y_3 \mid Y_2)$$

If Y_2 is unobserved, then Y_1, Y_3 are marginally dependent $Y_1 \not\perp Y_3$
If Y_2 is observed, Y_1, Y_3 become conditionally independent $Y_1 \perp Y_3 \mid Y_2$ (the path between Y_1, Y_3 is blocked by the observed (shaded) Y_2)

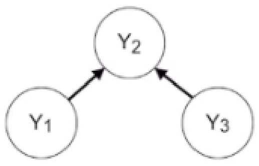
Head to Tail



$$P(Y_1, Y_3 \mid Y_2) = P(Y_1)P(Y_2 \mid Y_1)P(Y_3 \mid Y_2) = P(Y_1 \mid Y_2)P(Y_3 \mid Y_2)$$

Same behavior as before!
If Y_2 is unobserved, then Y_1, Y_3 are marginally dependent $Y_1 \not\perp Y_3$
If Y_2 is observed, Y_1, Y_3 become conditionally independent $Y_1 \perp Y_3 \mid Y_2$ (Y_2 again blocks the path)

Heat to Head Common effect



$$P(Y_1, Y_2, Y_3) = P(Y_1)P(Y_3)P(Y_2 \mid Y_1, Y_3)$$

If Y_2 is unobserved, then Y_1, Y_3 are marginally independent $Y_1 \perp Y_3$
If Y_2 is observed, then Y_1, Y_3 are conditionally dependent $Y_1 \not\perp Y_3 \mid Y_2$
If any Y_2 descendants is observed it unlocks the path.

Derived Conditional Independence Relationships A Bayesian network represent the local relationship encoded by the 3 basic structures plus the derived relationships.