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

$$\mathrm{x} = x_0, \ldots, x_N$$

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

$\forall\, t\ \ E[x_t] = \mu$
$\forall\, t\ \mathrm{Cov}(x_{t+\tau}, x_t) = \gamma_\tau$ 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 It's a revealing view on time series statistics.

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

$$\overline{\phi}_{\mathrm{x}_1\,\mathrm{x}_2}(\tau) = \frac{\phi_{\mathrm{x}_1\,\mathrm{x}_2}}{\sqrt{\sum_{t=0}^{T_1-1}(x_1(t))^2 \cdot \sum_{t=0}^{T_2-1}(x_2(t))^2}} \in [-1,+1]$$

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 $\mid |\alpha| < 1$

$\epsilon_t \Rightarrow$ 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 $\alpha_{i,1}, \dots, \alpha_{i,K}$ 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: $d(\mathrm{x}_1, \mathrm{x}_2) = \|\alpha_1 - \alpha_2\|_M^2$

novelty/anomaly detection: $\mathrm{TestErr}(x_t, \hat{x}_t) < \xi$ with $\hat{x}_t$ being the AR predicted value.

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 $\{e_1, \dots, e_k\}$ to represent any function $f \in E$

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

Given the orthonormal system

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

then the linear combination above becomes the Fourier series

$$\frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(kx) + b_k \sin(kx))$$

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 the power spectral density

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.

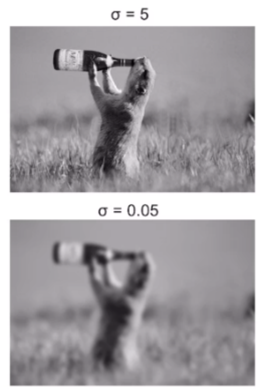
$$L_\sigma(x, y) = G(x, y, \sigma) * I(x, y)$$

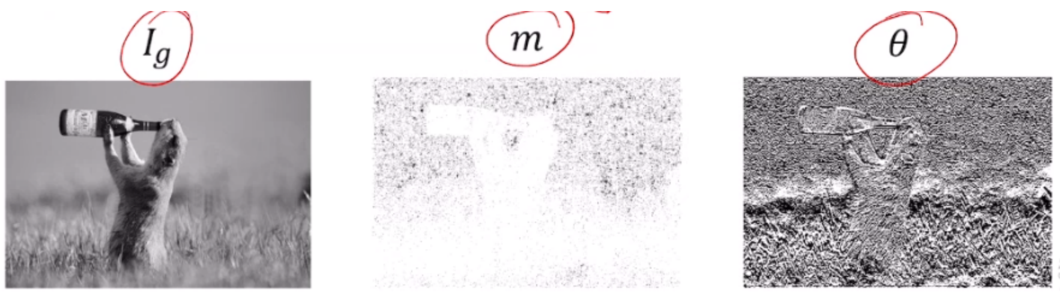
$$G(x, y, \sigma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

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

Gaussian Filter of an Image

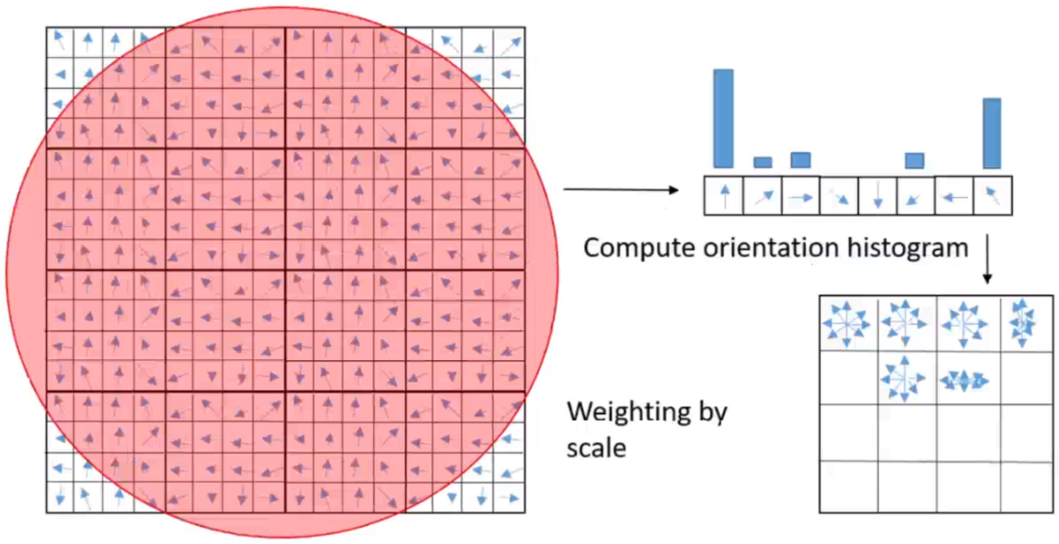
```
Iscale = imgaussfilt (I, sigma);
```





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

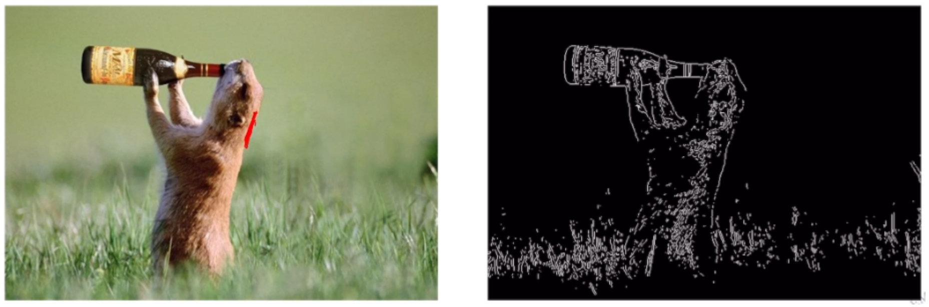


Detectors

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:

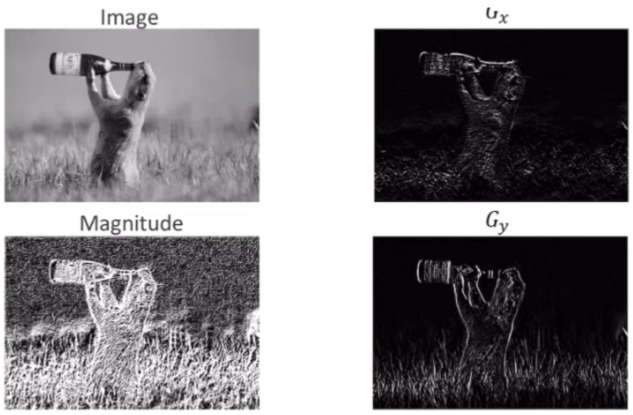
$G_x = \frac{\partial I}{\partial x} \simeq I(x+1,y) - I(x-1,y)$

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

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

Prewitt operators

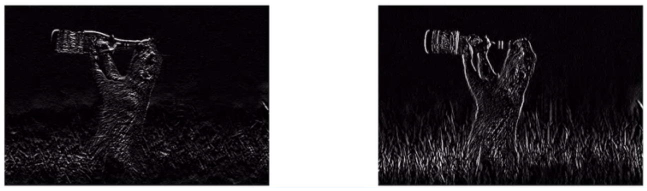
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$



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

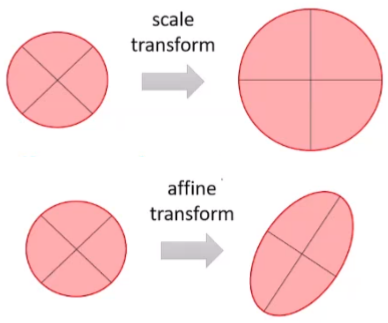
   Threshold

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 thresholds: stable with respect to variations in luminance, not scale dependent and doesn't assume circular regions. The key idea is to **take the blobs (extremal regions) which are nearly the same through a wide range of intensity thresholds**.
Blobs are generated (locally) by binarizing the image over a large number of thresholds:

Invariance to affine transformation of image intensities

Stability (they are stable on multiple thresholds)

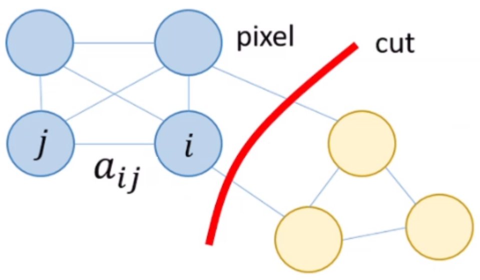
Multi-scale (connected components are identified by intensity stability not by scale)

Sensitive to local lightning effects, shadows...

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



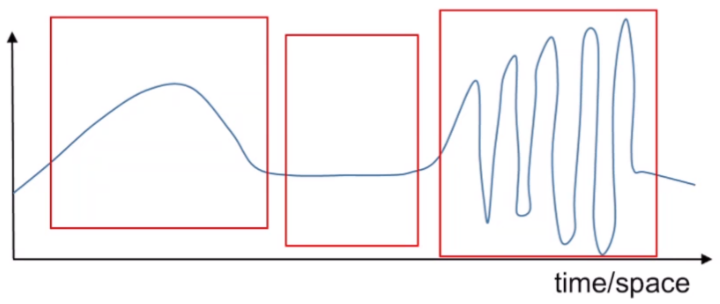
With each node being a pixel: an image is a graph. a_{ij} is the affinity between pixels at a certain scale σ. A cut of G is the set of edges such whose removal makes G a disconnected graph. Breaking the graph into pieces by cutting edges of low affinity.
The normalized cut problem is NP-hard, approximate 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

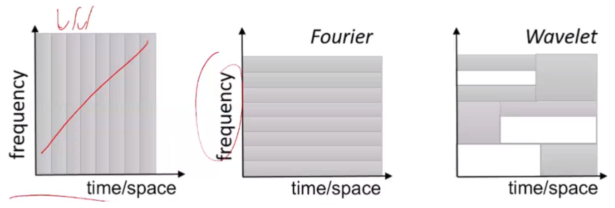
Image processing is a lot about convolutions: linear masks to perform gradient operations, gaussian functions to apply scale changes (zooming in and out). Computational efficiency is a driving factor: convolution in Fourier domain, superpixel, lightweight feature detectors…

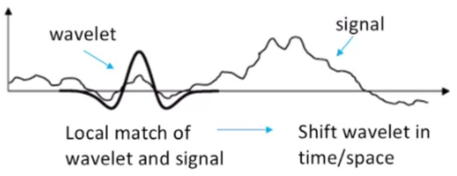
0.2.3 Wavelets

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.





wavelet
signal
Local match of wavelet and signal
Shift wavelet in time/space

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

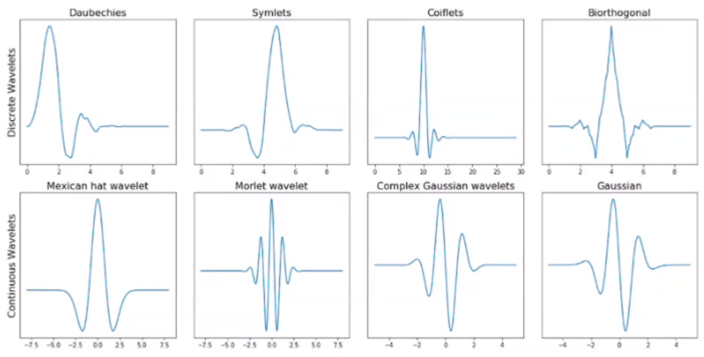
Split the signal with an orthonormal basis generate by translation and dilation of a mother wavelet

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

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.

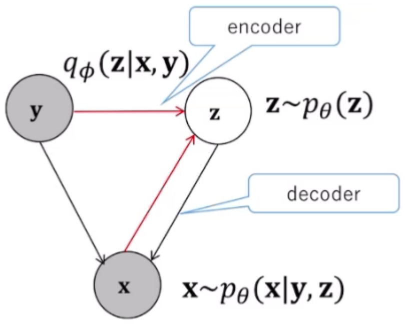
Inference How to query (predict with) a graphical model? Probability of unknown X given observations d, $P(X\,|\,d)$, the **most likely hypothesis** (parameters) X.

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 $P(X_1, \dots, X_n)$ 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 distributions: simplifies marginalization and inference algorithms, allowing to **incorporate prior knowledge** concerning causal relationships and associations between random variables.

Directed graphical models (Bayesian Networks)

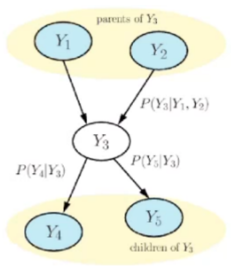
Undirected graphical models (Markov Random Fields)

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

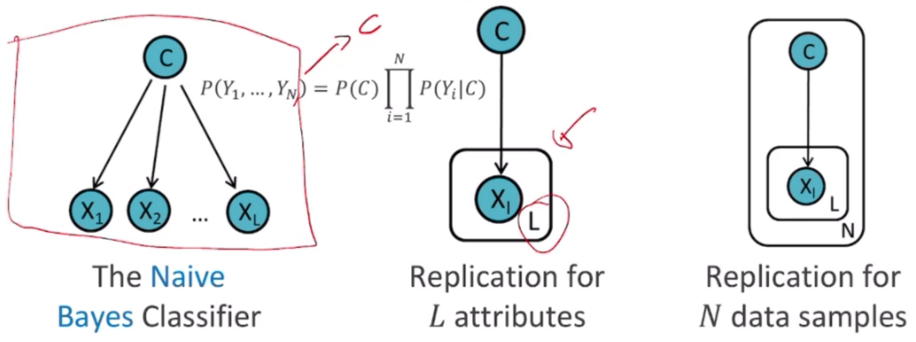


Conditional Probability Tables CPTs are local to each node and describe the probability distribution **given its parents**.

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

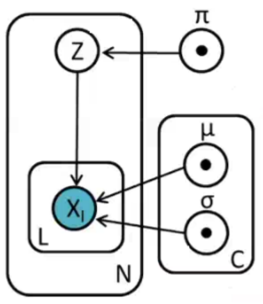
Plate notation If the same causal relationship is replicated for a number of variables, we can compactly represent it with plate notation.

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

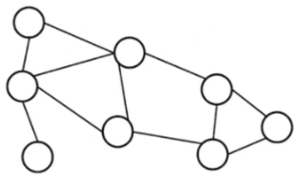


$P(Y_1, \dots, Y_N) = P(C) \prod_{i=1}^{N} P(Y_i|C)$
C
C
The Naive Bayes Classifier
Replication for L attributes
Replication for N data samples

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

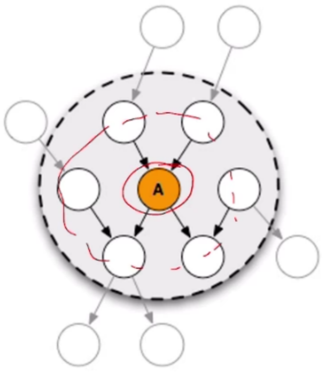
Can we reason on the structure of the graph to infer direct/indirect relationships between random variables?

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.

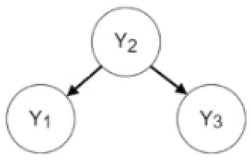
1. Pick a topological ordering of the nodes
2. Generate data by sampling from the local condition probabilities following this order

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.

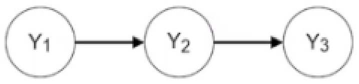
Tail to Tail Common cause



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

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)

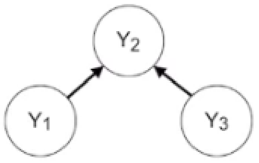
Head to Tail Causal Effect



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

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

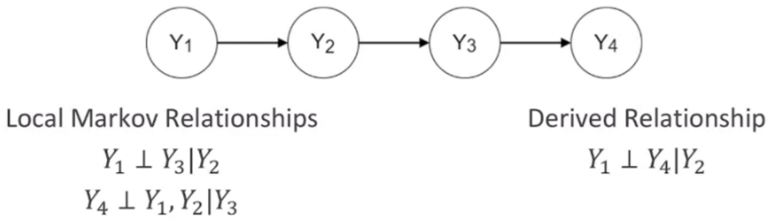


$$P(Y_1, Y_2, Y_3) = P(Y_1)P(Y_3)P(Y_2 \mid Y_1, Y_3)$$

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.

Given the same distribution I can have two different Bayesian Networks, which implies the same factorization.



***d*-separation** Let $r = Y_1 \leftrightarrow \ldots \leftrightarrow Y_2$ be an undirected path between Y_1, Y_2, r is d-separated by Z if there exist at least one node $Y_c \in Z$ for which path r is blocked. With Z being the set of variable for which we're assessing this separation.
In other words, this holds if at least one of the following holds:

r contains an head-to-tail structure $Y_i \to Y_c \to Y_j$ (or $Y_i \leftarrow Y_c \leftarrow Y_j$) and $Y_c \in Z$

r contains a tail-to-tail $Y_i \leftarrow Y_c \to Y_j$ and $Y_c \in Z$

r contains head-to-head $Y_i \to Y_c \leftarrow Y_j$ and neither Y_c nor its descendants are in Z

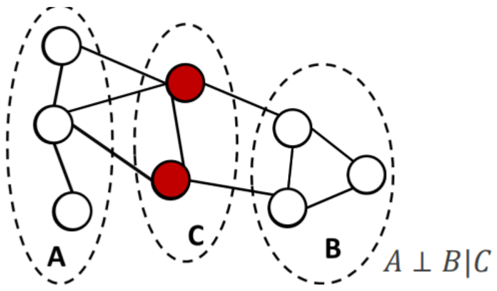
Two nodes Y_i, Y_j in a Bayesian Network G are d-separated by $Z \subset V \Leftrightarrow$ all undirected paths between Y_i, Y_j are d-separated by Z (denoted by $\text{Dsep}_G(Y_i, Y_j \mid Z)$)

Markov Blanket The Markov Blanket $Mb(Y)$ is the minimal set of nodes which d-separates a node Y from all other nodes (i.e. makes Y conditionally independent of all other nodes in the Bayesian Network)

$$Mb(Y) = \{\text{Parents}(Y), \text{Children}(Y), \text{Parents}(\text{Children}(Y))\}$$

Are Directed Models Enough? Bayesian Networks are used to model asymmetric dependencies. But Directed Models cannot express all conditional dependence relationships: expressing some precludes the expressions of others. What if we want to model symmetric dependencies: bidirectional effects, spatial dependencies... we need **undirected approaches**. Directed models cannot represent some bidirectional dependencies in the distributions.

0.3.5 Markov Random Fields



What is the undirected equivalent of d-separation in directed models? It's based on node separation: the two nodes in the middle separate the two lateral parts.
Node subsets $A, B \subset V$ are conditionally independent given $C \subset V \setminus \{A, B\}$ if all paths between nodes in A and B pass through at least one of the nodes in C.
The Markov Blanket of a node includes all and only its neighbors.

Joint Probability Factorization What is the undirected equivalent? We seek a product of functions defined over a set of nodes associated with some local properties of the graph. Markov blanket tells that nodes that are not neighbors are conditionally independent given the remainder of the nodes.

$$P(X_v, X_i \mid X_{V\setminus\{v,i\}}) = P(X_v \mid X_{V\setminus\{v,i\}})P(X_i \mid X_{V\setminus\{v,i\}})$$

Factorization should be chosen in a way that nodes X_v and X_i are not in the same factor: we use a well-known graph structure that includes only nodes that are pairwise connected.

Clique Subset of nodes C in graph G such that G contains an edge between all pair of nodes in C. It's maximal if you cannot add more nodes.

Maximal Clique Factorization Define $X = X_1, \dots, X_n$ as the random variables associated to the N nodes of the undirected graph G

$$P(X) = \frac{1}{Z}\prod_C \psi(X_C)$$

X_C are the random variables in the maximal clique C, $\psi(X_C)$ is the **potential function** over the maximal clique C and Z is the partition function ensuring normalization.

$$Z = \sum_X \prod_C \psi(X_C)$$

The partition function Z is the computational bottleneck of undirected modes: $O(K^N)$ for N discrete random variables with K distinct values.

Potential Functions Potential functions $\psi(X_C)$ are not probabilities, they express which configuration of the local variables are preferred. For example $\psi(X_1, X_2) = \begin{cases} 1 & \text{if } X_1 = X_2 \\ 4 & \text{if } X_2 = 2X_1 \\ 0 & \text{otherwise} \end{cases}$: you can hand-engineer feature functions.
If we restrict to strictly positive potential functions, the Hammersley-Clifford theorem provides guarantees on the distribution that can be repreesented by the clique factorization.

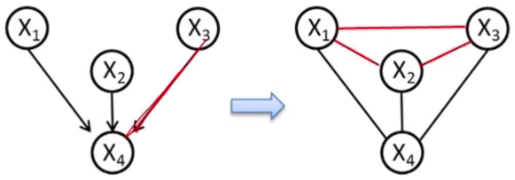
Boltzmann Distribution A convenient and widely used strictly positive representation of the potential function is

$$\psi(X_C) = e^{-E(X_C)}$$

where $E(X_C)$ is called **energy function**.

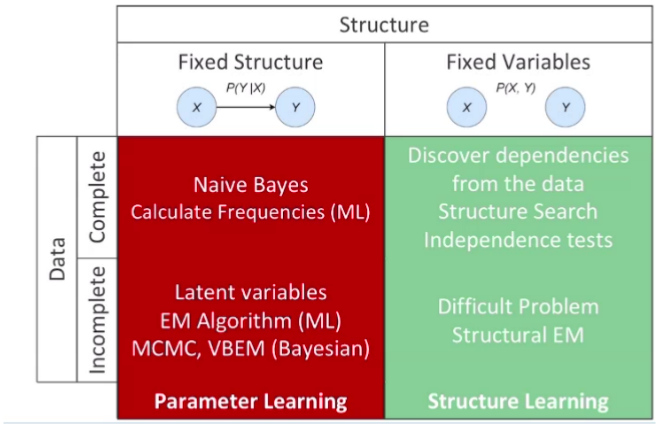
From Directed to Undirected

Straightforward when is linear.
Requires some work with v-structures, e.g. **moralization** (a.k.a. marrying of the parents).



0.3.6 Learning Causation from Data

Learning with Bayesian Network



Structure Learning Problem Observations are given for a set of fixed random variables, and network structure is not specified:

Determine which arcs exist in the network (causal relationships)

Compute Bayesian network parameters (conditional probability tables)

Determining causal relationships between variables entails deciding on arc presence and directing edges.

Structure Finding Approaches

Search and Score
A model selection approach, a search in the space of the graphs.
Search the space Graph(Y) of graphs G_k that can be built on the random variables $Y = Y_1, \dots, Y_N$, scoring each structure by $S(G_k)$ and returning the highest scoring graph G^*. So two fundamental aspects: the scoring function and the search strategy.
Scoring function: two fundamental properties:

Consistency: same score for graphs in the same equivalence class

Decomposability: can be locally computed

Two approaches:

Information theoretic: based on data likelihood plus some model-complexity penalization terms

Bayesian: score the structures using a graph posterior (likelihood plus proper prior choice

Search strategy:

Finding maximal scoring structures is NP complete

Constrain search strategy: starting from a candidate structure we modify iteratively by local operations (edge/node addition/deletion). Each operation has a cost, so a cost optimization problem.

Constrain search space can be

Known node order: can reduce the search space to the parents of each node (Markov Blankets)

Search in the space of structure equivalence classes

Search in the space of node ordering

Constraint Based
Tests of conditional independence $I(X_i, X_j \mid Z)$, constraining the network. Based on measures of association between two variables X_i and X_j given their neighbor nodes Z.
Testing strategy:

Choice of the testing order is fundamental in avoiding a super-exponential complexity.

Level-wise testing: tests $I(X_i, X_j \mid Z)$ are performed in order of increasing size of the conditioning set Z starting from $Z = \emptyset$ (PC algorithm)

Node-wise testing: tests are performed on a single edge at the time, exhausting independence checks on all conditioning variables (TPDA algorithm)

The nodes entering Z are chosen in the neighborhood of X_i, X_j

Hybrid
Model selection of constrained structures. Multi-stage algorithm combining previous approaches: independence tests to find a good sub-optimal skeleton as starting point, then search and score refining the skeleton.
Max-Min Hill Climbing (MMHC) model: optimized constraint-based approach to reconstruct the skeleton, using the candidate parents in the skeleton to run a search and score approach.

0.3.7 Hidden Markov Models

Sequence A sequence y is a collection of observations y_t where t represent the position of the element according to a complete order (e.g. time)

$$y_1 \to \ldots \to y_{t-1} \to y_t \to \ldots \to y_T$$

$$P(y_t \mid y_{t-1})$$

Also head-to-tail: observation at time t is independent from $t = 1, \ldots, t-1$: **first-order Markov assumption**.
Reference population is a set of independent and identically distributed sequences $y^1, \ldots, y^N$
Difference sequences generally have different lengths $T^1, \ldots, T^N$

Markov Chain First-Order Markov Chain is a directed graphical model for sequences such that element x_t only depends on We have X $= x - 1, \ldots, x_T$ that can be represented as

$$x_1 \to \ldots \to x_{t-1} \to x_t \to \ldots \to x_T$$

So we can write

$$P(\mathrm{X}) = P(x_1, \ldots, x_T) = P(x_1) \cdot \prod_{i=2}^{T} P(x_i \mid x_{i-1})$$

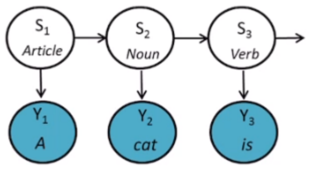
because $P(x_i \mid x_{i-1})$ is the same whenever the t.
$P(x_1)$ is the **prior distribution** (x_1 has nothing "before" it) and $P(x_i \mid x_{i-1})$ is the **transition distribution**.

If I assume $x_t \in \{a, \ldots, z\}$, so of 25 elements, this gives $P(x_1) = P(x_1 = \text{letter})$ so $P(x_1)$ is a vector with each position being the probability of x_1 being that letter. Summing the vector elements gives 1, because it's a distribution of probabilities.
$P(x_i \mid x_{i-1})$ is a 25×25 matrix: in position n, b is $P(x_i = n \mid x_{i-1} = b)$. The elements in a single column will give 1, because conditional probability gives a family of distribution: for each assignment I have a distribution.

The general form is the Lth order Markov chain, when x_i depends on L predecessors.

Observed Markov Chains We can use the Markov chain to model the relationships between observed elements in a sequence. The problem is that we can do that only pairwise: computational issue (very large matrices) and e.g. only co-occurrence of 2 words so unapplicable to natural language.
So we need to abstract from symbols to category: not relationship between words, but relationships between the general concepts represented by those words. The categories are not observable: Markov chain over non-observable elements.

Hidden Markov Models HMM infer categories: stochastic process where transition dynamics is disentangles from observations generated by the process.



S_i are **hidden states**, finite $i = 1, \dots, C$.
We need **clustering algorithms**: clustering symbols into a finite set of non-observable elements.
Multinomial state transition

$$A_{ij} = P(S_t = i \mid S_{t-1} = j)$$

Prior probability (**stationary assumption**)

$$\pi_i = P(S_1 = i)$$

Emission distribution (the "down arrow" $\begin{matrix} S_t \\ \downarrow \\ Y_t \end{matrix}$)

$$b_i(y_t) = P(Y_t = y_t \mid S_t = i)$$

HMM Joint Probability Factorization Discrete state HMMs are parameterized by the finite number of hidden states C and $\Theta = (\pi, A, B)$:

π prior distribution

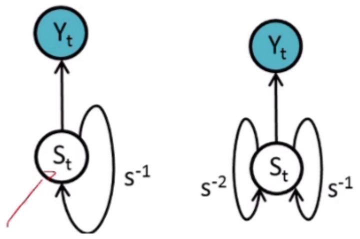
A state transition

B emission distribution (or its parameters)

$$P(Y = y) = \sum_s P(Y = y, S = s) =$$

$$= \sum_{s_1,\dots,s_T} \left(P(S_1 = s_1) P(Y_1 = y_1 \mid S_1 = s_1) \prod_{t=2}^{T} P(S_t = s_t \mid S_{t-1} = s_{t-1}) P(Y_t = y_t \mid S_t = s_t) \right)$$

HMMs as Recursive Models A graphical framework describing how contextual information is recursively encoded by both probabilistic and neural models.



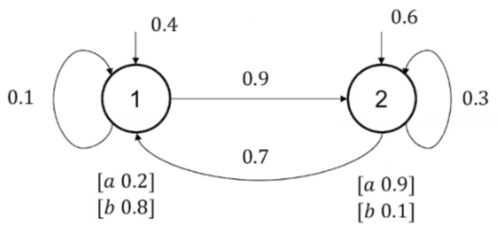
Indicates that the hidden state S_t at time t is dependent on context information from

the previous timestep s^{-1}, first-order

the previous two timesteps s^{-1}, s^{-2}, second-order

and so on.

HMMs as Automata Can also be generalized to transducers.



0.3.8 Notable Inference Problems

Smoothing Given a model Θ and an observed sequence y, determine the distribution of the hidden state at time t: $P(S_t \mid Y = y, \Theta)$
Forward Backward algorithm.

Learning Given a dataset of N sequences $D = \{y^1, \dots, y^N\}$ and the number of hidden states C, find the parameters π, A, B that maximize the probability model $\Theta = \{\pi, A, B\}$ having generated the sequences in D

Optimal State Assigment Given a model Θ and an observed sequence y, find an optimal state assignment $s = s_1^*, \dots, s_T^*$ for the hidden Markov chain.
Viterbi algorithm.

Forward-Backward Algorithm

Smoothing: how do we determine $P(S_t = i \mid \hat{y})$? We will compute $P(S_t = i, \hat{y})$, it's proportional (just divide by $P(\hat{y})$).
I know Θ, the model (its parameters). So I know $P(S_1) = \pi, P(S_t \mid S_{t-1} = A$ and $P(y_t \mid S_t) = B$: I need to express the quantity I want in terms of $\Theta = \{\pi, A, B\}$.
$\hat{y}$ are all the observations for each timestep $\Rightarrow P(S_t = i, y_1, \dots, y_{t-1}, y_t, y_{t+1}, \dots, y_T)$

$$S_1 \to \dots \to S_{t-1} \to S_t \to S_{t+1} \to \dots \to S_T$$

$$\downarrow \to \dots \to \downarrow \to \downarrow \to \downarrow \to \dots \to \downarrow$$

$$y_1 \to \dots \to y_{t-1} \to y_t \to y_{t+1} \to \dots \to y_T$$

We are at time t, so everything after that is the future $(y_{t+1:T})$, and everything up to t included is the past $(y_{1:t})$.

$$P(S_t = i, y_1, \dots, y_{t-1}, y_t, y_{t+1}, \dots, y_T) = P(y_{t+1:T} \mid S_t = i, y_{1:t}) P(S_t = i, y_{1:t})$$

If I observe S_t we block the path $y_{1:t}$, so $P(y_{t+1:T} \mid S_t = i, y_{1:t}) = P(y_{t+1:T} \mid S_t = i)$

$$P(S_t = i, y_1, \dots, y_{t-1}, y_t, y_{t+1}, \dots, y_T) = P(y_{t+1:T} \mid S_t = i) P(S_t = i, y_{1:t})$$

I can derive two "messages"

Past message $\alpha_t(i) = P(S_t = i, y_{1:t})$ (**forward recursion**)

Future message $\beta_t(i) = P(y_{t+1:T} \mid S_t = i)$ (**backward recursion**)

$$P(S_t, y_{1:t}) = \sum_{j=1}^{c} P(S_t, S_{t-1} = j, y_{1:t}) = \sum_{j=1}^{c} P(y_t \mid S_t, S_{t-1} = j, y_{1:t-1}) P(S_t, S_{t-1}, y_{1:t-1})$$

But we can get rid of $S_{t-1} = j$ and $y_{1:t-1}$ leaving us with $P(y_t \mid S_t)$ which is just the emission.
The second factor can be rewrited as $P(S_t \mid S_{t-1} = j, y_{1:t-1}) P(S_{t-1} = j \mid y_{1:t-1})$ and observing S_{t-1} allows us to get rid of $y_{1:t-1}$, giving us the transition distribution $P(S_t \mid S_{t-1} = j)$ and $\alpha_{t-1}(j)$

$$\alpha_t(i) = P(S_t = i, y_{1:t}) = \sum_{j=1}^{c} P(y_t \mid S_t = i) P(S_t = i \mid S_{t-1} = j) \alpha_{t-1}(j)$$

$$\alpha_1(j) = P(y_1 \mid S_1 = j) P(S_1 = j)$$

This just by reasoning with conditional independence.
Same thing can be done for

$$P(y_{t+1:T} \mid S_t = i) = \sum_j P(y_{t+1:T}, S_{t+1} = j \mid S_t = i) = \sum_j P(y_{t+2:T} \mid S_t, S_{t+1}, y_{t+1}) P(S_{t+1}, y_{t+1} \mid S_t = i)$$

Same as before, I can exclude S_t, y_{t+1} because we observe S_{t+1} so that factor is $\beta_{t+1}(j)$.
The second factor is rewritten as $P(S_{t+1} \mid S_t, y_{t+1}) P(y_{t+1} \mid S_{t+1})$ which is the transition distribution (we can exclude y_{t+1}) times the emission distribution.

$$\beta_t(i) = \sum_j P(y_{t+1} \mid S_{t+1} = j) P(S_{t+1} = j \mid S_t = i) \beta_{t+1}(j)$$

$$\beta_T = 1$$

Sum-Product Message Passing The Forward-Backward algorithm is an example of a sum-product message passing algorithm.
A forward recursion computing a generic message μ_α, backward recursion computing a generic message μ_β
A general approach to efficiently perform exact inference in graphical models, with $\alpha_t \equiv \mu_\alpha(X_n)$ and $\beta_t \equiv \mu_\beta(X_n)$

$$\mu_\alpha(X_n) = \sum_{X_{n+1}} \psi(X_n, X_{n+1})\mu_\beta(X_{n+1})$$

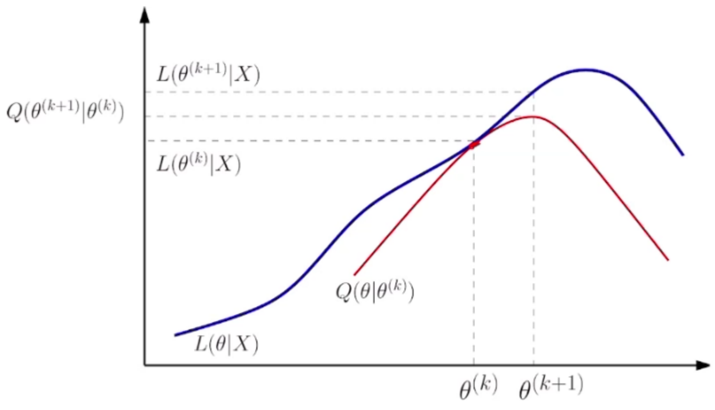
Learning in HMM

Learning parameters $\Theta = (\pi, A, B)$ by **maximum** (log) **likelihood**

$$L(\Theta) = \log \prod_{n=1}^{N} P(Y^n \mid \Theta) = \log \prod_{n=1}^{N} \left(\sum_{S_1^n,\dots,S_{T_n}^n} P(S_1^n)P(Y_1^n \mid S_1^n) \prod_{t=2}^{T} P(S_t^n \mid S_{t-1}^n)P(Y_t^n \mid S_t^n) \right)$$

Maximizing the joint likelihood of the sequences given the parameters considering them independent and identically distributed. We have to deal with the unobserved S_t^n and the nasty sum in the log.
Expectation-Maximization of the **complete likelihood** $L_c(\Theta)$, optimizing a slightly different problem obtaining a not-reducing similar result. It's completed with indicator variables $z_{ti}^n = \begin{cases} 1 & \text{if } n\text{th chain is in state } i \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$ about the assignments S_i^n

Expectation-Maximization Gives the red line: touching in the estimating point and not greater in the other points.



It's a matter of picking the right $Q(\Theta \mid \Theta^k)$
Introduce indicator variables in $L(\Theta)$ together with model parameters $\Theta = (\pi, A, B)$

$$L_C(\Theta) = \log P(X, Z \mid \Theta) =$$

But I built it assuming to know z, but I don't know it. The *expectation* part is in this: I don't know z_{ti}^n, but you can optimize the function in expectation $E[L_C(\Theta)]$
It's a 2-step iterative algorithm for the maximization of complete likelihood $L_C(\Theta)$ with respect to the model parameters Θ

E-step: given the current estimate of the model parameters Θ^t, compute

$$Q^{t+1}(\Theta \mid \Theta^t) = E_{Z \mid X, \Theta^t}[\log P(X, Z \mid Theta)]$$

So compute the expectation of the complete log likelihood with respect to indicator variables... Expectation with respect to a discrete random variable is

$$E_z[Z] = \sum_z z \cdot P(Z = z)$$

M-step: find the new estimate of the model parameters

$$\Theta^{t+1} = \arg\max_{\Theta} Q^{t+1}(\Theta \mid \Theta^t)$$

With appropriate Lagrange multiplier is multinomial.

Usefulness of HMMs

> **Regime Detection**: for example, you can only observe the volatility and you can model it according to a HMM that can capture it. For example with 2 states a model can be too simple, you can add hidden state (for example a 5-state HMM).
> The hidden states are **clustering the observations**.

Decoding Problem Find the optimal state assignment s $= s_1^*, \dots, s_T^*$ for an observed test sequence y given a trained HMM. No unique interpretation of the problem.
Can be done identifying the single hidden states s_t that maximize the posterior

or find the most likely **joint hidden state assignment**

$$\mathrm{s}^* = \arg\max_s P(Y, S = \mathrm{s})$$

Viterbi Algorithm Efficient dynamic programming algorithm based on a backward-forward recursion, example of max-product message passing algorithm. When exchanging, instead of $\sum$ we maximize the $\prod$.

$$\max_{\hat{s}=s_1,\dots,s_T} P(\hat{y}, \hat{s}) = \max_{\hat{s}} \prod_{t=1}^{T} P(y_t \mid s_t) P(s_t \mid s_{t-1})$$

because is emission and prior as always. Let's focus on a simplified problem: first try to find the state that maximize s_T. Let's focus on T

$$\max_{s_T} \prod_{t=1}^{T} P(y_t \mid s_t) P(s_t \mid s_{t-1}) =$$

we can exclude a lot

$$= \prod_{t=1}^{T-1} P(y_t \mid s_t) P(s_t \mid s_{t-1}) \cdot \max_{s_T} P(y_T \mid s_T) P(s_T \mid s_{T-1})$$

which is a unique term, let's call $\epsilon(s_{T-1}) = \max_{s_T} P(y_T \mid s_T) P(s_T \mid s_{T-1})$, and s_{T-1} has c possible values, the number of hidden states.
So $\epsilon(s_{T-1})$ it's a vector of c positions, and in position j we have $\epsilon(s_{T-1} = j)$

$$\prod_{t}^{T-1} P(y_t \mid s_t) PSs_t \mid s_{t-1}) \epsilon(s_{T-1})$$

Let's try solving

$$\max_{s_{T-1}} \prod_{t}^{T-1} P(y_t \mid s_t) PSs_t \mid s_{t-1}) \epsilon(s_{T-1})$$

we would do the same procedure. So we can iteratively start from the last item and use the information iteratively to compute the previous one.
In general we compute

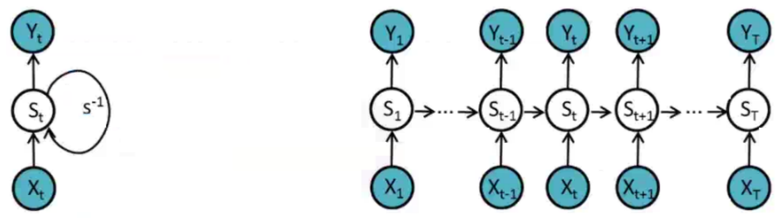
$$\epsilon(s_{t-1}) = \max_{s_t} P(y_t \mid s_t) P(s_t \mid s_{t-1}) \epsilon(s_t)$$

So s_t is received by s_{t-1} to compute the new $\epsilon(s_{t-1})$ which is in turn passed to s_{t-2} and so on, ending at the root. In practice we never choose the state, only computing the maximum. At the root, I have no predecessor states and can solve the maximization problem

$$s_1^* = \arg\max_{s_1} P(y_1 \mid s_1) P(s_1)$$

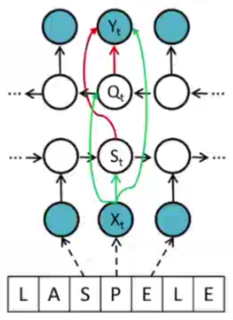
From state s_1 we pick up s_1^* and send it to s_2, which will use this information and pick the correct state that maximize the ϵ.

0.3.9 Input-output Hidden Markov Models

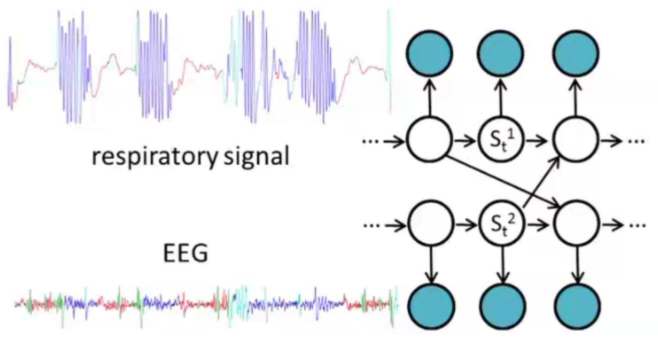


Translates an input sequence in an output sequence (**transduction**). State transition and emission depend on input observations (**input-driven**).
Recursive model highlights analogy with RNNs.

Bidirectional Input-Driven Models Remove causality assumption that current observation



Coupled HMMs Describing interacting processes whose observation follow different dynamics while the underlying generative processes are interlaced.



Dynamic Bayesian Networks HMMs are a specific and the simples case of a class of directed models that represent dynamic processes and data with changing connectivity template. Other examples are: Hierarchical HMMs and structure changing information.
DBNs are graphical models whose structure changes to represent evolution across time and/or between different samples.

0.3.10 Markov Random Fields

MFRs are undirected graphs $G = (V, E)$. Nodes $v \in V$ are random variables X_v ecc.

Likelihood Factorization Define X $= X_1, \dots, X_n$ as the random variables associated to the N nodes in the undirected graph G

$$P(\mathrm{X}) = \frac{1}{Z} \prod_c \psi_c(X_c)$$

X_C are the random variables associated to the maximal clique C, $\psi_C(X_C)$ is the potential function for clique C With Z normalization term used to transform to probability.

$$Z = \sum_X \dots$$

As already stated, potential functions are not probabilities, they express which configurations of the local variables are preferred.

$$\psi_C(X_C) = \exp\left(-E(X_C)\right)$$

with $E(X_C)$ called energy function.
In general we will assume to work with MRF where the partition functions factorize as

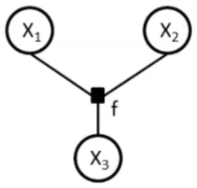
$$\psi_C(X_C) = \exp\left(-\sum_{k=1}^{K} \theta_{Ck} f_{Ck}(X_C)\right)$$

K defines the number of feature functions that we use, so the cardinality of a dictionary of feature functions f_{Ck}

$\theta_{Ck} in R$ are parameters

Undirected graphical models do not express the factorization of potentials into feature functions, you cannot express f graphically $\Rightarrow$ **factor graphs**.

Factor Graphs Random Variables are still circular nodes, factors f_{Ck} are denoted with square nodes and edges connect a factor to the random variable.



$$\psi(X_1, X_2, X_3) = f(X_1, X_2, X_3)$$

Sum-Product Inference A powerful class of exact inference algorithms. Use factor graph representation to provide a unique algorithm for directed/undirected models. So factor graph are a "unifying language" to represent both models, directed and undirected.
Inference is **feasible for chain and tree structures**. We restructure the graph to obtain a tree-like structure to perform message passing (junction tree algorithm) and then approximated inference (variational, sampling).
Even better: we constrain the MRF to obtain tractable classes of undirected models.

Restricting to Conditional Probabilities In ML a part of the random variables can be assumed to be always observable (input data).

X_k are observable inputs in the factor k

Y_k are hidden random variables

$f_k(X_k, Y_k)$ is the factor feature function

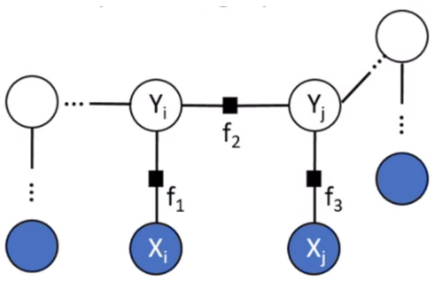
Instead of $P_\theta(x, y)$ we compute $P_\theta(y|x) \cdot P_\theta(x)$. Under this assumptio we can directly model the conditional distribution

$$P(Y \mid X) = \frac{1}{Z(X)} \prod_k \exp\left(\theta_k f_k(X_k, Y_k)\right)$$

We note that Z depends on X

$$Z(X) = \sum_y \prod_k \exp\left(\theta_k f_k(X_k, Y_k = y)\right)$$

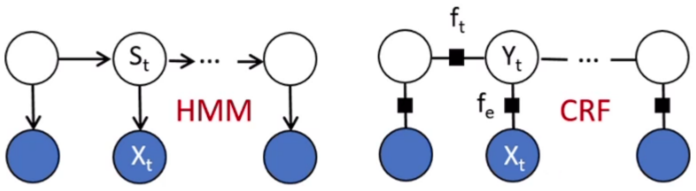
Conditional Random Field CRF are constrained MRF models representing input-conditional distributions.



Feature Functions Represent coupling or constraints between random variables, and are often very simple such as linear functions.

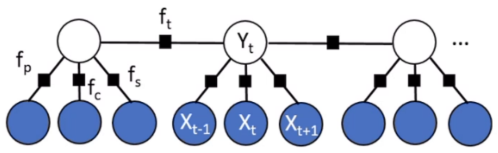
Discriminative Learning in Graphical Models X is always observable input while Y **can** be unobserved. Let's consider a single Y and multiple Xs. We can observe the Y_n corresponding to X_n for some n, and we can use this information to fit θ in $P(Y \mid X, \theta)$

CRF for Sequences Undirected and dicriminative equivalent of an HMM.



Meaning that $f_t(Y_{t-1}, Y_t)$ and it looks like $P(S_t \mid S_{t-1})$ of the HMM. $f_e(X_t, Y_t)$ looks like the emission $P(X_t \mid S_t)$, but **I can place as many feature functions** f_t **I want between the same variables** while I can't place more transition probabilities. The other difference is the main essences.

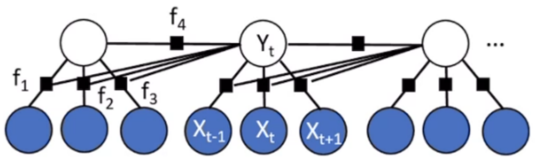
Generalization of HMM CRF are much more powerful



Each hidden state may depend on the previous, with f_t, but also on the emissions for the previous, current and next symbols. This cannot be easily implemented in HMMs. Kind of time stationality.

$$P(Y \mid X, \Theta) = \frac{1}{Z(X)} \prod_t \exp(\ldots)$$

The cliques are simple $(Y_t, X_{t-1}), (Y_t, X_t), (Y_t, X_{t+1}), (Y_t, Y_{t-1})$ for each t.
We can also model explicitly input influence on transition.



Use indicator variables in f_k definition to include or disregard the influence of specific random variables.

Posterior Inference in LCRF Is there an equivalent of the smoothing problem? Yes

$$P(Y_t, Y_{t-1} \mid X)$$

Solved by exact forward backward inference. Sum-product message passing (alpha-beta recursion) on the LCRF factor graph.

Also Viterbi can be done, because we can do Max-Product. The expensive part is the computation of the exponential summation in $Z(X)$. The forward-backward algorithm computes it efficiently as normalization term of $P(Y_t, Y_{t-1} \mid X)$. More articulated posteriors interact with $Z(X)$, which is a summation over everything that's not observable, difficult when there's a lot of unobservable variables. Exact inference in CRF other than chain-like is likely to be computationally impractical. We need to approximate: Markov Chain Monte Carlo…

Maximum (conditional) log-likelihood, for training

$$\max_{\Theta} L(\Theta) = \max_{\Theta} \sum_{n=1}^{n} \log P(y^n \mid x^n, \Theta)$$

we can substitu the LCRF conditional formulation because $P(y^n \mid x^n, \Theta) = \frac{1}{Z(X)} \exp(\sum \Theta_k f_k)$

$$= \sum_n \sum_t \sum_k a - b$$

To get proper marginalization $Z(X^n) = \sum_{y_t, y_{t-1}} \sum_t \exp(\sum_k \Theta_k f_k(y_t, y_{t-1}, x_t^n))$
With $\frac{\partial L(\Theta)}{\partial \Theta_k}$ we can maximize it because typically $L(\Theta)$ cannot be maximized in closed form.

We have sum of expectations, first when Y is not random, with samples drawn from a finite dataset and on the right the expectation using the posterior so the expectation of the feature function under the model distribution.

$$\frac{\partial L(\Theta)}{\partial \Theta_k} = E[f_k(y, y', x_t^n)] - E_{P(Y \mid X, \Theta)}[f_k(y, y', x_t^n)]$$

We need to match those two expectations.
There's a regularization term $\sum_k \frac{\Theta_k^2}{2\sigma^2}$ on $\|\Theta\|^2$, a posteriori regularization on the gaussian $P(\Theta)$ with $\mu = 0$ mean and $\sigma^2 I$ variance.
In practice, then, Θ can be learn with stochastic gradient descent or variants.

Applications Linear CRF have various applications: POS-tagging, semantic role identification, bioinformatics. Feature functions have the form

In computer vision they can be used to define bi-dimensioal lattice on images, bg/fg segmentation and to impost constraints.

0.4 Bayesian Learning and Variational Inference

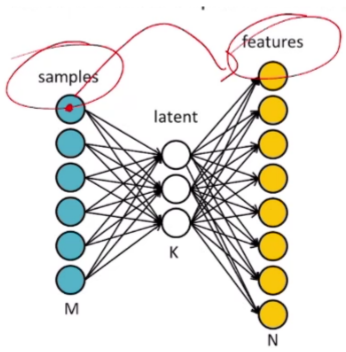
Introducing basic concepts of variational learning useful for both generative models and deep learning.

Latent Variables Unobserved random variables that define a hidden generative process of observed data. They explain complex relation between many observable variables. An example: hidden states in HMM/CRF.
Latent variables likelihood

$$P(x) = \int_z \prod_{i=1}^{N} P(x_i \mid z) P(z) dz$$

with $Z \longrightarrow X$

Latent Spaces Spaces where high-dimensional data can be represented. Each of the M samples is made of N features represented with $k << N$ dimensions.



The assumption is

Tractability Hidden variables can make intractable the posteriors. We need stuf that simplify those posteriors.

0.4.1 Kullback-Leiber Divergence

An information thoretic measure of closeness of two distributions p and q

$$KL(q\|p) = E_q\left[\log \frac{q(z)}{q(z \mid x)}\right] = \langle \log q(z) \rangle_q - \langle \log p(z) \rangle_p$$

Tells how the distribution q differs from p. It's a divergence so it is asymmetric, $KL(q\|p) \neq KL(p\|q)$

if q and p high, then good

if q is high and p is low, the it's unhappy

if q is low we don't care (due to the expectation)

Jensen Inequality Property of linear operators on convex/concave functions

$$\lambda f(x) + (1-\lambda) f(x) \geq f(\lambda x + (1-\lambda) x)$$

The curve is longer than the line connecting the two points. With concave we have $\leq$. Applied to probability

$$f(E[X]) \geq E[f(X)]$$

$$\log(E[X]) \geq E[\log(X)]$$