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

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

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

Given the orthonormal system

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

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

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

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)

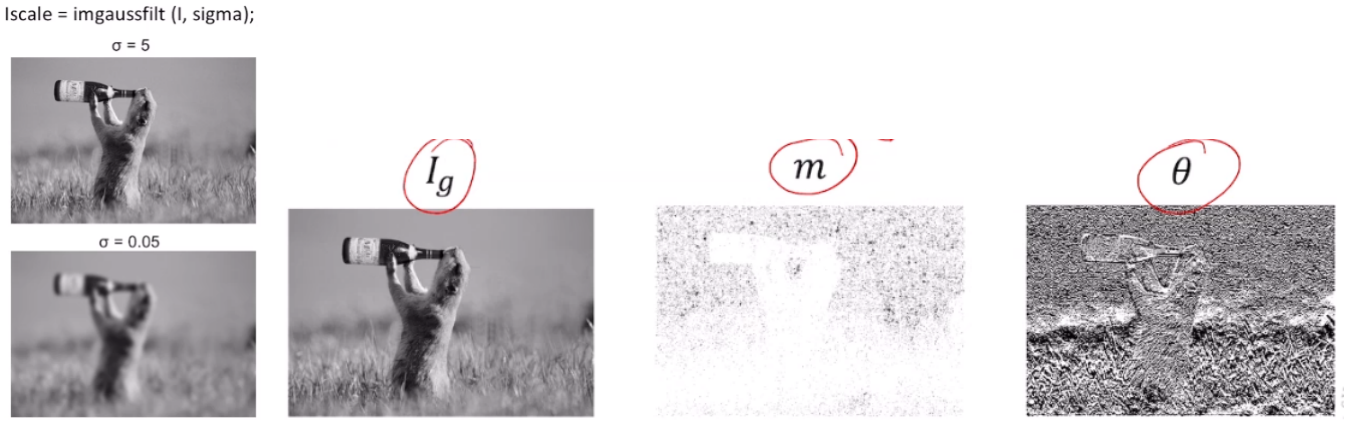
   Convolve the image with a Gaussian filter with standard variation σ, basically computing average of pixels with the coefficient taken from a Gaussian distribution. With a smooth Gaussian, we artificially smooth the object, and vice versa. We can compute different versions of the image.

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

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

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

Gaussian Filter of an Image



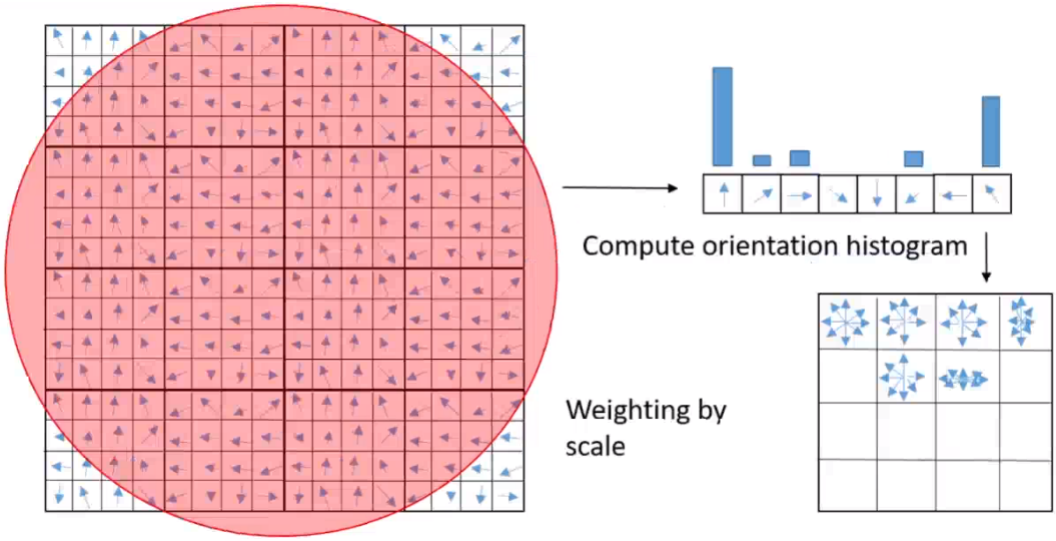
4. Create **gradient histogram**

   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

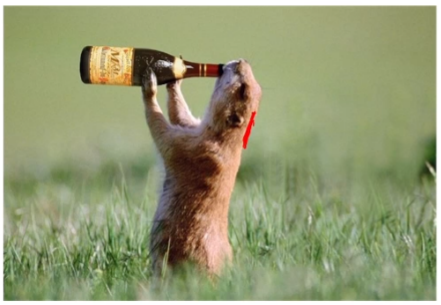


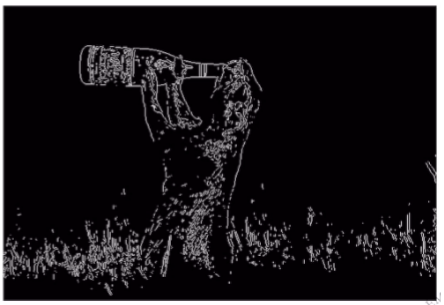
Detectors

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:

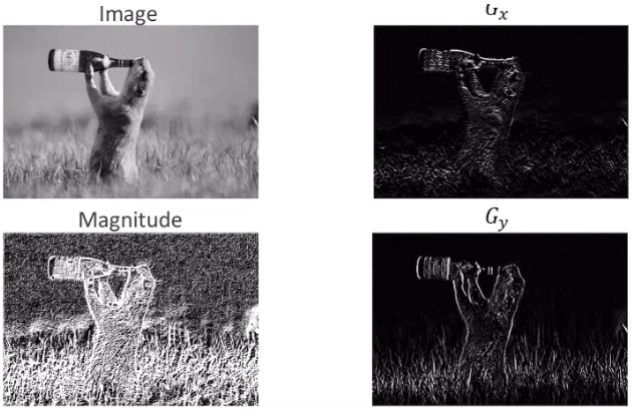
$$G_x = \frac{\partial I}{\partial x} \simeq I(x+1, y) - I(x-1, y)$$

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

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

Prewitt operators

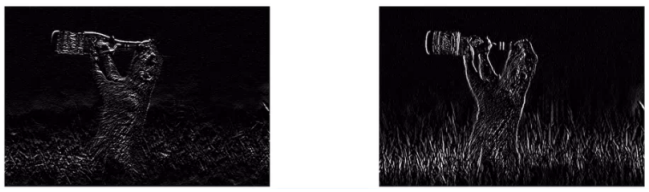
$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$



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

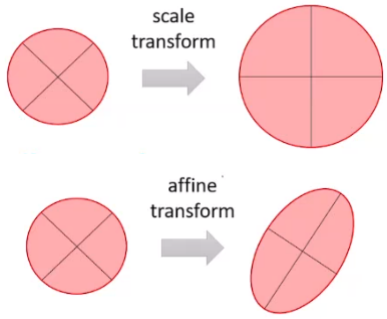
   Threshold

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)

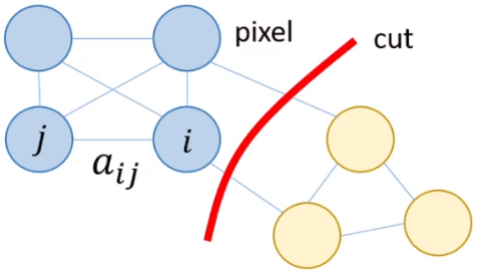
Multi-scale (connected components are identified by intensity stability not by scale)

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



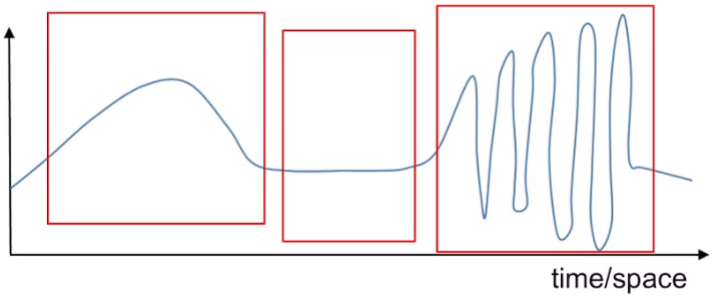
With each node being a pixel: an image is a graph. a_{ij} is the affinity between pixels at a certain scale σ. A cut of G is the set of edges such whose removal makes G a disconnected graph. Breaking the graph into pieces by cutting edges of low affinity.
The normalized cut problem is NP-hard, approximate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

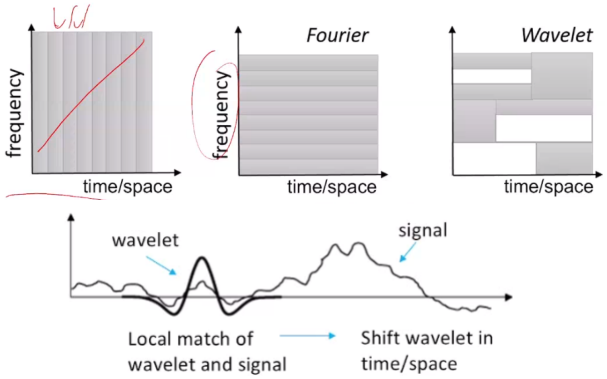
Image processing is a lot about convolutions: linear masks to perform gradient operations, gaussian functions to apply scale changes (zooming in and out). Computational efficiency is a driving factor: convolution in Fourier domain, superpixel, lightweight feature detectors...

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.



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

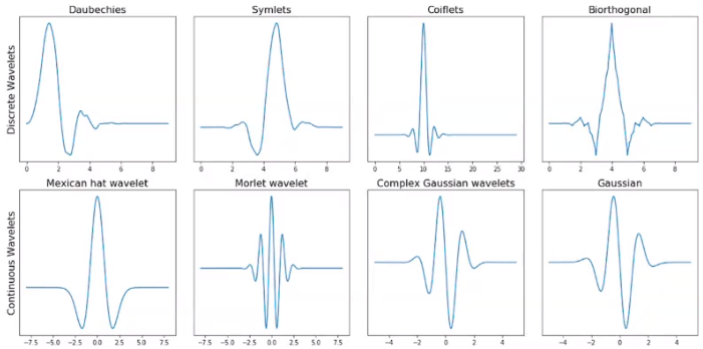
Split the signal with an orthonormal basis generate by translation and dilation of a mother wavelet

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

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.

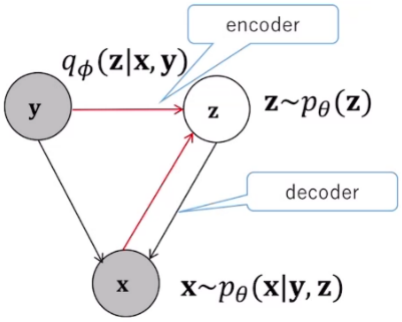
Inference How to query (predict with) a graphical model? Probability of unknown X given observations d, $P(X \mid d)$, the **most likely hypothesis** (parameters) X.

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.



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 distributions: simplifies marginalization and inference algorithms, allowing to **incorporate prior knowledge** concerning causal relationships and associations between random variables.

Directed graphical models (Bayesian Networks)

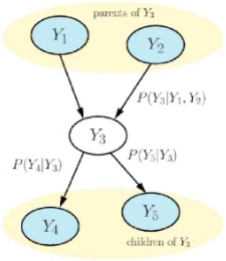
Undirected graphical models (Markov Random Fields)

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

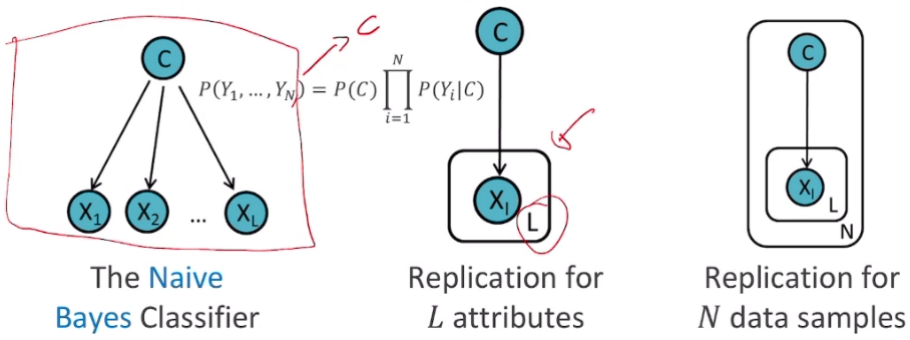


Conditional Probability Tables CPTs are local to each node and describe the probability distribution **given its parents**.

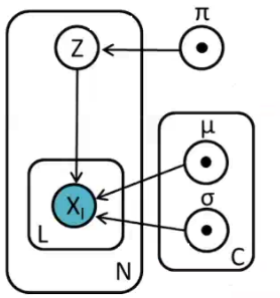
$$P(Y_1, \ldots, Y_n) = \prod_{i=1}^{N} P(Y_i \mid \text{Parents}(Y_i))$$

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

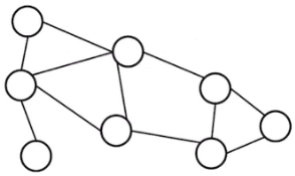
$$P(Y_1, \ldots, Y_N, C) = P(C) \prod_{i=1}^{N} P(Y_i \mid C)$$



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

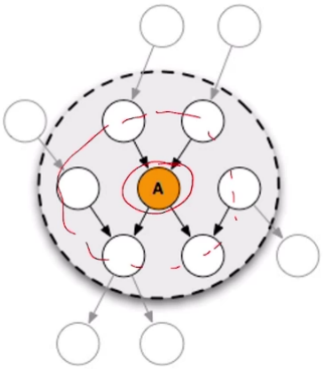
Can we reason on the structure of the graph to infer direct/indirect relationships between random variables?

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.

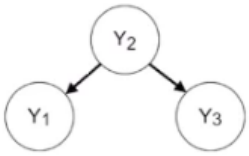
1. Pick a topological ordering of the nodes
2. Generate data by sampling from the local condition probabilities following this order

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.

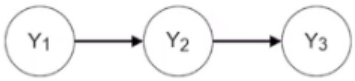
Tail to Tail Common cause



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

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)

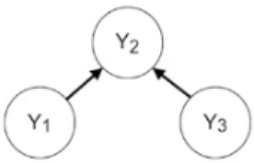
Head to Tail Causal Effect



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

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

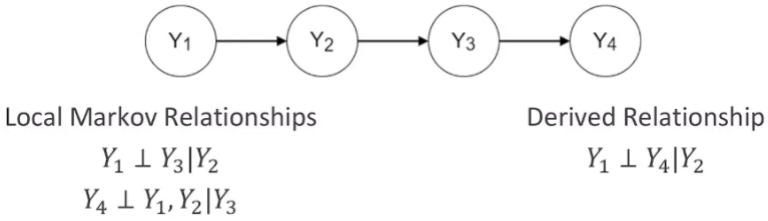


$$P(Y_1, Y_2, Y_3) = P(Y_1)P(Y_3)P(Y_2 \mid Y_1, Y_3)$$

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 $Y_i \rightarrow Y_c \rightarrow Y_j$ (or $Y_i \leftarrow Y_c \leftarrow Y_j$) and $Y_c \in Z$
- r contains a tail-to-tail $Y_i \leftarrow Y_c \rightarrow Y_j$ and $Y_c \in Z$
- r contains head-to-head $Y_i \rightarrow Y_c \leftarrow Y_j$ and neither Y_c nor its descendants are in Z

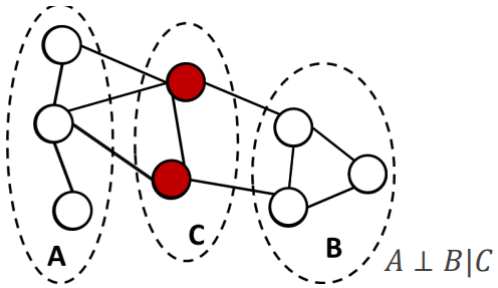
Two nodes Y_i, Y_j in a Bayesian Network G are d-separated by $Z \subset V \Leftrightarrow$ all undirected paths between Y_i, Y_j are d-separated by Z (denoted by $\text{Dsep}_G(Y_i, Y_j \mid Z)$)

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 $X = X_1, \ldots, X_n$ 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.

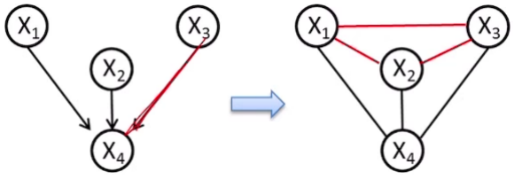
Boltzmann Distribution A convenient and widely used strictly positive representation of the potential function is

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

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

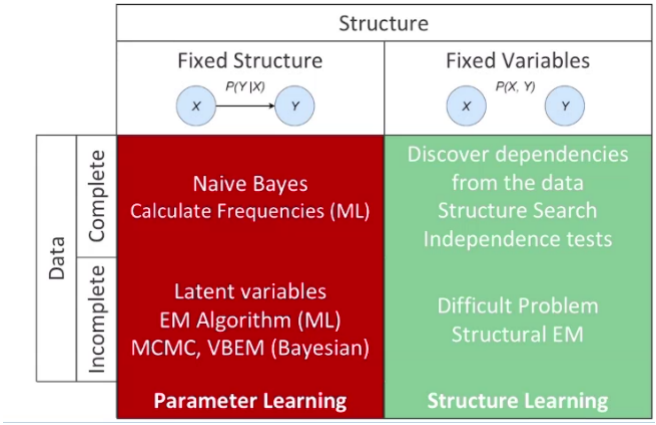
From Directed to Undirected

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 $Y = Y_1, \ldots, Y_N$, 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 the $t-1$ previous nodes.
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

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

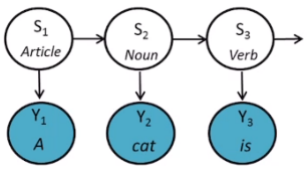
because $P(x_i \mid x_{i-1})$ is the same whenever the t.
$P(x_1)$ is the **prior distribution** (x_1 has nothing "before" it) and $P(x_i \mid x_{i-1})$ is the **transition distribution**.

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 $i = 1, \ldots, C$.
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

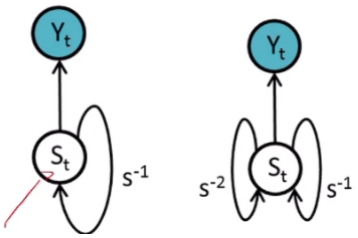
A state transition

B emission distribution (or its parameters)

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

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

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



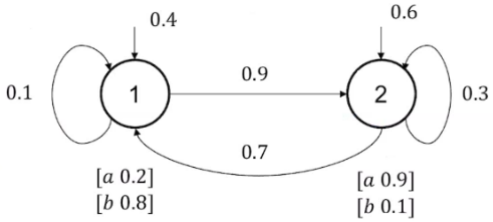
Indicates that the hidden state S_t at time t is dependent on context information from

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

$$\begin{array}{ccccccccccccc} S_1 & \to & \dots & \to & S_{t-1} & \to & S_t & \to & S_{t+1} & \to & \dots & \to & S_T \\ \downarrow & & \dots & & \downarrow & & \downarrow & & \downarrow & & \dots & & \downarrow \\ y_1 & \to & \dots & \to & y_{t-1} & \to & y_t & \to & y_{t+1} & \to & \dots & \to & y_T \end{array}$$

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 $\beta_t \equiv \mu_\beta(X_n)$

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

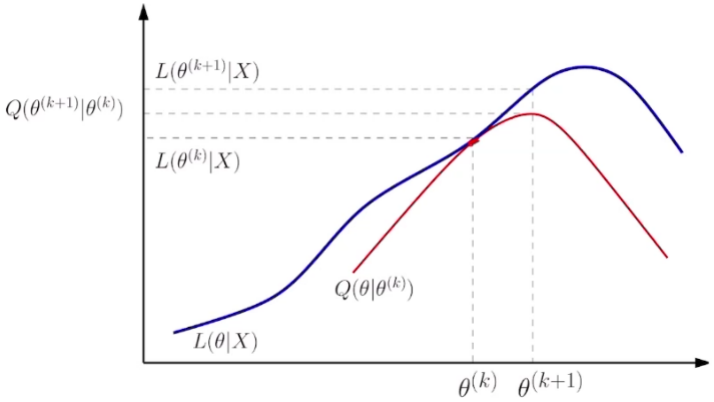
Learning in HMM

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

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

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 z_{ti}^n assuming estimated parameters $\Theta^t = (\pi^t, A^t, B^t)$ fixed at time t.
Expectation with respect to a discrete random variable is

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

To compute the conditional expectation $Q^{t+1}(\Theta \mid \Theta^t)$ for the complete HMM log likelihood we need to estimate

$$E_{Z \mid Y, \Theta^k}[z_{ti}] = P(S_t = i \mid y)$$
$$E_{Z \mid Y, \Theta^k}[z_{ti} z_{(t-1)j}] = P(S_t = i, S_{t-1} = j \mid y)$$

And we know how to compute the posteriors thanks to the forward-backward algorithm:

$$y_t(i) = P(S_t = i \mid Y) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{C} \alpha_t(j)\beta_t(j)}$$

$$y_{t,t-1}(i,j) = P(S_t = i, S_{t-1} = j \mid Y) = \frac{\alpha_{t-1}(j)A_{ij}b_i(y_t)\beta_t(i)}{\sum_{m,l=1}^{C} \alpha_{t-1}(m)A_{lm}b_j(y_t)\beta_t(l)}$$

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

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

Optimization problem, using the posteriors computed at the E-step. As usual with

$$\frac{\partial Q^{t+1}(\Theta \mid \Theta^t)}{\partial \Theta}$$

where $\Theta = (\pi, A, B)$ are now variables. The parameters can be distributions, so we need to preserve sum-to-one constraints (Lagrange Multipliers).
State distributions are

$$A_{ij} = \frac{\sum_{n=1}^{N}\sum_{t=2}^{T^n} \gamma_{t,t-1}^n(i,j)}{\sum_{n=1}^{N}\sum_{t=2}^{T^n} \gamma_{t-1}^n(j)}$$

$$\pi_i = \frac{\sum_{n=1}^{N} \gamma_1^n(i)}{N}$$

and the emission distribution, multinomial, is

$$B_{ki} = \frac{\sum_{n=1}^{N}\sum_{t=1}^{T^n} \gamma_t^n(i)\delta(y_t = k)}{\sum_{n=1}^{N}\sum_{t=1}^{T^n} \gamma_t^n(i)}$$

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 $\mathrm{s} = s_1^*, \ldots, s_T^*$ 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

$$s_t^* = \arg\max_{i=1,\ldots,C} P(S_t = i \mid Y)$$

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 $\prod$.

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

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

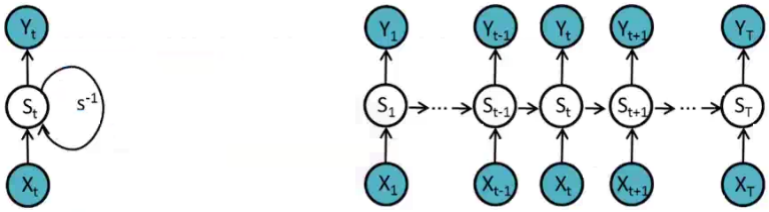
$$\epsilon(s_{t-1}) = \max_{s_t} P(y_t \mid s_t)P(s_t \mid s_{t-1})\epsilon(s_t)$$

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

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

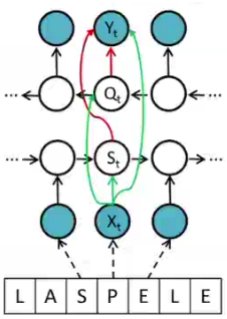
From state s_1 we pick up s_1^* and send it to s_2, which will use this information and pick the correct state that maximize the ϵ.

0.3.9 Input-output Hidden Markov Models

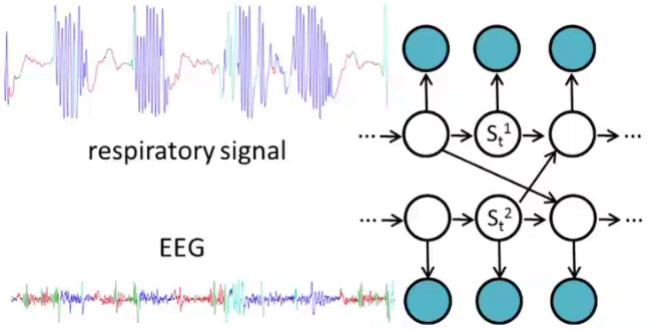


Translates an input sequence in an output sequence (**transduction**). State transition and emission depend on input observations (**input-driven**).
Recursive model highlights analogy with RNNs.

Bidirectional Input-Driven Models Removes the causality assumption that current observation doesn't depend on the future and homogeneity assumption that a state transition doesn't depend on the position in the sequence.



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, edges $e \in E$ are bi-directional dependencies between variables.

Likelihood Factorization Define $\mathrm{X} = X_1, \ldots, X_N$ 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, from a **partition function**:

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

As already stated, potential functions are not probabilities, they express which configurations of the local variables are preferred. A conveniently and widely used strictly positive representation of the potential function is the **Boltzmann Distribution**:

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

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

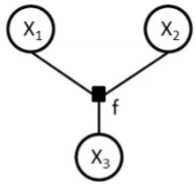
$$\psi_C(X_C) = \exp\left(-\sum_{k=1}^{K} \theta_{Ck} f_{Ck}(X_C)\right)$$

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

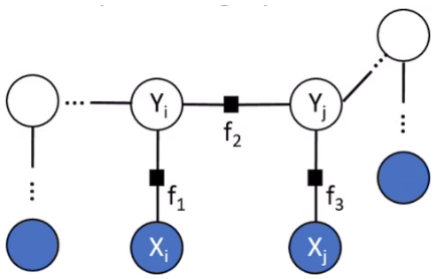
Instead of $P_\theta(x, y)$ we compute $P_\theta(y|x) \cdot P_\theta(x)$. Under this assumptio we can directly model the conditional distribution

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

We note that Z depends on X

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

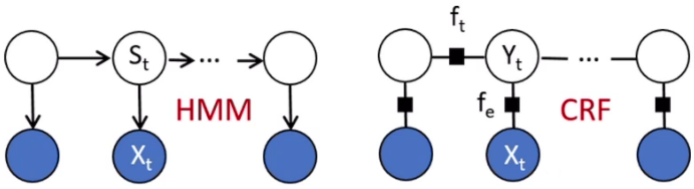
Conditional Random Field CRF are constrained MRF models representing input-conditional distributions.



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

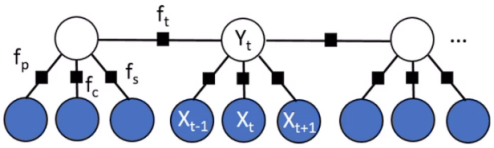
Discriminative Learning in Graphical Models X is always observable input while Y **can** be unobserved.
Let's consider a single Y and multiple Xs. We can observe the Y_n corresponding to X_n for some n, and we can use this information to fit θ in $P(Y \mid X, \theta)$

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.

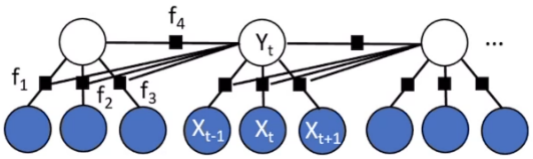
Generalization of HMM CRF are much more powerful



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.

$$P(Y \mid X, \Theta) = \frac{1}{Z(X)} \prod_t \exp\left(\Theta_p f_p(X_{t-1}, Y_t) + \Theta_c f_c(X_t, f_t) + \Theta_s f_s(X_{t+1}, Y_t) + \Theta_t f_t(Y_{t-1}, Y_t)\right)$$

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



The general Linear Conditional Random Fields likelihood is

$$P(Y \mid X, \Theta) = \frac{1}{Z(X)} \prod_t \exp\left(\Theta_k f_k(Y_t, Y_{t-1}, X_t)\right)$$

Uses indicator variables in f_k definition to include or disregard the influence of specific random variables: e.g. $I_{Y_t=i}, I_{X_t=o}$

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.

$$P(Y_t, Y_{t-1} \mid X) \simeq \alpha_{t-1}(Y_{t-1}) \psi_t(Y_t, Y_{t-1}, X_t) \beta_t(Y_t)$$

Clique weighting $\psi_t(Y_t, Y_{t-1}, X_t) = \exp(\Theta_c f_c(X_t, Y_t) + \Theta_t f_t(Y_{t-1}, Y_t))$

Forward message $\alpha_t(i) = \sum_j \psi_t(i, j, X_t) \alpha_{t-1}(j)$

Backward message $\beta_t(i) = \sum_i \psi_{t+1}(i, j, X_{t+1}) \beta_{t+1}(i)$

Also Viterbi can be used, 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 (sample y rather than estimate $P(y)$) or Variational Belief Propagation (reduce to message passing on trees).

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

$$L(\Theta) = \sum_n \sum_t \sum_k \Theta_k f_k(Y_t^n, Y_{t-1}^n, X_t^n) - \sum_n \log Z(X^n) \left(- \sum_k \frac{\Theta_k^2}{2\sigma^2} \right)$$

With the last being a regularization term based on $\|\Theta\|^2$.
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.

$$\frac{\partial L(\Theta)}{\partial \Theta_k} = \sum_{n,k} f_k(Y_t^n, Y_{t-1}^n, X_t^n) - \sum_{n,t} \sum_{y,y'} f_k(y, y', X_t^n) P(y, y' \mid X^n) - \frac{\Theta_k}{\sigma^2}$$

We have sum of expectations: the first term is $E[f_k]$ when Y is not random, with samples drawn from a finite dataset (**empirical distribution**), and the right term is $E[f_k]$ 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, meaning that when the gradient is zero these are equal.
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 $f_k(X_k, Y_k) = I_{y_k = \hat{y}_k} q(X_c)$, with f_k non-zero only for a specific output configuration $\hat{y}_k$, and then depending only on X_k (the features not shared by the classes). $q(X_c)$ is the observation functions: word begins with capital, ends with -ing...
In computer vision they can be used to define bi-dimensioal lattice on images, bg/fg segmentation and to impost constraints.

0.3.11 Bayesian Learning and Variational Inference

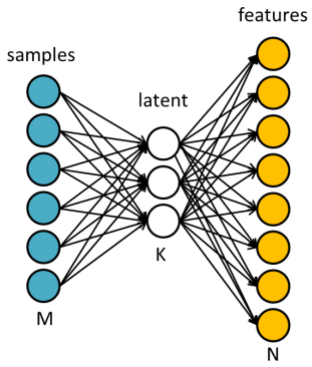
Introducing basic concepts of variational learning useful for both generative models and deep learning.

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 a graph like $Z \longrightarrow X$ so of nodes $\{Z, X\}$ and a directed arc (Z, 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 that latent variables conditional and marginal distribution are more tractable than the joint distribution $P(X)$.

Tractability Hidden variables can make intractable the posteriors. We need stuff that simplify those posteriors: Bayesian learning introduces priors which introduce integrals in the posteriors computation which are not always analytically or computationally tractable.

Kullback-Leiber Divergence

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

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

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)

The expectation E_q means taking all possible assignments z weighted according to the probability $q(z)$.

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

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

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

The log-likelihood for a model with a single hidden variable Z and parameters θ with a single sample assumed for simplicity is the following

$$\log P(x \mid \Theta) = \log \int_z P(x, z \mid \Theta) dz = \log \int_z \frac{Q(z \mid \phi)}{Q(z \mid \phi)} P(x, z \mid \Theta) dz$$

Q is a distribution, used over z with parameters ϕ and $Q(z \mid \phi) \neq 0$. That is the definition of expectation, we have $\int_z Q(z \mid \phi) \cdot \frac{1}{Q(z \mid \phi)} P(x, z \mid \Theta)$ which is $\sum_z q \cdot g(z)$. Using Jensen we have

$$\log P(x \mid \Theta) = \log E_Q\left[\frac{P(x,z)}{Q(z)}\right] \geq E_Q\left[\log \frac{P(x,z)}{Q(z)}\right] = E_Q[\log P(x,z)] - E_Q[\log Q(z)] = L(x, \Theta, \phi)$$

So we are lower bounding $\log P(x \mid \Theta)$ with the expected joint distribution minus the entropy. So we have a lower bound on something I want to maximize, so we can maximize the lower bound. Maximizing this term entails that

we're supported by the data (using the expectation of joint distribution $E_Q[\log P(x,z)]$).
How good is this lower bound? Meaning $\log P(x \mid \Theta) - L(x,\Theta,\phi) =$? We introduce $Q(z)$ by marginalization

$$\int_z Q(z)\log P(x) - \int_z Q(z)\log\frac{P(x,z)}{Q(z)} = \int_z Q(z)\log\frac{P(x)Q(z)}{P(x,z)} = E_Q\left[\log\frac{Q(z)}{P(z \mid x)}\right] = KL\left(Q(z \mid \phi)\|P(z \mid x,\Theta)\right)$$

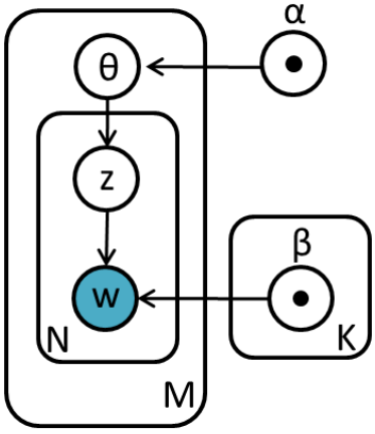
Because $\frac{P(x)}{P(x,z)} = \frac{1}{P(z \mid x)}$. So it's an optimization problem of finding Θ and ϕ that maximize $L(x,\Theta,\phi)$ reduced to a minimization problem with KL.
We can assume the existence of a probability $Q(z \mid \phi)$ which allows to bound the likelihood $P(x \mid \Theta)$ from below using $L(x,\Theta,\phi)$.
$L(x,\Theta,\phi)$ is called **variational bound** or **ELBO** (evidence lower bound). The optimal bound is obtained from $KL\left(Q(z \mid \phi)\|P(z \mid x,\Theta)\right) = 0$ choosing $Q(z \mid \phi) = P(z \mid x,\Theta)$. So the problem now becomes maximizing ELBO

$$\max_{\Theta,\phi}\sum_{n=1}^{N} L(x_n,\Theta,\phi)$$

where Θ are the model parameters and ϕ is used in $Q(z \mid \phi)$.
If $P(z \mid x,\Theta)$ is tractable then we use it s $Q(z \mid \phi)$ (optimal ELBO). Otherwise we choose $Q(z \mid \phi)$ as a tractable family of distributions: find ϕ that minimize $KL(Q(z \mid \phi)\|P(z \mid x,\Theta))$ or find ϕ that maximize $L(\ ,\phi)$.

Example Bag of Words representations are classical examples of multinomial data. A BOW dataset X is the $N \times M$ document matrix, with N number of vocabulary items w_j and M is the number of documents d_i and $x_{ij} = n(w_j,d_i)$ the number of occurrences of w_j in d_i.
Often M is very very large ($\simeq$ 30k elements). So we want a smaller representation. Mixture of topics: a topic identifies a pattern in the co-occurrence of multinomial items w_j within the documents. Mixture, so we associate an interpretation (topic) to each item in a document, whose interpretation is then a mixture of the items' topics. We use Latent Variables.

Latent Dirichlet Allocation LDA models a document as a mixture of topics z. We assign one topic z to each item w with probability $P(w \mid z,\beta)$ and pick a topic for the whole document with probability $P(z \mid \Theta)$



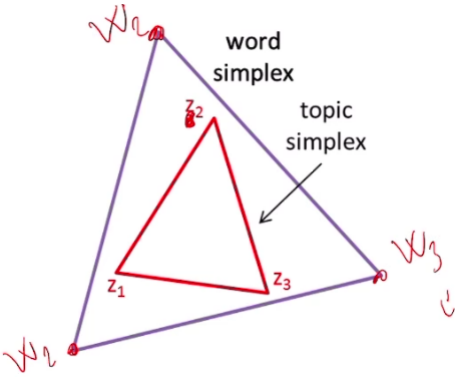
Each document has its personal topic proportion Θ sampled from a distribution. Θ defines a multinomial distribution, but it is a random variable as well. α is the prior.

Dirichlet distribution Conjugate prior to multinomial distribution.
If the likelihood is multinomial with a Dirichlet prior, then posterior is Dirichlet.

$$P(\Theta \mid \alpha) = \frac{\Gamma(\sum_{k=1}^{K}\alpha_k)}{\prod_{k=1}^{K}\Gamma(\alpha_k)}\prod_{k=1}^{K}\Theta_k^{\alpha_k-1}$$

Γ is the generalization of the factorial. With a big α we make the topics almost equiprobable, while with lower α we get substatially different proportions (less and less topics active). So big α means that every document can express each topic, while low α is almost deterministic. Usually, $\alpha = 1$ or similar but smaller.

LDA finds a set of K projection functions on the K-dimensional topic simplex.



LDA Generative Process For each of the M documents, we choose Θ with Dirichlet(α) and for each of the N items, we choose a topic z with Multinomial(Θ) and pick an item w_j with multinomial probability $P(w_j \mid z, \beta)$.
We get a multinomial topic-item parameter matrix $[\beta]_{k \times V}$

$$\beta_{kj} = P(w_j = 1 \mid z_k = 1) \text{ or } P(w_j = 1 \mid z = k)$$

$$P(\Theta, z, w \mid \alpha, \beta) = P(\Theta \mid \alpha) \prod_{j=1}^{N} P(z_j \mid \Theta) P(w_j \mid z_j, \beta)$$

It's a completed likelihood with the conditional independence assumption.

Learning Marginal distribution of a document $d = \mathrm{w}$

$$P(\mathrm{w} \mid \alpha, \beta) = \int \sum_{z} P(\Theta, z, w \mid \alpha, \beta) d\Theta = \int P(\Theta \mid \alpha) \prod_{j=1}^{N} \sum_{z_j=1}^{k} P(z_j \mid \Theta) P(w_j \mid z_j, \beta) d\Theta$$

given $\mathrm{w}_1, \ldots, \mathrm{w}_M$ find α, β that maximize.
Key problem is inferring latent variables posterior:

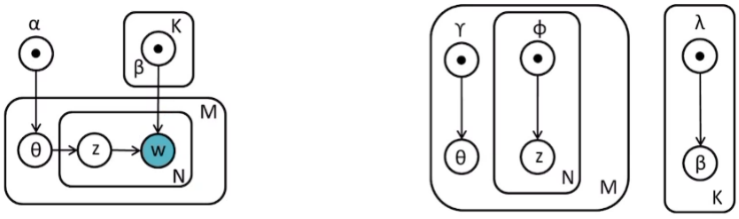
$$P(\Theta, \mathrm{z} \mid \mathrm{w}, \alpha, \beta) = \frac{P(\Theta, \mathrm{z}, \mathrm{w} \mid \alpha, \beta)}{P(\mathrm{w} \mid \alpha, \beta)}$$

but the denominator is intractable because of couplings between β and Θ under exponenziation in the summation over topics. So we don't use the posterior, we pick a function Q that helps in solving the problem (variational inference).

Variational Inference We write $Q(z|\phi)$ function that is sufficiently similar to the posterior but tractable. It should be such that β and Θ are no longer coupled, fitting ϕ so that is close to the posterior according to KL.
Fast convergence but it's an approximation.
The key idea is to assume that $Q(z \mid \phi)$ is tractable: **mean-field assumption**.

$$Q(z \mid \phi) = Q(z_1, \ldots, z_K \mid \phi) = \prod_{k=1}^{K} Q(z_k \mid \phi_k)$$

Can be generalized by factorizing on groups of latent variables. Does not contain the true posterior because hidden variables are dependent.
We optimize ELNO using $Q(z|\phi)$ using the factorized distribution. Simple **coordinate ascent inference**: iteratively optimize each variational distribution holding the others fixed, so when learning we use the model on the right, breakdown of the independence.



Given $\Phi = \{\gamma, \phi, \lambda\}$ as variational approximation parameters

$$Q(\theta, \boldsymbol{z}, \beta | \Phi) = Q(\theta|\gamma) \prod_{n=1}^{N} Q(z_n|\phi_n) \prod_{k=1}^{K} Q(\beta_k|\lambda_k)$$

Variational Expectation-Maximization Find the Φ, Ψ that maximize the ELBO

$$L(w, \Phi, \Psi) = E_Q[\log P(\Theta, z, w \mid \Psi)] - E_Q[\log Q(\Theta, z, \Psi \mid \Phi)]$$

by alternate maximization:

Fix Ψ and update variational parameters Ψ^* (**E-Step**)

Fix $\Phi = \Phi^*$ and update the model parameters Ψ^* (**M-Step**)

Repeat until little improvement on the likelihood

Unlike EM, variational EM has no guarantee to reach a local maximizer of L.

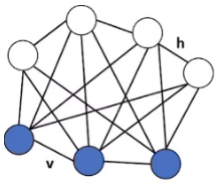
0.3.12 Boltzmann Machines

Examples of Markov Random Fields:

Visible random variables $v \in \{0, 1\}$

Latent random variables $h \in \{0, 1\}$

$s = [vh]$ (concatenation)



Linear energy function

$$E(s) = -\frac{1}{2}\sum_{ij} M_{ij} s_i s_j - \sum_j b_j s_j = -\frac{1}{2} s^T M s - b^T s$$

with symmetric and no self-recurrent connectivity. Model parameters $\Theta = \{M, b\}$ encode the interactions between the variables (observable and not).
Boltzmann machines are a type of Recurrent Neural Networks. They can be interpreted as stochastic neural network: the state of a unit at a given timestep is sampled from a given probability distribution and the network learns a probability distribution $P(V)$ from the training patterns. The network includes both visible v and hidden h units, and the activity is a sample from posterior probability given the inputs (visible data).

Stochastic Binary Neurons Binary output s_j at any time t. Typically discrete time model with time into small Δt intervals. At each time interval $t + 1 = t + \Delta t$ the neuron can emit a spike with probability $p_j^{(t)}$

$$s_j^{(t)} = \begin{cases} 1 & \text{with probability } p_j^{(t)} \\ 0 & \text{with probability } 1 - p_j^{(t)} \end{cases}$$

The key is in the definition of the spiking probability (local potential x_j

$$p_j^{(t)} \simeq \sigma(x_j^{(t)})$$

The Boltzmann machine has N neurons with binary activations s_j, a weight matrix $M = [M_{ij}]_{i,j} \in \{1, \dots, N\}$ and a bias vector $b = [b_j]_j \in \{1, \dots, N\}$
Local neuron potential x_j is the usual

$$x_j^{(t+1)} = \sum_{i=1}^{N} M_{ij} s_j^{(t)} + b_j$$

This assuming full connectivity with weight 0 if you want to ignore a neuron in the potential. A chosen neurpon fire with spiking probability which is a sigmoid

$$p_j^{(t+1)} = P(s_j^{(t+1)} = 1 \mid s^{(t)}) = \sigma(x_j^{(t+1)}) = \frac{1}{1 + e^{-x_j^{(t+1)}}}$$

Clearly has Markovian dynamics, $P(s^{t+1} \mid s^t)$

How does the model state (activation of all neurons) evolve in time?

Parallel Dynamics Assuming we can compute each activation in parallel every Δt, updating each random variable in parallel.

$$P(s^{t+1} \mid s^t) = \prod_{j=1}^{N} P(s_j^{t+1} \mid s^t) = T(s_j^{t+1} \mid s^t)$$

Yielding a Markov process for state update

$$P(s^{t+1} = s') = \sum_s T(s' \mid s) P(s^t = s)$$

Glauber Dynamics One neuron at random is chosen for update at each step. No fixed-point guarantees for s, but it has a stationary distribution for the network at equilibrium state when its connectivity is symmetric.

Boltzmann-Gibbs Distribution Undirected connectivity enforces detailed balance condition

$$P(s)T(s' \mid s) = P(s')T(s \mid s')$$

Ensures reversible transitions guaranteeing existence of equilibrium distribution (Boltzmann-Gibbs)

$$P_\infty(s) = \frac{e^{-E(s)}}{Z}$$

where $E(s)$ is the energy function and $Z = \sum_s e^{-E(s)}$ is the partition function.

Learning Boltzmann machines can be trained so that the equilibrium distribution tends towards any arbitrary distribution across binary vectors given samples from that distribution. Basically, you can represent any distribution of binary variables.
Couple of simplifications:

bias b is just another row in the weight matrix M

consider only visible random variables, meaning $s = v$

We use probabilistic learning techniques to fit the parameters, i.e. maximizing the log-likelihood

$$L(M) = \frac{1}{L} \sum_{l=1}^{L} \log P(v_l \mid M)$$

given the P visible training patterns v_l, the set of all the visible units (so it's a joint distribution $v_{l1}, \ldots, v_{ln}$. We need a way to write the likelihood $P(v_l \mid M)$, and we can use the Boltzmann-Gibbs. So we can optimize it solving a maximum likelihood problem.

$$\log P(v_l \mid M) = \log \frac{e^{-E(v)}}{Z} = -E(v) - \log Z$$

$$\frac{\partial L}{\partial M_{ij}} \Rightarrow \frac{\partial(-E(v) - \log Z)}{\partial M_{ij}} \Leftrightarrow$$

Given that $E(v) = -\frac{1}{2} v^T M^T v = -\sum_{ij} M_{ij} v_1 v_j$ we get that every M_{ij} will be constant except for the M_{ij} we are differentiating leaving with $v_i v_j$. As for $\log Z$, differentiating the potential function leaves us with $\sum_v P(v \mid M) v_i v_j$ from $\partial \log \partial \exp \cdot \partial E$

$$\Leftrightarrow v_i v_j - \sum_v P(v \mid M) v_i v_j = 0$$

The second term is $\sum_v P(v \mid M) v_i v_j = E_{v_i v_j \in P(v \mid M)}[v_i v_j] = \langle v_i v_j \rangle_M$ the expectation of the coactivation of two units v_i, v_j when the values of those two units are taken from $P(v \mid M)$ the distribution learned by the model M.
So for a single l:

$$v_i^l v_j^l - \langle v_i v_j \rangle_M$$

not introducing l in the second term because it's marginalized, all the possible values in all possible configurations, it's an expectation and the data is already included in the formulation.
So $\forall\, v^l \in L$

$$\frac{\partial L}{\partial M_{ij}} = \frac{1}{L} \sum_{l=1}^{L} (v_i^l v_j^l) - \langle v_i v_j \rangle_M =$$

We have the clamped expectation under the empirical distribution $\frac{1}{L}\sum_{l=1}^{L}(v_i^l v_j^l) = E_{v_i^l v_j^l \in L}[v_i^l v_j^l] = \langle v_i v_j \rangle_c$

$$= \langle v_i v_j \rangle_c - \langle v_i v_j \rangle_M = \Delta M_{ij}$$

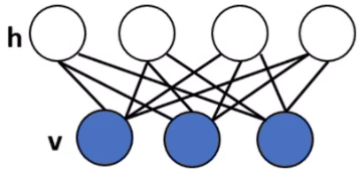
So we're focusing on the neurons i, j and the link between them M_{ij}, and we increase the connection when both are 1 so when $\langle v_i v_j \rangle_c$ is larger, so Hebbian learning. The second term $\langle v_i v_j \rangle_M$, if they are in disagreement then they are flipped then it's zero, if instead they are coactive (both on or off) then it's close to -1. It's anti-Hebbian, has the purpose of nearing the expectation of the model towards the reality.
When $\langle v_i v_j \rangle_c - \langle v_i v_j \rangle_M = 0 - 1$, we get that the model thinks that they are correlated while the are different, so the model stop believing that they are correlated.

With hidden variables doesn't change much. We have $s = [hv]$ The wake hebbian term $\sum_h s_i s_j P(h \mid v)$ and the dream anti-hebbian term $\sum_s s_i s_j P(s)$

$$\frac{\partial P(v \mid M)}{\partial M_{ij}} = \sum_h s_i s_j P(h \mid v) - \sum_s s_i s_j P(s) = \langle s_i s_j \rangle_c - \langle s_i s_j \rangle_M = \Delta M_{ij}$$

Again intractable. So we restrict.

Restricted Boltzmann Machines RBM are special Boltzman machines: bipartite graphs and connections only between hidden and visible units, not with "themselves".



This becomes tractable because we can compute the activation of each hidden in parallel and then compute the visible. The energy function is a specialization that higlights the bipartition in hidden and visible units

$$E(v, h) = -v^T M h - b^T v - c^T h$$

Hidden units are conditionally independent given visible units and viceverse

$$P(h_j \mid v) = \sigma\left(\sum_i M_{ij} v_i + c_j\right)$$

$$P(c_i \mid v) = \sigma\left(\sum_j M_{ij} h_j + b_i\right)$$

Training is again based on the likelihood maximization

$$\frac{\partial L}{\partial M_{ij}} = \langle v_i h_j \rangle_c - \langle v_j h_j \rangle = \Delta M_{ij}$$

Again, we have data $-$ model which are both expectations that need to be estimated. The first has the sum on just h, the second is a full summation over both v and h.
With a Gibbs sampling approach, for the wake/data term

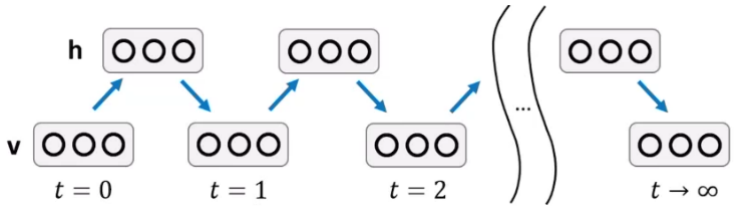
clamp data on v

sample v_i, h_j for all pairs of connected units

Repeat for all elements of dataset

as to stay as much as possible to the empirical data, and for the model/dream term

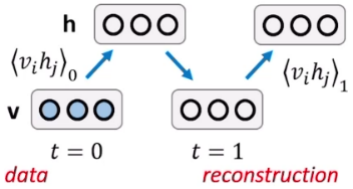
computing a correlation between ideally v^∞, h^∞, but of course can't wait until the infinity sample and we cut at some sample k so v^k, h^k.



We start with a training vector on the visible units, alternating between updating all the hidden units in parallel and updating all the visibile units in parallel

$$\frac{\partial L}{\partial M_{ij}} = \langle v_i h_i \rangle_0 - \langle v_j h_j \rangle_\infty = \text{data} - \text{model}$$

Contrastive-Divergence Learning Because Gibbs sampling converges really slowly: clamp a training vector v^l on visible units. CD-1.



Clamping a training vector v^l on visible units, update all hidden units in parallel, update all the visible units in parallel to get a **reconstruction** and update the hidden units again.

$$\underset{\text{data}}{\langle v_i h_j \rangle_0} - \underset{\text{reconstruction}}{\langle v_i h_j \rangle_1}$$

Learns a very crude approximation of the gradient of log-likelihood, not even following it closely. Why use it? Because in practice it works well.

0.3.13 Wrap Up

We've seen three families of models

> **Bayesian Networks**: unsupervised data understanding, interpretability but weak on supervised performance. Directed.
>
> **Markov Random Fields**: undirected version of BN, powerful when knowledge and constraints can be expressed with dictionaries. CRF: supervised way to be generative.
> But computationally heavy.
>
> **Dynamic Models**: topology unfolds on data structure, structured data processing but complex causal relationships.

Tractability Generative models to be used when:

> Need interpretability
>
> Need to incorporate prior knowledge
>
> Unsupervised learning or learning with partially observable supervision
>
> Need reusable/portable knowledge

To be avoided when:

> Having tight computational constraints
>
> Dealing with raw, noisy, low-level data

Variational inference and sampling are efficient ways to learn approximations to intractable distribution. Neural networks can be used as variational functions or to implement sampling processes.

0.4 Sampling Methods

Probability Recap

Sampling Drawing a set of realisations $X = \{x_1, \dots, x_L\}$ of a random variable x with a distribution $p(x)$. The set contains L samples.
If we have a sampling set we can use it to approximate $p(x)$

$$p(x) \simeq \frac{1}{L} \sum_{l=1}^{L} I[X_l = i]$$

with $I[c] = 1 \Leftrightarrow c$ is true.
We can also approximate the expectation of a function f

$$E_{p(x)}[f(x)] \simeq \frac{1}{L} \sum_{l=1}^{L} f(x_l)$$

We need sampling when $p(x)$ is intractable. For example, in Bayesian models the parameters are random variables but the posteriors are often intractable so we can use sampling to obtain the model parameters.

Sampling Procedure as Distributions A sampler S is a procedure that generates a sample set X from a generic distribution $p(x)$. Since X contains realizations of random variables, also X has a probability distribution. We denote with $\hat{p}_S(X)$ the probability to obtain the sample set X from the sampler S.
A **sampler** and its properties **are fully defined by its distribution** $\hat{p}_S(X)$. In general

$$\hat{p}(X) \neq p(x)$$

$p(x)$ is the distribution we would like to sample from, usually intractable

$\hat{p}(X)$ is the distribution over the samples set and defined by the sampling procedure

Let us consider the sampling approximation of the expectation

$$E_{p(x)}[f(x)] \simeq \frac{1}{L} \sum_{l=1}^{L} f(x_l) = \hat{f}_X$$

Since $\hat{f}_X$ estimates a value, we could ask:

Is $\hat{f}_X$ an unbiased estimator?

How much is the variance of the approximation?

An unbiased estimator $\hat{\Theta}$ of the unknown $\Theta \Rightarrow$ the approximation is exact on average. Meaning: let $\hat{p}(X)$ the distribution over all possible realizations of the sampling set X, then $\hat{f}_X$ is unbiased estimator if

$$E_{\hat{p}(X)}[\hat{f}_X] = E_{p(x)}[f(x)]$$

This is true provided that $\hat{p}(x_l) = p(x_l)$
A sampler with this property is called **valid** because it draws samples from the desired distribution.

The variance of $\hat{f}(X)$ tells us how much we can rely on the approximation computed using the sampling set X. Let

$$\Delta \hat{f}_X = \hat{f}_X - E_{\hat{p}(X)}[\hat{f}_X]$$

then the variance of $\hat{f}(X)$ is given by

$$E_{\hat{p}_X}[(\Delta \hat{f}_X)^2]$$

we would like low variance meaning that $\hat{f}(X)$ is quite always close to the average, i.e. to $E_{p(X)}[f(X)]$.

So if the sampler has same marginals and draws sample independently we obtain

$$E_{\hat{p}_X}[(\Delta \hat{f}_X)^2] = \frac{1}{L} \mathrm{Var}_{p(x)}[f(x)]$$

The variance reduces linearly with respect to the number of samples provided that $\mathrm{Var}_{p(x)}[f(x)]$ is finite.

We've shown that we need sampling to approximate expectations and to do inference in Bayesian models. Propertis of the sampling procedure depends on $\hat{p}(X)$: valid sampler and low approximation variance.

0.4.1 Univariate Sampling

Easy: random number generator R which produces a value uniformly at random in $[0, 1]$ and $p(x)$. We use $p(x)$ to divide $[0, 1]$ in bins and sample accordingly.

0.4.2 Multivariate Sampling

In the multivariate case $p(x)$ represents a joint distribution over a set of discrete variables $\{s_1, \dots, s_n\}$ each with C states. Hence each sample $x_i \in X$ contains n values.
We build a univariate distribution $p(S)$ where S is a discrete variables with C^n states (all possible combinations) and we can sample from $p(S)$ using the univariate schema.

S	s_1	s_2	s_3	s_4	s_5	$p(S)$
1	1	1	1	1	1	$p(1,1,1,1,1)$
2	1	1	1	1	2	$p(1,1,1,1,2)$
3	1	1	1	1	3	$p(1,1,1,1,3)$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
C^n	C	C	C	C	C	$p(C,C,C,C,C)$

The problem is the number of possible states C^n, so computationally unfeasible.
Using the chain rule we can rewrite the joint distribution as a chain of conditional distributions. Then we sample in order:

$\hat{s}_1 \simeq p(s_1)$

$\hat{s}_2 \simeq p(s_2 \mid \hat{s}_1)$

$\dots$

$\hat{s}_n \simeq p(s_n \mid \hat{s}_1, \dots, \hat{s}_{n-1})$

Each is univariate, so easy, but unfortunately computing the distribution $p(s_i \mid s_{j<i})$ often requires summation over an exponential number of states.
This approach is called **Ancestral Sampling**. There are cases where it cannot be used.

Example We can apply directly if a distribution $p(s_1, \dots, s_n)$ is already presented as a belief network:

The BN ancestral order tells us the sampling order

$$\{s_1, s_2, s_4\} \prec \{s_3\} \prec \{s_6\} \prec \{s_5\}$$

Sample $\hat{s}_1 \simeq p(s_1)$

Sample $\hat{s}_4 \simeq p(s_4)$

Sample $\hat{s}_2 \simeq p(s_2)$

Sample $\hat{s}_3 \simeq p(s_3)$

Sample $\hat{s}_6 \simeq p(s_6)$

Sample $\hat{s}_5 \simeq p(s_5)$

We obtain a single sample $x^l = \hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{s}_4, \hat{s}_5, \hat{s}_6$.

Suppose a subset s_ϵ of variables that are visible, with $s = s_\epsilon \cup s_{\setminus\epsilon}$ we want to sample from

$$P(s_{\setminus\epsilon} \mid s_\epsilon) = \frac{p(s_{\setminus\epsilon}, s_\epsilon)}{p(s_\epsilon)}$$

Can we still use Ancestral Sampling?

Clamping variables changes the structure of the bayesian network, and computing the new structure is as complex as running exact inference.

We could run AS on the original structure and discarding any samples which do not match the evidence.

Sampling under evidence is very important, in probabilistic models the inference is based on the posterior which is exactly that. So we need an efficient method of sampling under evidence.

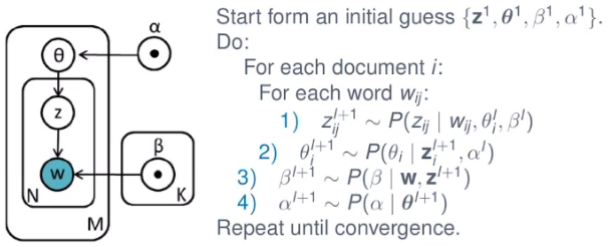
Gibbs Sampling Procedure The idea is to start from a sample $x^1 = \{s_1^1, \dots, s_n^1\}$ and **update only one variable at a time**. Dealing with evidence is easy, we just do not select the visible variables.
During the $(l+1)$th iteration:

Select a variable s_j

we sample the new value according to

$$s_j^{l+1} \simeq p(s_j \mid s_{\setminus j}) = \frac{1}{Z} p(s_j \mid \text{Parents}(s_j)) \prod_{k \in \text{Children}(s_j)} P(s_k \mid \text{Parents}(s_k))$$

Which depends only on the Markov blanket of s_j and $s_{\setminus j}$ is clamped to$\{s_1^l, \dots, s_{j-1}^l, s_{j+1}^l, \dots, s_n^l\}$



$\hat{p}(X)$ **of Gibbs sampling** Considering a set of variables $X = \{x^1, \dots, x^L\}$ obtained with a Gibbs sampler: each sample x^{l+1} is obtained from the previous sample x^l. We can define a probability $q(x^{l+1} \mid x^l)$, the probability to obtain x^{l+1} given x^l:

$$q(x^{l+1} \mid x^l) = \sum_{j=1}^{N} P(\text{Variable } j \text{ is selected}) P(\text{New value } s_j^l \text{ given } x^l)$$

With $P(\text{New value } s_j^l \text{ given } x^l)$ from equation 4 and $P(\text{Variable } j \text{ is selected})$ up to you.
With this we can compute $\hat{p}(X)$

$$\hat{p}(X) = \prod_{l=1}^{L} q(x^{l+1} \mid x^l)$$

Markov Chain Monte Carlo Sampling Framework Gibbs sampling is a specialization of the Markov Chain Monte Carlo sampling framework. The idea is to build a Markov Chain whose stationary distribution is $p(x)$.
Let $q(x^{l+1} \mid x^l)$ be the Markov Chain state-transition distribution, if the Markov Chain is

Irreducible, meaning it's possible to reach any state from anywhere

Aperiodic, meaning at each time-step we can be anywhere

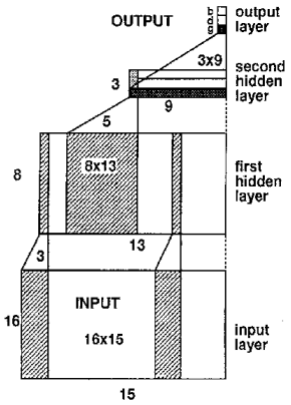
then it has a unique stationary distribution. So we define $q(x^{l+1} \mid x^l)$ such that the Markov Chain converges to $p(x)$.

0.5 Convolutional Neural Networks

Deep Learning Effectively is learning the representation of data.

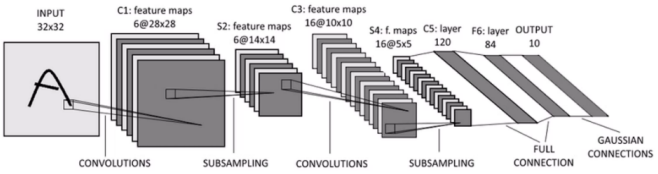
CNNs The principle is complex neurons that take the outputs of simple neurons and assemble it into more complex structures.

For sequences:

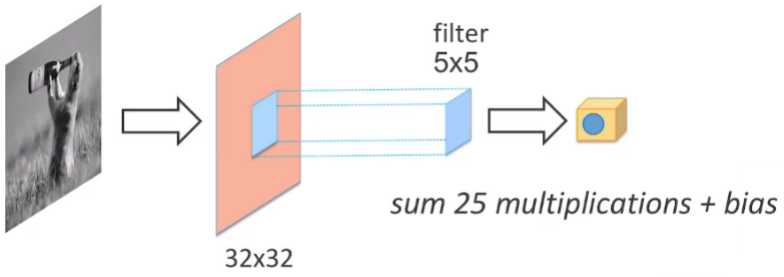


Apply a bank of 16 convolution kernels to sequences (window of 15 elements) trained by backpropagation with parameter sharing.

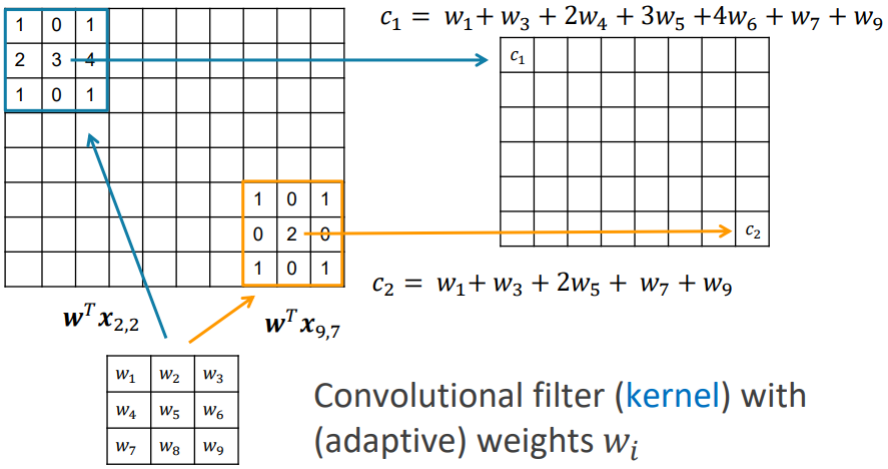
For images:

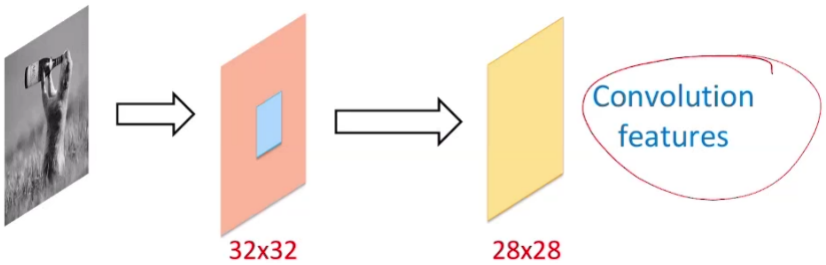


Dense Vector Multiplication E.g. take the 32x32 image and reshape into a vector of 3072. Input-sized weight vector for each hidden neuron W of e.g. 100 neurons: 100×3072 weights, a huge amount of parameters. That's why we want convolutional filters.



A set of weights that are applied to a 5×5 region of the image. It's a **kernel** that outputs into the **feature map**. No longer a 1:1 correspondence between a pixel and a parameter.





With a number of slices equal to the number of image channels. Basically a filter per channel color, but they are convolved together still giving a 28×28 feature map even with 3 color channels for example. The **convolution map stays bidimensional**.

Stride Basic convolution slides the filter on the image one pixel at the time (stride = 1). The slide value is an hyperparameter. The stride reduces the number of multiplication, sumbsampling the image. With a stride of 2 for example we would end up with a 14×14 map.

Activation Map Size What is the size of the image after applying a filter with a given stride S and size $K \times K$, on a image $W \times H$?
For example, a 3×3 filter with stride 2, $K = 3, S = 2$ on a 7×7 image: output image is 3×3.

$$W' = \frac{W - K}{S} + 1$$

$$H' = \frac{H - K}{S} + 1$$

Zero Padding Add columns and rows of zeros to the border of image to handle border pixels, padding of P rows/columns.

$$W' = \frac{W - K + 2P}{S} + 1$$

Zero padding is also used to retain the original size of the image

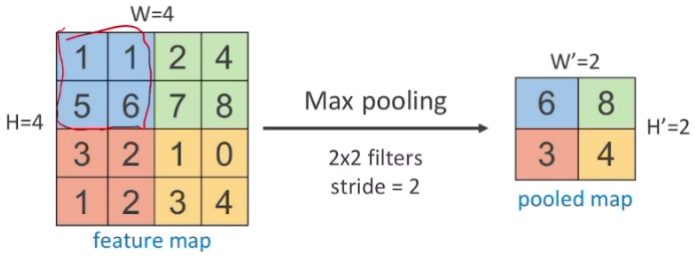
$$P = \frac{K - 1}{2}$$

Feature Map Transformation Convolution is a linear operator, applying an element-wise nonlinearity we obtain a transformed feature map.

$$\max(0, w^T x_{ij} + b)$$

The above is ReLU, used because of the simplicity of computing the gradient.

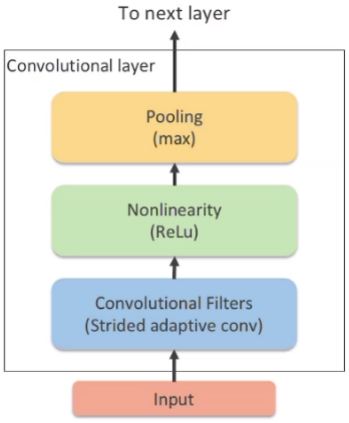
Pooling Operates on the feature map to make the representation: smaller (subsampling) and robust to some transformations.
Very simple:



Max pooling is the one used more frequently, but there's more: average pooling, L2-norm pooling, even random pooling.
It's uncommon to use zero padding with pooling.

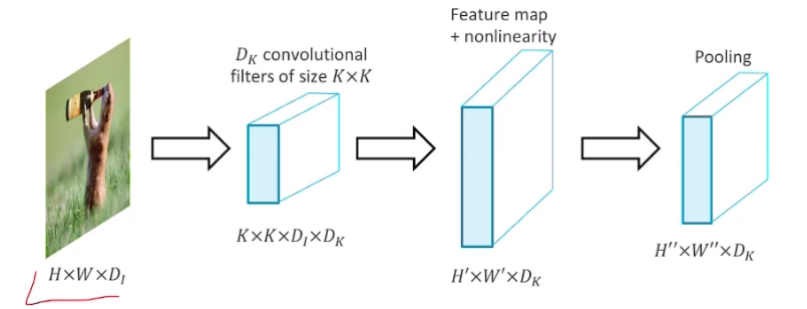
$$W' = \frac{W - K}{S} + 1$$

Convolutional Architecture



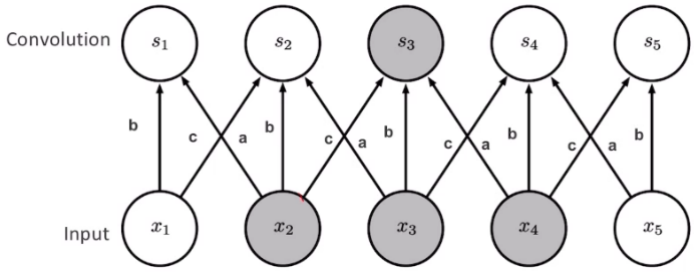
Usually use a certain number of convolutional layers ending in a small final image feeded into some fully connected layers.

Filter Banks



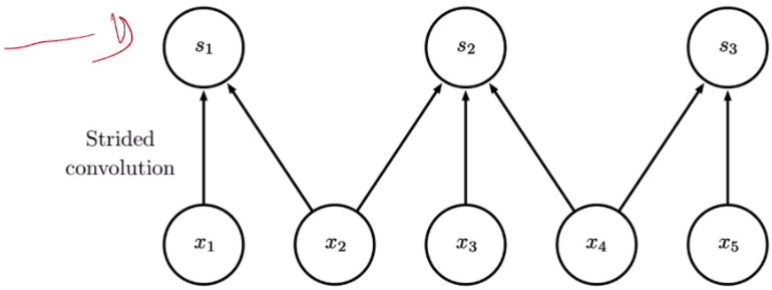
The number of parameters in this model is $K \times K \times D_I \times D_K$ because pooling is non parametric and feature maps have no parameters, so the parameters are just in the filters.

CNNs as Sparse Neural Networks Let's use a 1D input sequence to ease graphics.

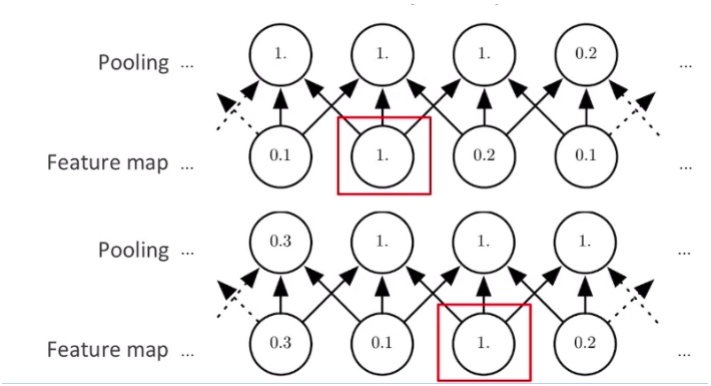


Convolution amounts to sparse connectivity (reduce parameters) with parameter sharing (enforces variance).
s_i is a convolutional neuron with a kernel size 3×1

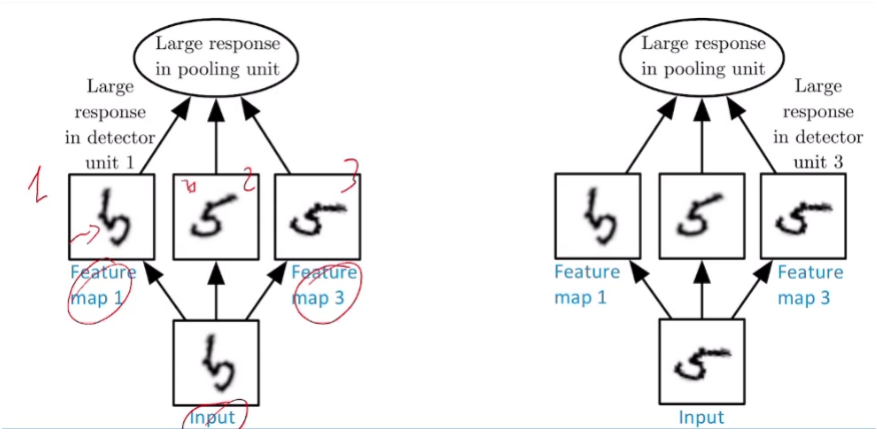
Strided Convolution A neuron after every stride, making connectivity sparser.



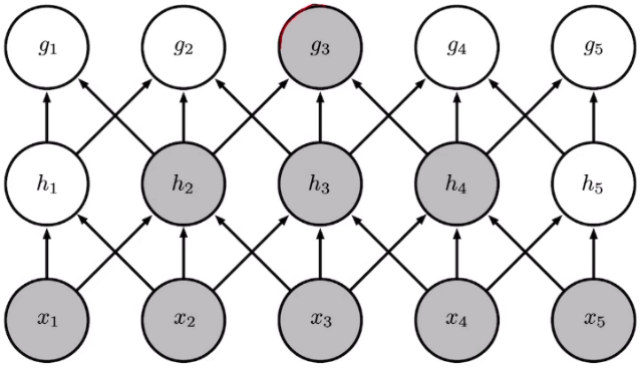
Pooling Just another neuron.



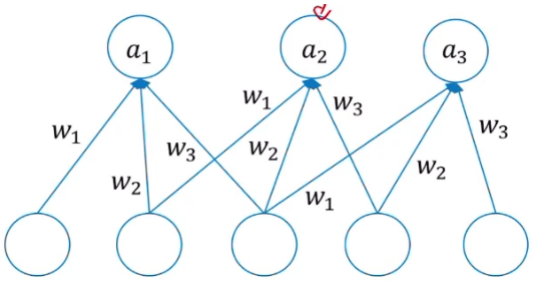
Cross-Channel Pooling and Spatial Invariance



Hierarchical Feature Organization

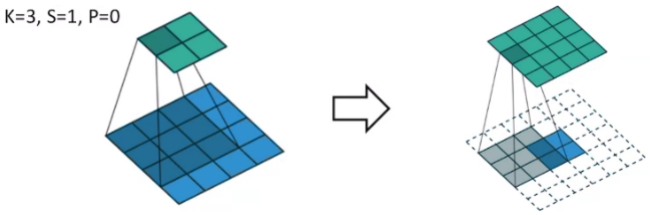


CNN Training Training by variants of the standard backpropagation that account for the fact that connections share weights (convolution parameters).



The gradient Δw_i is obtained by summing the contributions from all the connections sharing that weight.
We can write convolution as dense multiplication with shared weights.

Deconvolution



We can obtain the transposed convolution using the same logic of the forward convolution. If you had no padding in the forward convolution you need to use much padding when doing the transposed convolution.

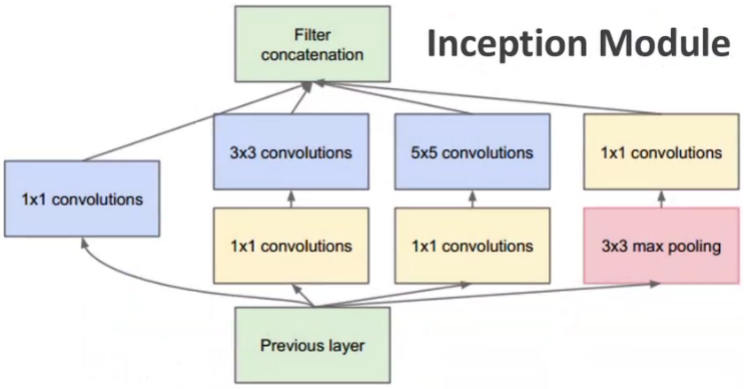
ReLU ReLU helps counteract gradient vanish: sigmoid first derivative vanishes as we increase or decrease z, ReLU first derivative is 1 when unit is active and 0 otherwise. ReLU second derivative is 0 (no second order effects).
Easy to compute and favors sparsity.

VGGNet Standardized convolutional layer:

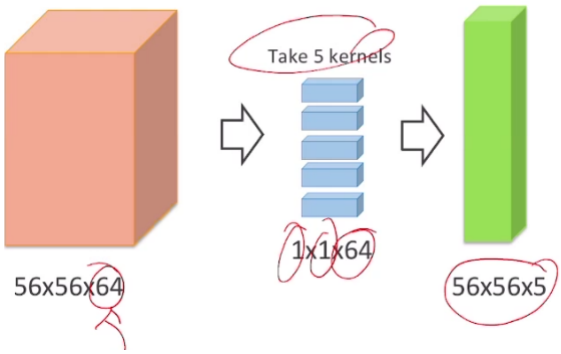
- 3×3 convolutions of stride 2
- 2×2 convolutions of stride 2
- 16 convolutional layers with 3 fully connected layers

Not very good because very limited, need to push for diversity.

GoogLeNet With the Inception module, based on the idea of diversifying the convolution.



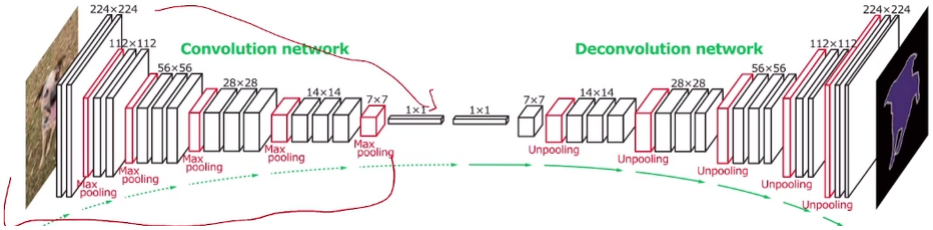
But many parameters: number of channels times number of convolutional layers times the dimensions. If not careful the application of this module explodes the number of parameters.
Kernels of different size to capture details at varied scale, aggregated before sending to next layer.
1×1 convolutions are helpful.



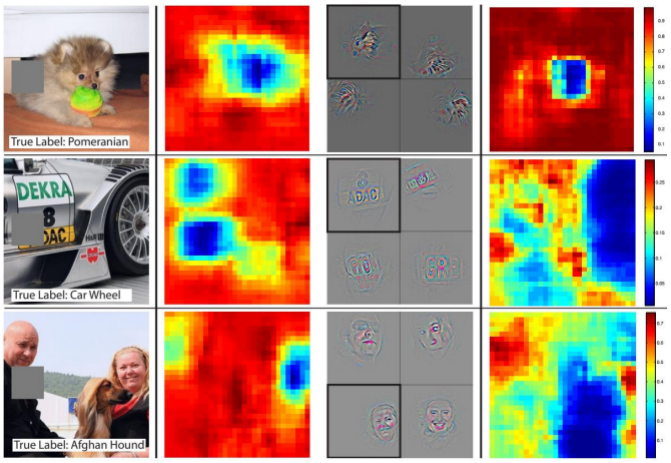
Only 5M parameters, followed by v2, v3 and v4 which added more filter factorization and introduced heavy use of Batch normalization.

Batch Normalization Very deep neural networks are subject to internal covariate shift. Distribution of inputs to a layer N may vary (shift) with different minibatches due to adjustments at layer $N-1$. Layer N can get confused by this, so (simplifying) we normalize for mean and variance in each minibatch.

Deconvolution Network Attach a DeConvNet to a target layer, pluggin an input and forward porpagation activations until layers, then backpropagate on the DeConvNet and see what parts of the reconstructed image are affected.

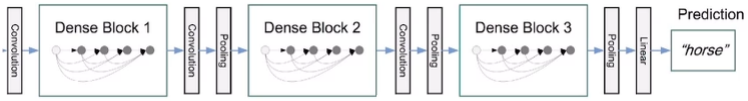


Occlusions Measure what happens to feature maps and object classification if we occlude part of the image. Slide a grey mask across the image and project back the response of the best filters using deconvolution.



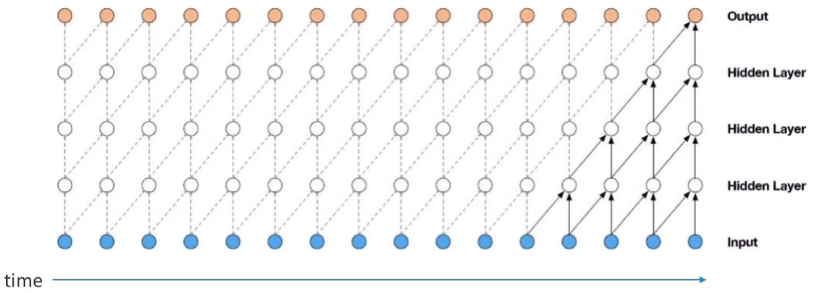
Zeiler&Fergus, Visualizing and Understanding Convolutional Networks, ICML 2013

Dense CNN



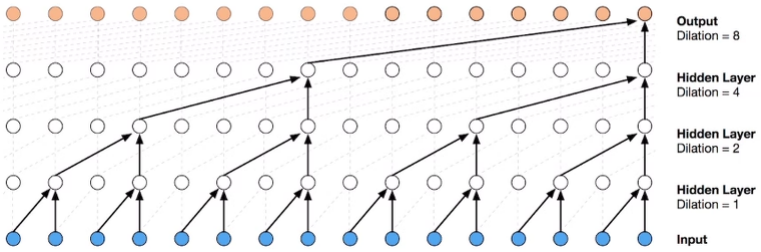
Gradient flows well in bypass connections, and each layer in the dense blocks has access to all the information from previous layers.

Causal Convolutions Basically meaning asymmetrical, to prevent a convolution to see into the future.



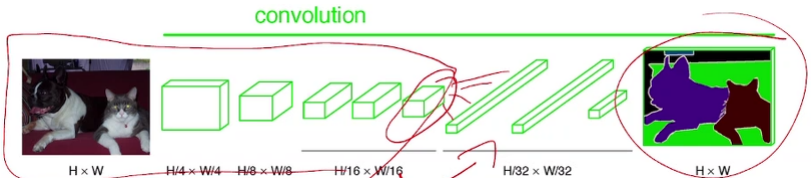
Problem is the context size glows slow with depth. So Dilated Convokutions

$$(I * K)(i,j) = \sum_m \sum_n I(i - lm, i - ln) K(m,n)$$

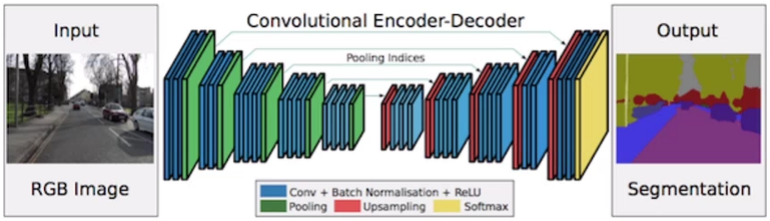


Basically convoluting every now and then. Similar to striding but size is preserved.

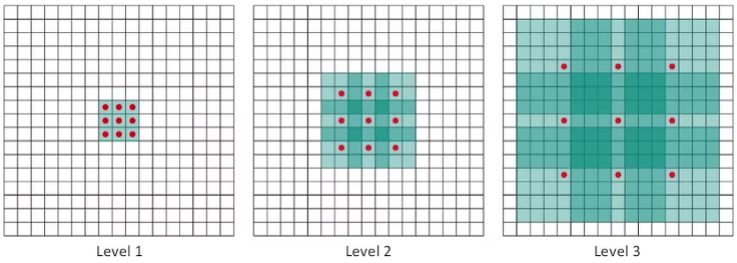
Fully Convolutional Networks Due to not maintaining the original dimension we cannot perform semantic segmentation, so we need **Fully Convolutional Networks**



Deconvolutional Architecture Basically you decompress the segmentation.



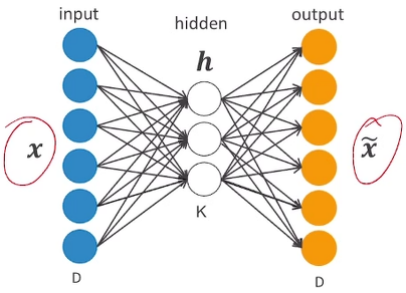
With Dilated Convolution out of the temporal domain we can perform semantic segmentation efficiently. For example with 3×3 convolutions with no pooling at each level:



0.6 Autoencoders

The first and the latest deep learning model.

Basic Autoencoder Train a model to reconstruct the input, passing through some form of **information bottleneck**.



h is known as **latent space projection**, like with probabilistic models.
The architecture is composed of the encoder f $(x \to h)$, and the decoder g $(h \to \tilde{x})$. In principle $K << D$, the bottleneck. Or you can penalize the model by making h sparsely active.

Neural Autoencoders Generally we want to train nonlinear AEs with possibly $K > D$ that do not learn trivial identity. Regularized AE:

Sparse Autoencoder Add a term to the cost function to penalize h (want the number of active units to be small)

$$J_{SAE}(\Theta) = \sum_{x \in S}(L(x, \tilde{x}) + \lambda \Omega(h))$$

Typically norm 1

$$\Omega(h) = \Omega(f(x)) = \sum_j |h_j(x)|$$

Probabilistic interpretation: training with regularization is MAP inference

$$\max \log P(h, x) = \max(\log P(x \mid h) + \log P(h))$$

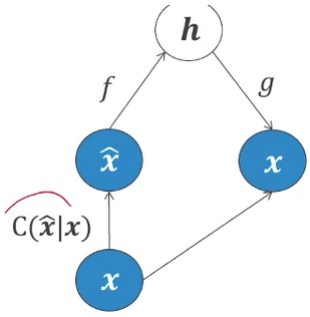
MAP is formed by the likelihood plus the prior $P(h)$

$$P(h) = \frac{\lambda}{2} \exp\left(-\frac{\lambda}{2}|h|_1\right) \Leftrightarrow \log P(h) = \lambda |h|_1 = \Omega(h)$$

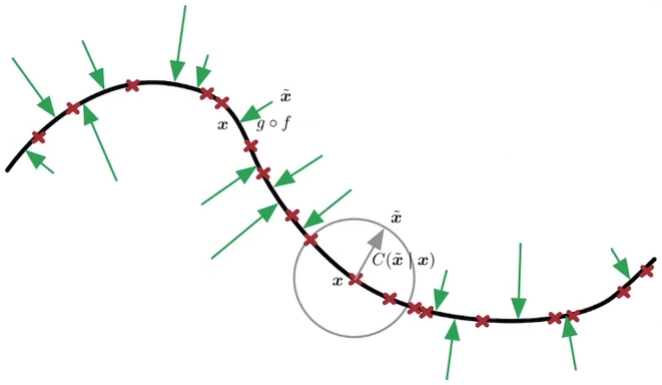
Denoising Autoencoders Train the AE to minimize the function

$$L(x, g(f(\hat{x})))$$

where $\hat{x}$ is a version of the original input corrupted by some noise process $C(\hat{x} \mid x)$, so the usual $\hat{x} = x + \epsilon$ with ϵ from a gaussian of mean 0 and variance 1.
Key intuition: learned representations should be robust to partial destruction of the input.
Probabilistic interpretation:



Learns the denoising distribution $P(x \mid \hat{x})$ by minizing $-\log P_d(x \mid h = f(\tilde{x}))$.
Manifold Learning: learning a vector field (green arrows) approximating the gradient of the unknown data generating distribution.



$$C(\hat{x} \mid x) = N(\hat{x} \mid x, \sigma^2)$$

$$g(h) - x \propto \frac{\partial \log p(x)}{\partial x}$$

Remembering that h is the encoding $h(x)$ depending on x. So basically compares the reconstruction with the original data.
The Manifold Assumption: assume data lies on a lower dimensional non-linear manifold since variables in data are typically dependent. Regularized AR can afford to represent only variations that are needed to reconstruct training examples. AR mapping is sensitive only to changes in manifold direction.

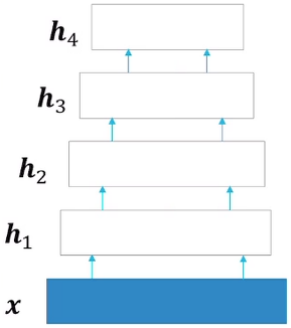
Contractive Autoencoder Penalize encoding function for input sensitivity.

$$J_{CAE}(\Theta) = \sum_{x \in S} (L(x, \tilde{x}) + \lambda \Omega(h))$$

$$\Omega(h) = \Omega(f(x)) = \| \frac{\partial f(x)}{\partial x} \|_F^2$$

You can as well penalize on higher order derivatives.

0.6.1 Basic Autoencoders

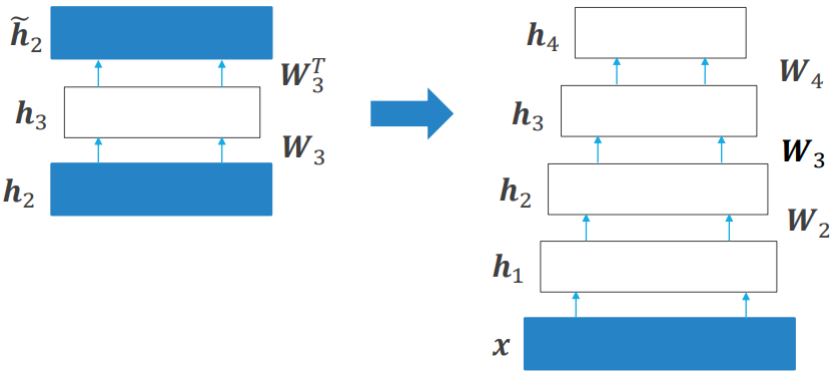


Unsupervised training

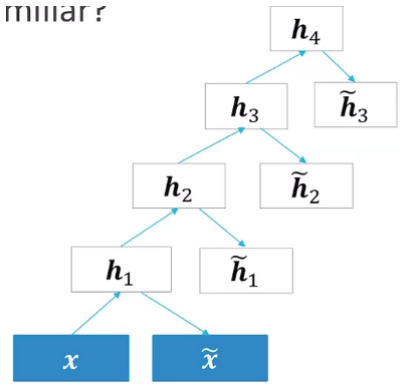
Hierarchical autoencoder

Extracts a representation of inputs that facilitates: data visualization, exploration, indexing... also facilitates the realization of a supervised task: adding another layer on top we can perform supervised learning using the representation learned by the autoencoder.

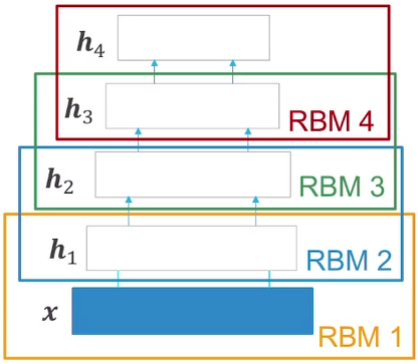
Unsupervised Layerwise Pretraining Incremental unsupervised construction of the deep Autoencoders.



Train h_1 without $\tilde{x}$ getting all the encoding vectors. Then train a new layer h_2 recostructing h_1 in output ad so on, obtaining the desired structure.
At the end, fine tune the whole autoencoder to optimize input reconstruction. You can use backpropagation, but it remains a supervised task.
If we rearrange the graph we obtain a stack of restricted Boltzmann machine:



This is called a **Deep Belief Network**: a stack of pairwise Restricted Boltzmann Machines.



A DBM it's **not recurrent**, is a deep autoencoder but not a deep RBM.
Training of a Deep Boltzmann Machine requires attention because of the recurrent interactions from higher layers to the bottom.

$$P(h_j^1 \mid x, h^2) = \sigma\left(\sum_j W_{ij} x_i + \sum_m W_{jm}^2 h_m^2\right)$$

$$P(x_i \mid h^1) = \sigma\left(\sum_j W_{ij}^1 h_j^1\right)$$

To train this, first pretrain the first layers, meaning fitting this model:

$$P(x \mid \Theta) = \sum_{h^1} P(h^1 \mid W^1) P(x \mid h^1, W^1)$$

Then pretrain the second layer changes h^1 prior by

$$P(h^1 \mid W^2) = \sum_{h^2} P(h^1, h^2 \mid W^2)$$

A trick, averaging the two models of h^1 can be approximated by taking half contribution from W^1 and half from W^2. Using full both would double count x contributions as h^2 depends on x.

Discriminative Fine Tuning