

# Data Preprocessing

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Introduction

---

④ **Unstructured datasets:**

- Examples described by a flat set of attributes: attribute-value matrix

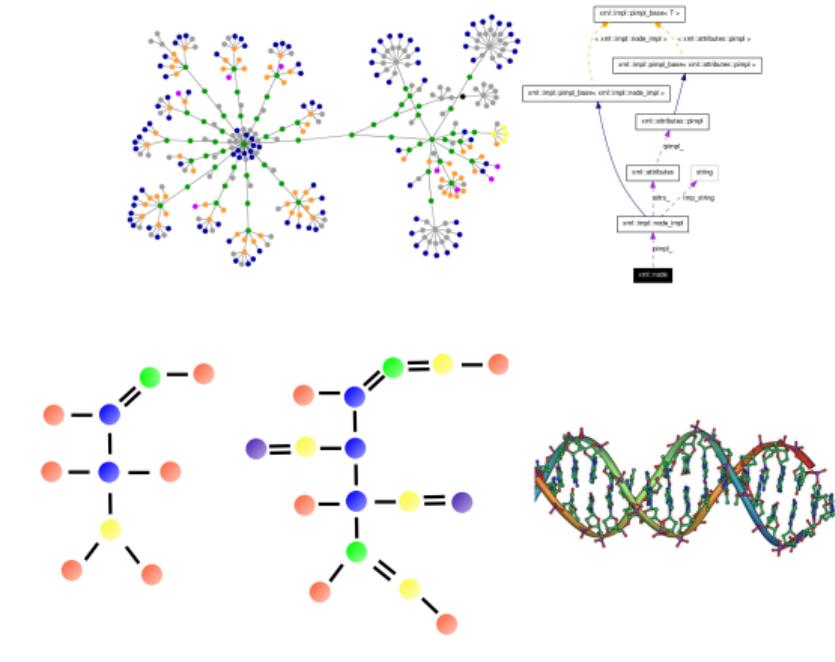
④ **Structured datasets:**

- Individual examples described by attributes but with relations among them:  
sequences (time, spatial, ...), trees, graphs
- Sets of structured examples (sequences, graphs, trees)

- ④ Only one table of observations
- ④ Each example represents an instance of the problem
- ④ Each instance is represented by a set of attributes (discrete, continuous)

A	B	C	...
1	3.1	a	...
1	5.7	b	...
0	-2.2	b	...
1	-9.0	c	...
0	0.3	d	...
1	2.1	a	...
:	:	:	..

- One sequential relation among instances (Time, Strings)
  - Several instances with internal structure
  - Subsequences of unstructured instances
  - One large instance
- Several relations among instances (graphs, trees)
  - Several instances with internal structure
  - One large instance



- ◎ Endless sequence of data
  - Several streams synchronized
  - Unstructured instances
  - Structured instances
- ◎ Static/Dynamic model



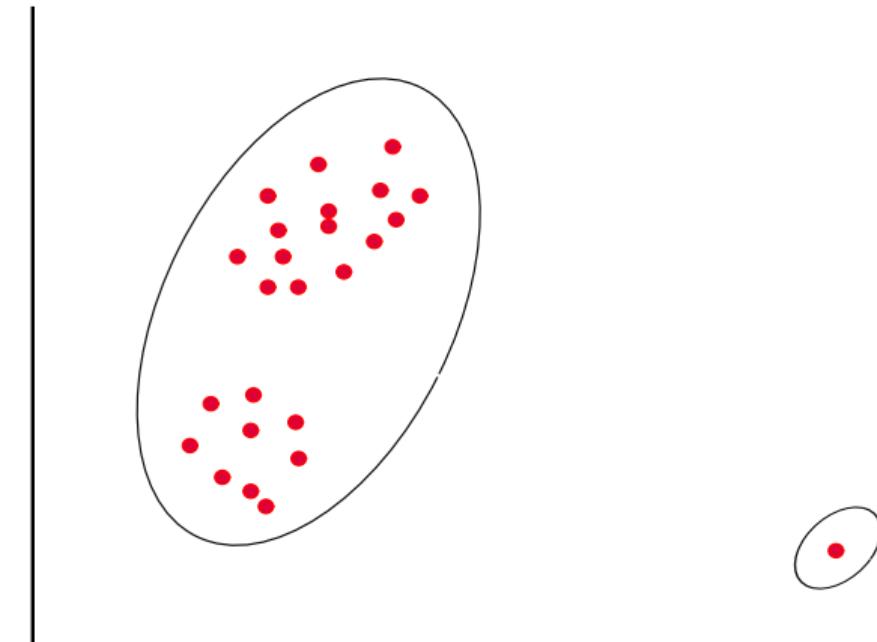
- Most unsupervised learning algorithms are specifically fitted for unstructured data
- The data representation is equivalent to a database table (attribute-value pairs)
- Specialized algorithms have been developed for structured data: Graph clustering, Sequence mining, Frequent substructures
- The representation of these types of data is sometimes algorithm dependent

# Data Preprocessing

---

- Usually raw data is not directly adequate for analysis
- The usual reasons:
  - The **quality** of the data (noise/missing values/outliers)
  - The **dimensionality** of the data (too many attributes/too many examples)
- The first step of any data task is to assess the quality of the data
- The techniques used for data preprocessing are usually oriented to unstructured data

- **Outliers:** Examples with extreme values compared to the rest of the data
- Can be considered as examples with erroneous values
- Have an important impact on some algorithms



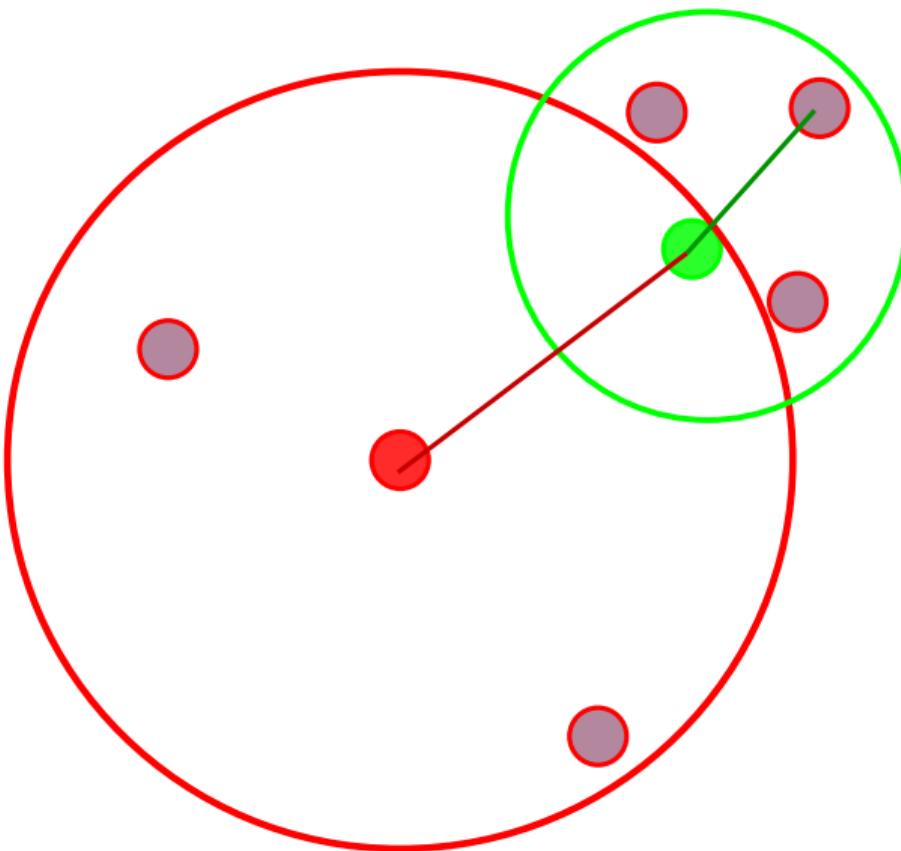
- ⑤ The exceptional values could appear in all or only a few attributes
- ⑤ The usual way to correct this problem is to eliminate the examples
- ⑤ If the exceptional values are only in a few attributes these could be treated as missing values

- Assumes a probabilistic distribution for the attributes
- **Univariate**
  - Perform  $Z$ -test or *student's* test
- **Multivariate**
  - **Deviation method:** reduction in data variance when eliminated
  - **Angle based:** variance of the angles to other examples
  - **Distance based:** variation of the distance from the mean of the data in different dimensions

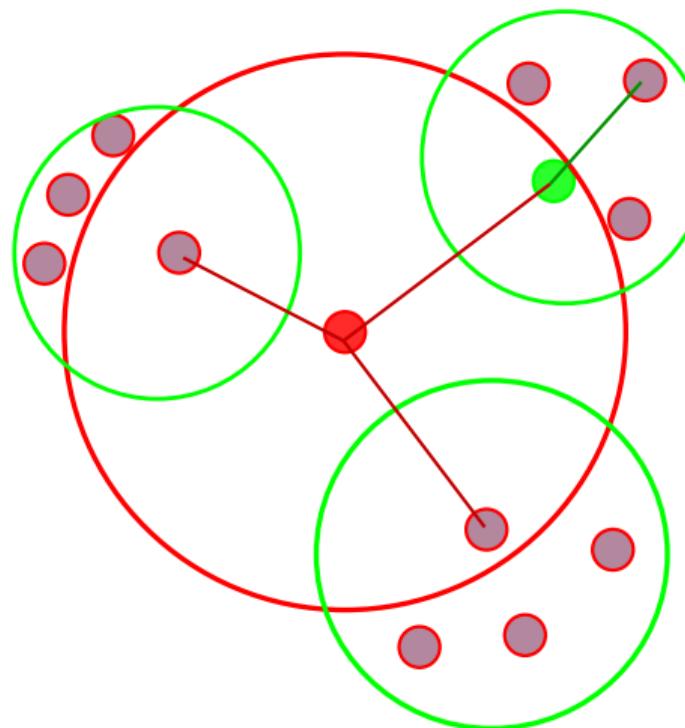
- ④ **Histogram based:** Define a multidimensional grid and discard cells with low density
- ④ **Distance based:** Distance of outliers to their k-nearest neighbors are larger
- ④ **Density based:** Approximate data density using Kernel Density estimation or heuristic measures (Local Outlier Factor, LOF)

- ⑤ LOF quantifies the outlierness of an example adjusting for variation in data density
- ⑥ Uses the distance of the k-th neighbor  $D_k(x)$  of an example and the set of examples that are inside this distance  $L_k(x)$
- ⑦ The **reachability distance** between two data points  $R_k(x, y)$  is defined as the maximum between the distance  $dist(x, y)$  and the  $y$ 's k-th neighbour distance

## Reachability distance



- ⑤ The **average reachability distance**  $AR_k(x)$  with respect of an example's neighbourhood ( $L_k(x)$ ) is defined as the average of the reachability distances to the neighbourhood

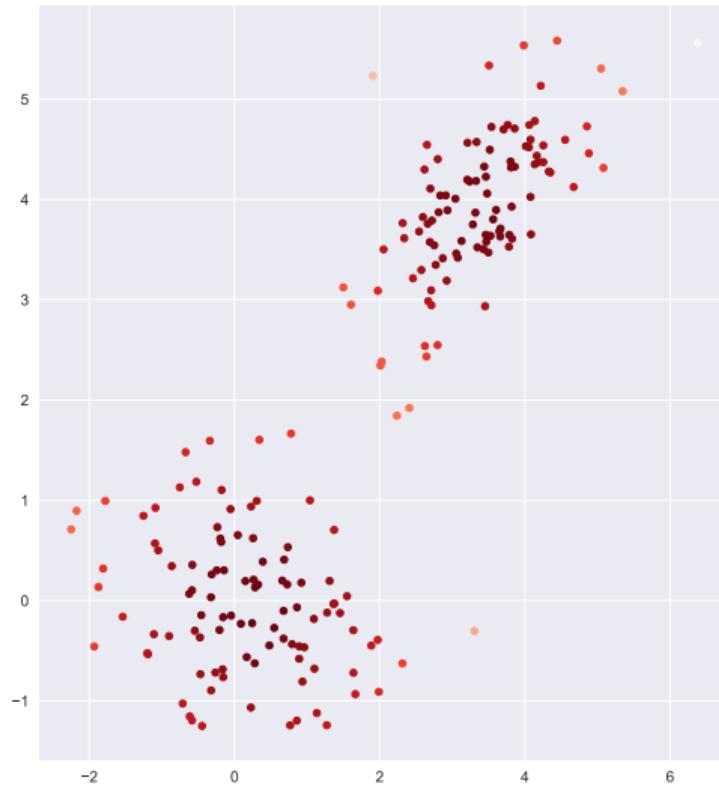
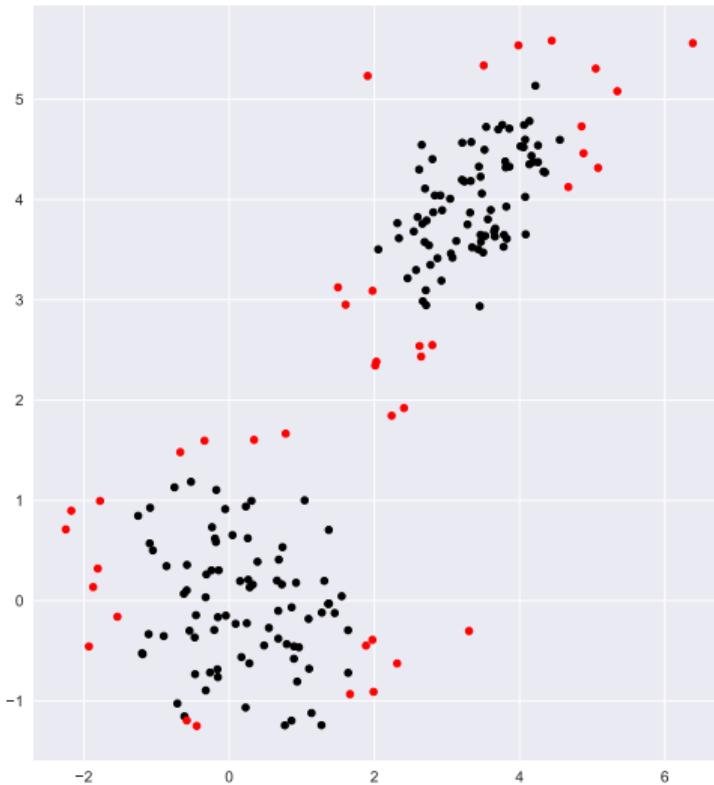


- ④ The **LOF** of an example is computed as the mean ratio between  $AR_k(x)$  and the average reachability of its  $k$  neighbors:

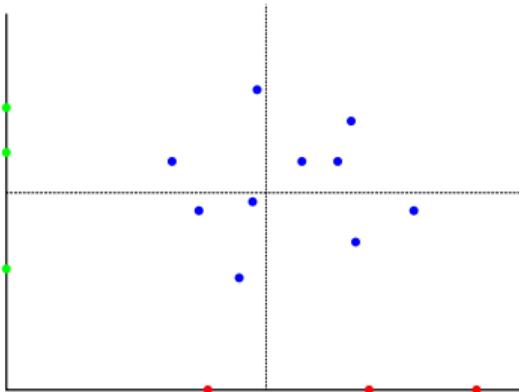
$$LOF_k(x) = \frac{1}{k} \sum_{y \in L_k(x)} \frac{AR_k(x)}{AR_k(y)}$$

- ⑤ This value ranks all the examples

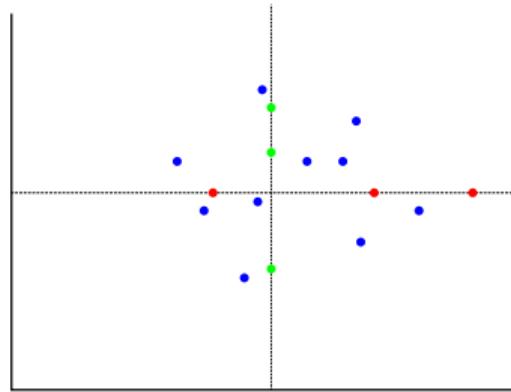
## Outliers: Local Outlier Factor



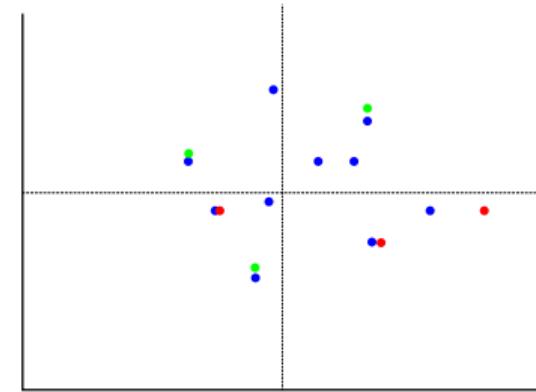
- Missing values appear because of errors or omissions during the gathering of the data
- They can be **substituted** to increase the quality of the dataset (**value imputation**)
  - Global constant for all the values
  - Mean or mode of the attribute (global central tendency)
  - Mean or mode of the attribute but only of the  $k$  nearest examples (local central tendency)
  - Learn a model for the data (regression, bayesian) and use it to predict the values
- **Problem:** changes the statistical distribution of the data



Missing Values



Mean substitution



1-neighbor substitution

- ⑤ Normalizations are applied to quantitative attributes in order to obtain attribute with similar behaviours
- ⑥ Possible transformations
  - Rescale the values, so they have the same range
  - Change the moments of the data distributions, so all the attributes have the same
  - Change the data distribution to a well behave probability function using non linear transformations

- ④ **Range Normalization:** Transform all the values of the attribute to a preestablished scale (e.g.: [0,1], [-1,1])

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

- ④ **Standard Score normalization:** Transform the data assumed gaussian distributed to  $\mathcal{N}(0, 1)$

$$\frac{x - \mu_x}{\sigma_x}$$

- ④ **Robust Scaling:** Transform using the median and the 50% interquartile range to reduce the effect of possible outliers

- ④ **Quantile Transformation:** Compute a quantized cumulative distribution function of the attribute and transform it using the inverse of the theoretical quantile function of the target distribution
- ④ **Power Transformation:** Apply a non linear monotonic transformation that maps the values of the feature closer to a gaussian distribution (Box-Coz, Yeo-Johnson)

$$x(\lambda) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \log(x) + \sigma^2 & \lambda = 0 \end{cases}$$

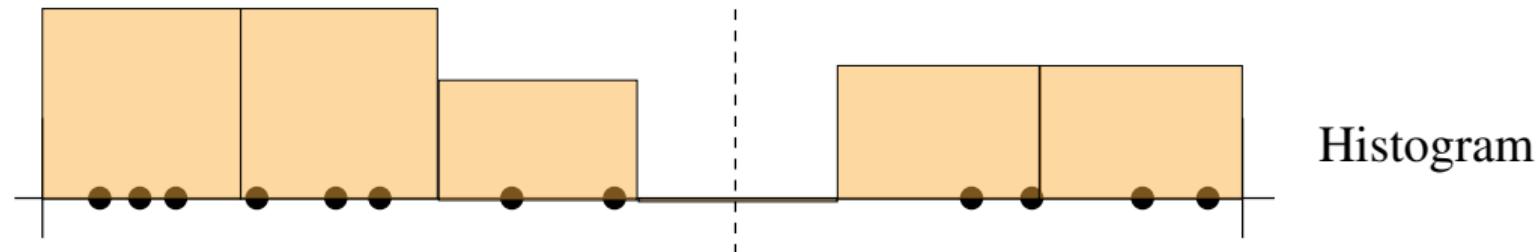
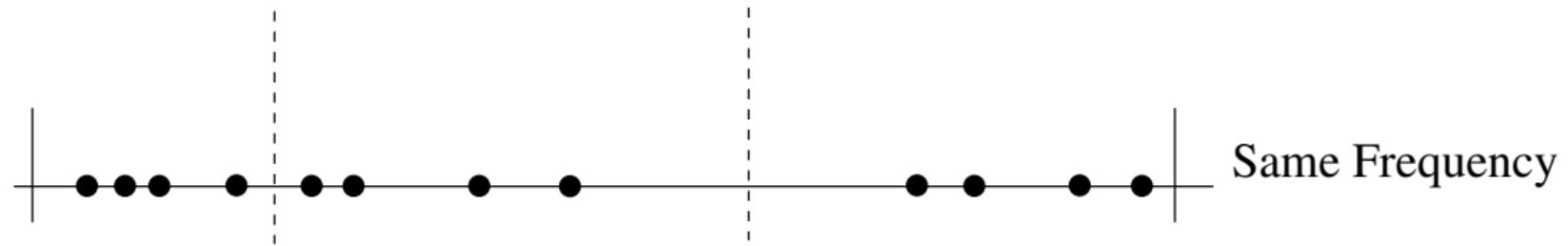
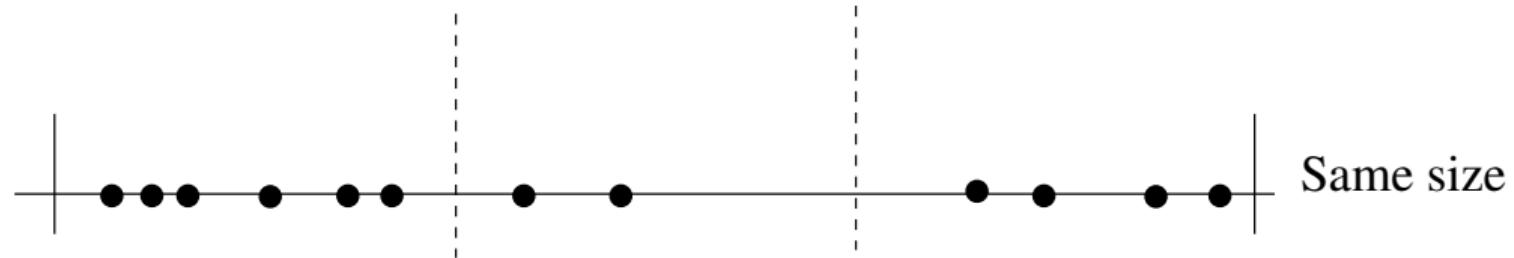
$$x(\lambda) = \begin{cases} \frac{(x+1)^\lambda - 1}{\lambda} & \lambda \neq 0, x \geq 0 \\ \log(x+1) + \sigma^2 & \lambda = 0, x \geq 0 \\ -\frac{(x+1)^{2-\lambda} - 1}{2-\lambda} & \lambda \neq 2, x < 0 \\ -\log(-x+1) + \sigma^2 & \lambda = 0, x < 0 \end{cases}$$

Discretization allows transforming quantitative attributes to qualitative attributes

- **Equal size bins:** Pick the number of values and divide the range of data in equal sized bins
- **Equal frequency bins:** Pick the number of values and divide the range of data so each bin has the same number of examples (the size of the intervals will be different)

Discretization allows transforming quantitative attributes to qualitative attributes

- **Distribution approximation:** Calculate a histogram of the data and fit a kernel function (KDE), the intervals are where the function has its minima
- **Other techniques:** Apply entropy based measures, Minimum Description Length (MDL), clustering



# Notebooks

---

These two Python Notebooks show some examples of the effect of missing values imputation and data discretization and normalization

- Missing Values Notebook ([click here](#) to go to the url)
- Preprocessing Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# Dimensionality Reduction

---

- ⑤ Problems due to the **dimensionality** of data
  - The **computational cost** of processing the data
  - The **quality** of the data
- ⑥ Elements that define the dimensionality of data
  - The number of examples
  - The number of attributes
- ⑦ Usually the problem of having too many examples can be solved using sampling.

- ① The number of attributes has an impact on the **performance**:
  - Poor scalability
  - Inability to cope with **irrelevant/noisy/redundant attributes**
- ② Methodologies to reduce the number of attributes:
  - **Dimensionality reduction**: Transforming to a space of less dimensions
  - **Feature subset selection**: Eliminating not relevant attributes

- New dataset that preserves most of the information of the original data but with less attributes
- Many techniques have been developed for this purpose
  - Projection to a space that **preserve the statistical distribution** of the data (PCA, ICA)
  - Projection to a space that **preserves distances** among the data (Multidimensional scaling, random projection, nonlinear scaling)

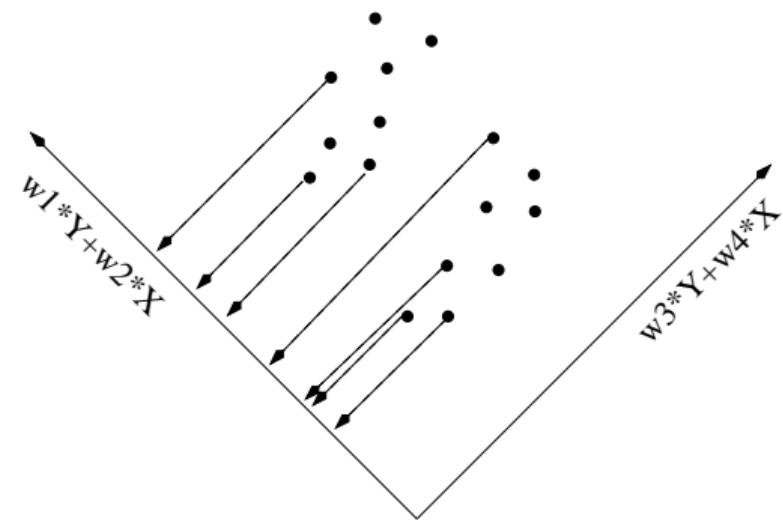
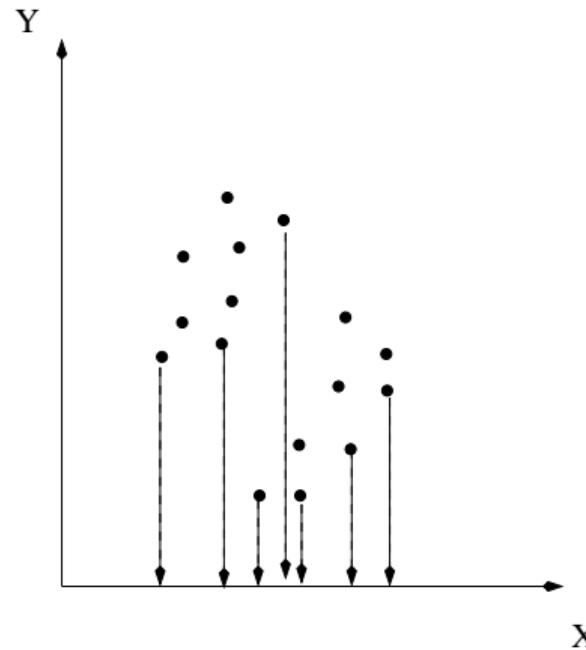
## Linear methods

---

## ⑤ Principal Component Analysis:

- Data is projected onto a set of **orthogonal dimensions** (components) that are a **linear combination** of the original attributes
- The components are **uncorrelated** and are **ordered** by the information they have
- We assume data follows **gaussian** distribution
- Global variance is preserved

Computes a projection matrix where the dimensions are orthogonal (linearly independent) and data variance is preserved



- ④ Principal components: vectors that are the best linear approximation of the data

$$f(\lambda) = \mu + V_q \lambda$$

$\mu$  is a location vector in  $\mathbb{R}^p$ ,  $V_q$  is a  $p \times q$  matrix of  $q$  orthogonal unit vectors and  $\lambda$  is a  $q$  vector of parameters

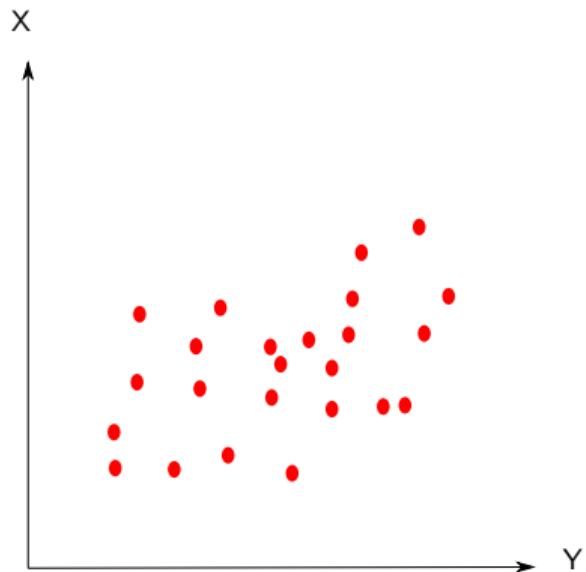
- ④ The reconstruction error for the data is minimized:

$$\min_{\mu, \{\lambda_i\}, V_q} \sum_{i=1}^N \|x_i - \mu - V_q \lambda_i\|^2$$

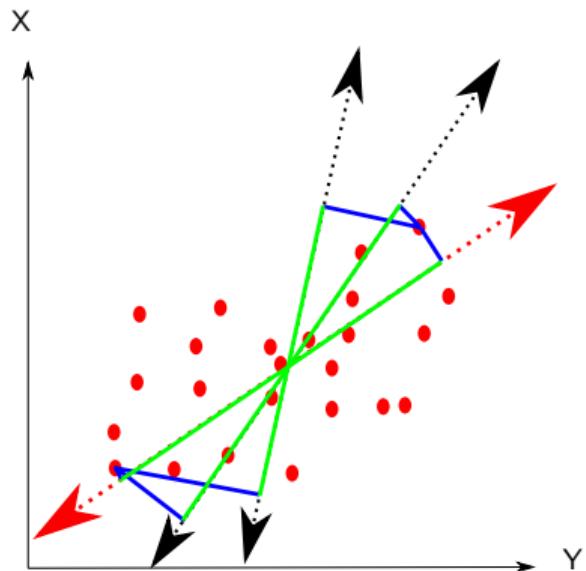
- Assuming  $\bar{x} = 0$  we can obtain the projection matrix by **Singular Value Decomposition** of the data matrix  $X$

$$X = UDV^T$$

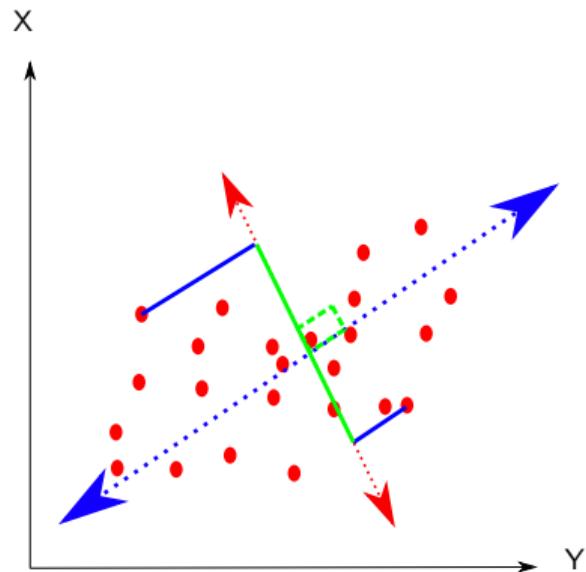
- $U$  is a  $N \times p$  orthogonal matrix, its columns are the **left singular vectors**
- $D$  is a  $p \times p$  diagonal matrix with ordered diagonal values called the **singular values**
- The columns of  $UD$  are the **principal components**
- We can pick the first principal components that account for a percentage of the total variance (e.g. 90%)



Original Data



First component along the maximum variance of the data



Next component maximum variance perpendicular to the other components

- PCA is a linear transformation, this means that if data is linearly separable, the reduced dataset will be linearly separable (given enough components)
- We can use the **kernel trick** to map the original attribute to a space where non linearly separable data is linearly separable
- Distances among examples are **defined as a dot product** that can be obtained using a kernel:

$$d(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) = K(x_i, x_j)$$

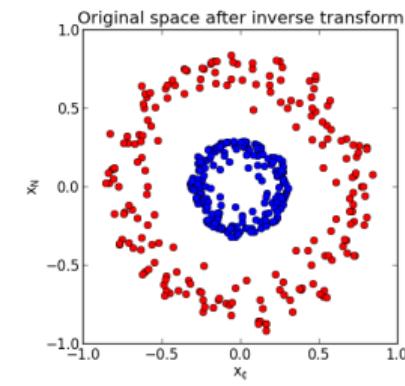
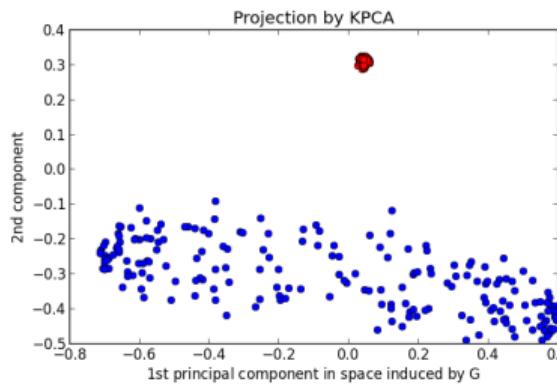
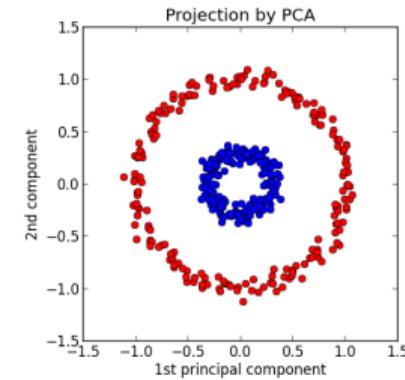
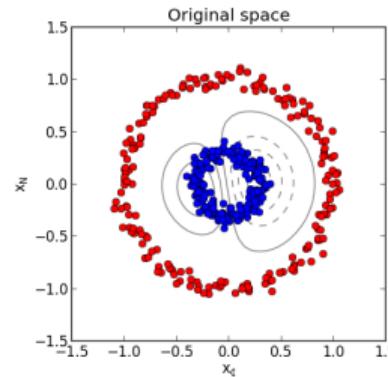
- Different kernels can be used to perform the transformation to the feature space (polynomial, gaussian...)
- The computation of the components is equivalent to PCA but performing the eigen decomposition of the covariance matrix computed for the transformed examples

$$C = \frac{1}{M} \sum_{j=1}^M \Phi(x_j) \Phi(x_j)^T$$

- The components are lineal combinations of features in the feature space

- ⑤ **Pro:** Helps to discover patterns that are non linearly separable in the original space
- ⑥ **Con:** Does not give a weight/importance for the new components

# Kernel PCA

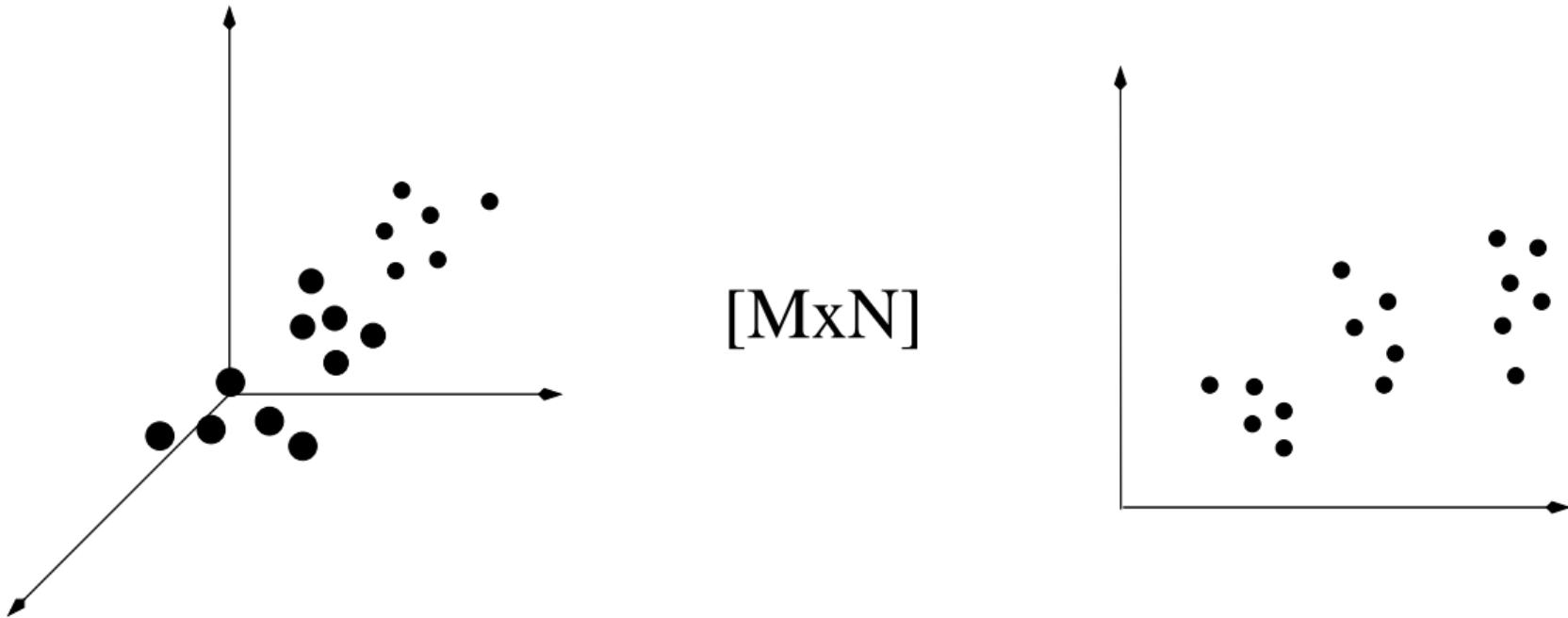


- PCA transforms data to a space of the same dimensionality (all eigenvalues are non zero)
- An alternative is to solve the minimization problem posed by the reconstruction error using regularization
- A penalization term is added to the objective function proportional to the norm of the eigenvalues matrix

$$\min_{U,V} \|X - UV\|_2^2 + \alpha \|V\|_1$$

- The  $\ell_1$ -norm regularization will encourage sparse solutions (zero eigenvalues)

A transformation matrix maps a dataset from M dimensions to N dimensions preserving pairwise data distances



- ④ **Multidimensional Scaling:** Projects the data to a space with less dimensions **preserving the pair distances** among the data
- ④ A projection matrix is obtained by optimizing a function of the pairwise distances (stress function)
- ④ The actual attributes are not used in the transformation
- ④ Different objective functions that can be used (least squares, Sammung mapping, classical scaling...).

- ④ Least Squares Multidimensional Scaling (MDS)
- ④ The distortion is defined as the square distance between the original distance matrix and the distance matrix of the new data

$$S_D(z_1, z_2, \dots, z_n) = \left[ \sum_{i \neq i'} (d_{ii'} - \|z_i - z_{i'}\|_2)^2 \right]$$

- ④ The problem is defined as:

$$\arg \min_{z_1, z_2, \dots, z_n} S_D(z_1, z_2, \dots, z_n)$$

- ④ Several optimization strategies can be used
- ④ If the distance matrix is euclidean it can be solved using eigen decomposition just like PCA
- ④ In other cases gradient descent can be used using the derivative of  $S_D(z_1, z_2, \dots, z_n)$  and a step  $\alpha$  in the following fashion:
  1. Begin with a guess for  $Z$
  2. Repeat until convergence:

$$Z^{(k+1)} = Z^{(k)} - \alpha \nabla S_D(Z)$$

- ④ Sammong Mapping (emphasis on smaller distances)

$$S_D(z_1, z_2, \dots, z_n) = \left[ \sum_{i \neq i'} \frac{(d_{ii'} - \|z_i - z_{i'}\|)^2}{d_{ii'}} \right]$$

- ④ Classical Scaling (similarity instead of distance)

$$S_D(z_1, z_2, \dots, z_n) = \left[ \sum_{i \neq i'} (s_{ii'} - \langle z_i - \bar{z}, z_{i'} - \bar{z} \rangle)^2 \right]$$

- ④ Non metric MDS (assumes a ranking among the distances, non euclidean space)

$$S_D(z_1, z_2, \dots, z_n) = \frac{\sum_{i,i'} [\theta(\|z_i - z_{i'}\|) - d_{ii'}]^2}{\sum_{i,i'} d_{i,i'}^2}$$

- A random transformation matrix is generated:
  - Rectangular matrix  $N \times d$
  - Columns must have unit length
  - Elements are generated from a gaussian distribution
- A matrix generated this way is almost orthogonal
- The projection will preserve the relative distances among pairs of examples
- The Johnson-Lindenstrauss lemma allows to pick a number of dimensions to obtain the desired approximation

- This formulation assumes that the data is a sum of unknown positive latent variables
- NMF performs an approximation of a matrix as the product of two matrices

$$V = W \times H$$

- The **main difference** with PCA is that the values of the matrices are **constrained to be positive**
- The positiveness assumption helps to interpret the result
  - Eg.: In text mining, a document is an aggregation of topics

- The optimization problem for NMF is defined as given a number of dimensions  $k < \min(m, n)$ , find the pair of non negative matrices  $W \in \mathbb{R}^{m \times k}$  and  $H \in \mathbb{R}^{k \times n}$  that minimize

$$f(W, H) = \frac{1}{2} \|A - WH\|_F^2$$

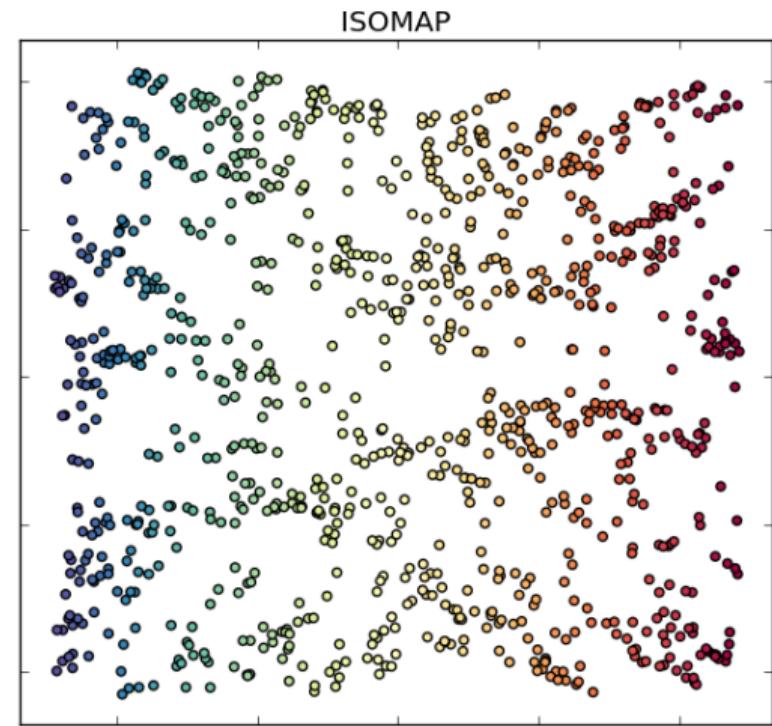
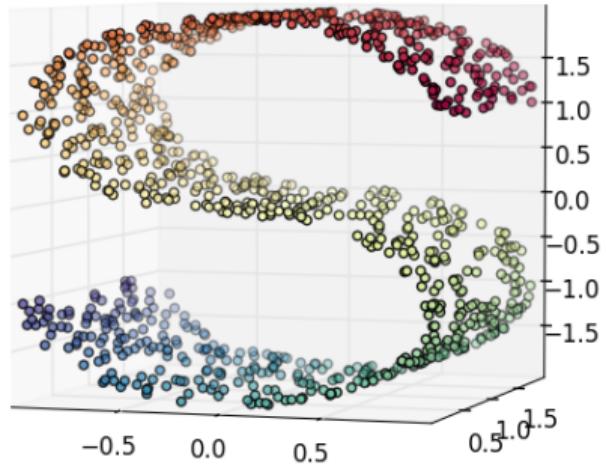
- Given that this is a non convex problem can be solved using gradient descent or alternate least squares
- There are several variants forcing sparsity or orthogonality of the matrices

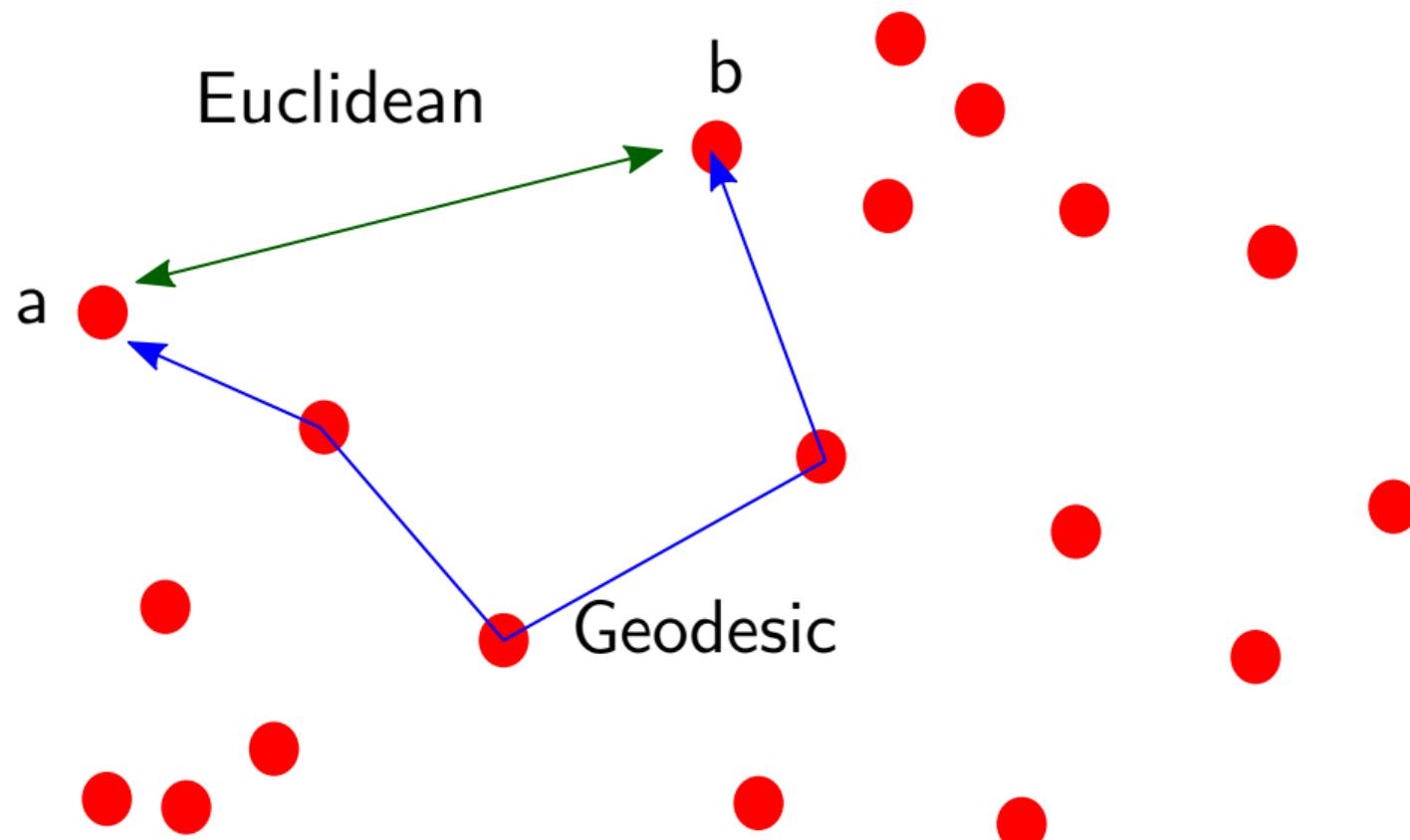
## Non linear methods

---

- The previous methods perform a linear transformation between the original space, and the final space
- For some datasets this kind of transformation is not enough to maintain the information of the original data
- Nonlinear transformations methods:
  - ISOMAP
  - Local Linear Embedding
  - Local MDS
  - t-SNE

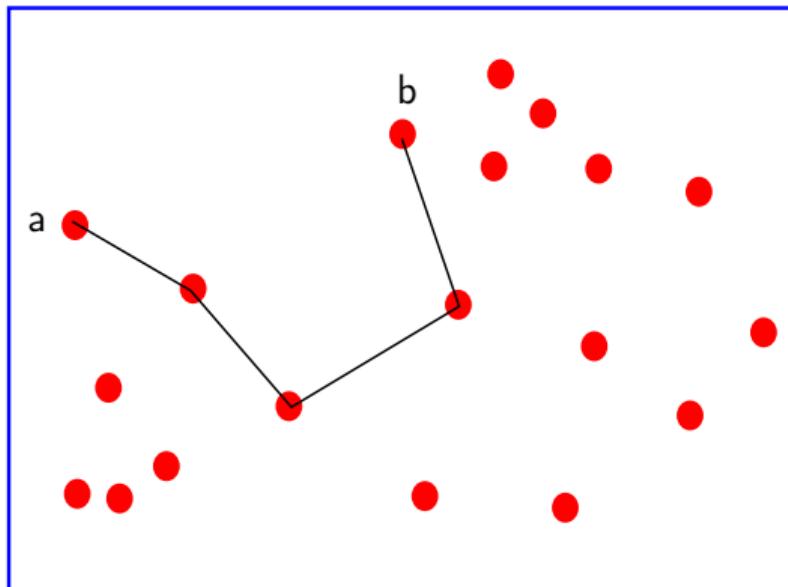
- Assumes a low dimensional dataset embedded in a larger number of dimensions
- The **geodesic distance** is used instead of the euclidean distance
- The relation of an instance with its immediate neighbors is more representative of the structure of the data
- The transformation generates a new space that preserves neighborhood relationships



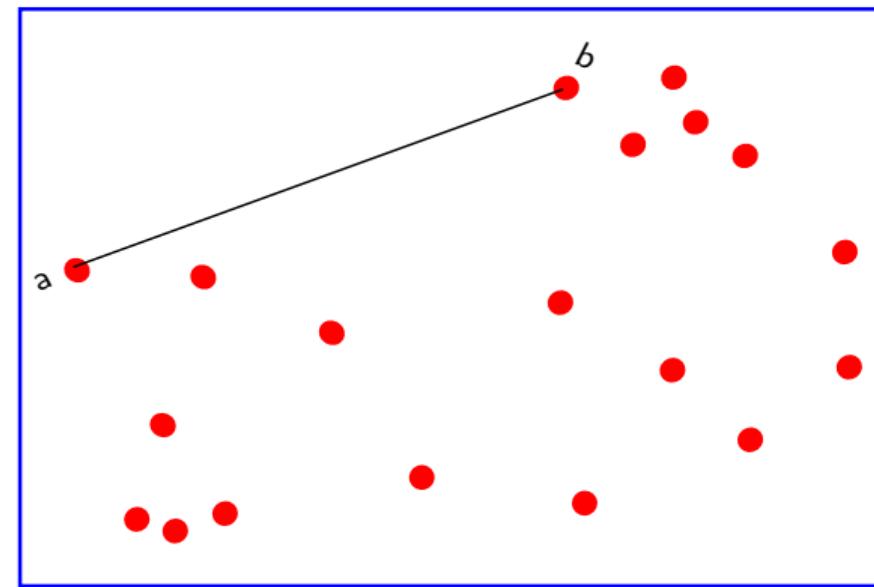


1. For each data point find its  $k$  closest neighbors (points at minimal euclidean distance)
2. Build a graph where each point has an edge to its closest neighbors
3. Approximate the geodesic distance for each pair of points by the shortest path in the graph
4. Apply a MDS algorithm to the distance matrix of the graph

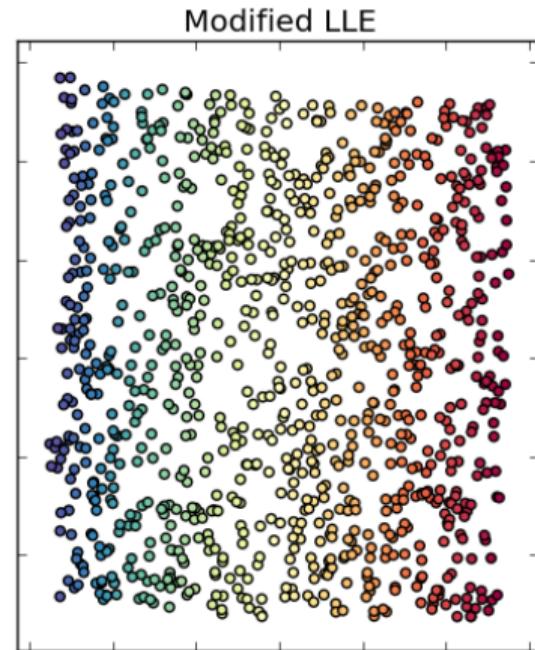
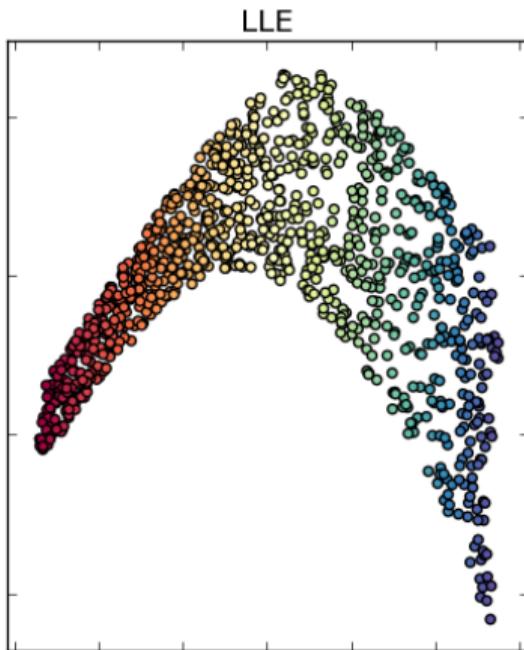
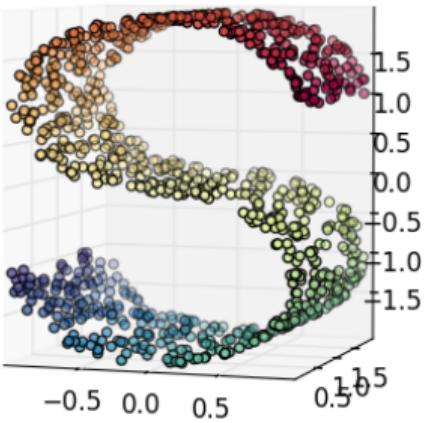
Original



Transformed



- Performs a transformation that **preserves local structure**
- Assumes that each instance can be reconstructed by a linear combination of its neighbors (weights)
- From these weights a new set of data points that preserve the reconstruction is computed for a lower dimensional space
- Different variants of the algorithm exist



1. For each data point find the K nearest neighbors in the original space of dimension  $p$  ( $\mathcal{N}(i)$ )
2. Approximate each point by a mixture of the neighbors:

$$\min_{W_{ik}} \|x_i - \sum_{k \in \mathcal{N}(i)} w_{ik} x_k\|^2$$

and  $\sum_{k \in \mathcal{N}(i)} w_{ik} = 1$  and  $K < p$

3. Find points  $y_i$  in a space of dimension  $d < p$  that minimize:

$$\sum_{i=0}^N \|y_i - \sum_{k \in \mathcal{N}(i)} w_{ik} y_k\|^2$$

- ④ Performs a transformation that preserves locality of closer points and puts farther away non neighbor points
- ④ Given a set of pairs of points  $\mathcal{N}$  where a pair  $(i, i')$  belong to the set if  $i$  is among the  $K$  neighbors of  $i'$  or viceversa
- ④ Minimize the function:

$$S_L(z_1, z_2, \dots, z_N) = \sum_{(i, i') \in \mathcal{N}} (d_{ii'} - \|z_i - z_{i'}\|)^2 - \tau \sum_{(i, i') \notin \mathcal{N}} (\|z_i - z_{i'}\|)$$

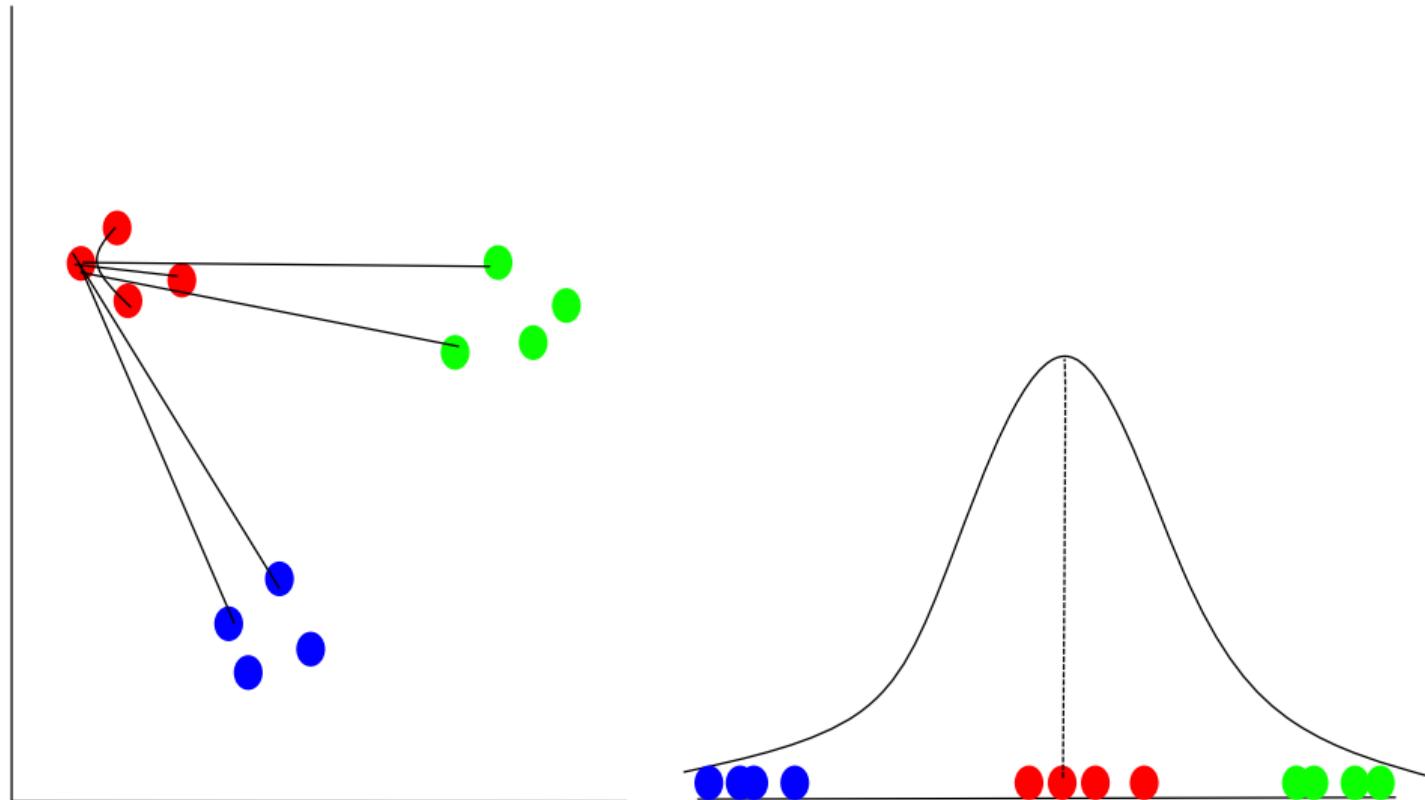
- ④ The parameters  $\tau$  controls how much the non neighbors are scattered

- ④ t-Stochastic Neighbor Embedding (t-SNE)
- ④ Used mainly as visualization tool (2-3 dimensions), not for transforming the data for applying ML algorithms
- ④ Assumes that the distances among examples define a probability distribution that must be preserved in a lower dimensional space
- ④ Many parameters apart from the number of target dimensions, tricky to use (see this [link](#))
- ④ Stochastic algorithm (depends on initialization), it can be initialized to the result of PCA

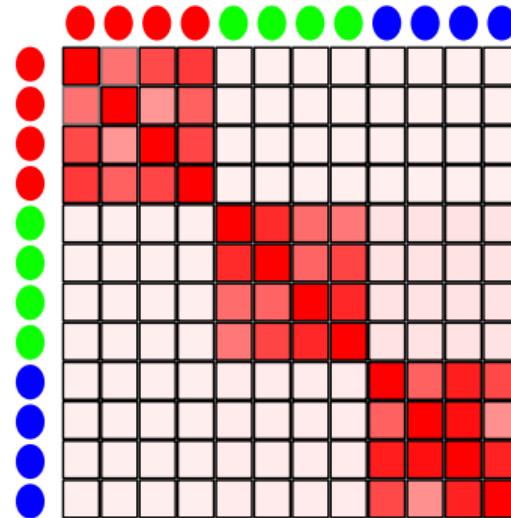
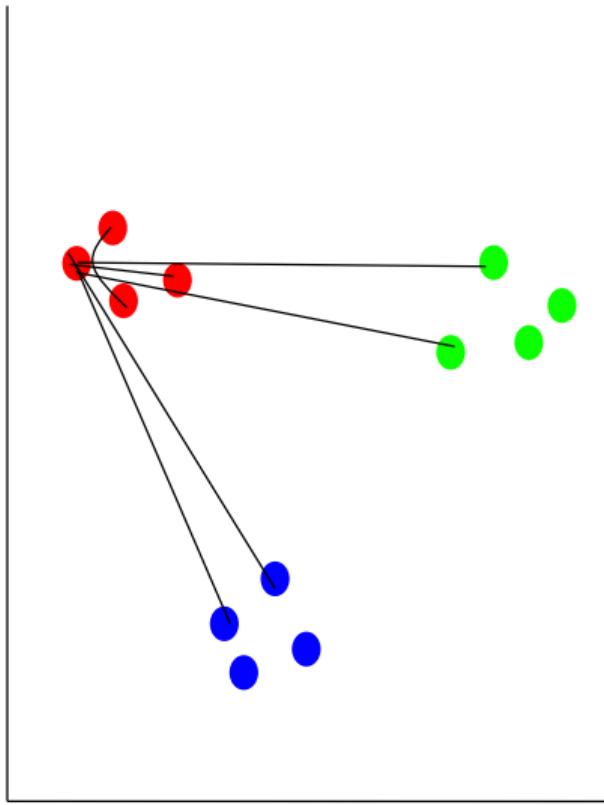
- ⑤ t-SNE assumes that distances among examples are defined as a gaussian distribution
- ⑥ Distances from each example to the rest are scaled to sum one (so it is a proper probability distribution)
- ⑦ We want to find a projection of the data that preserves this probability distribution on a lower dimensionality space
- ⑧ Examples are distributed in the new space, and their distance distributions are computed
- ⑨ Examples are iteratively moved to minimize the Kullback-Leibler distance<sup>1</sup> among the distribution of the neighbours distances in the original and in the projected space

---

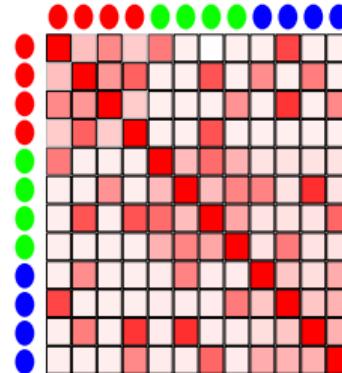
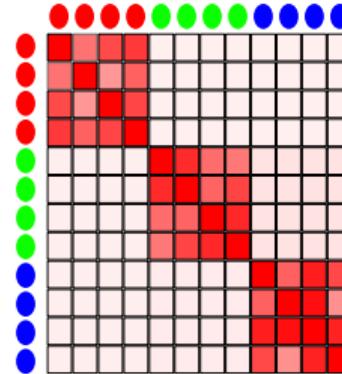
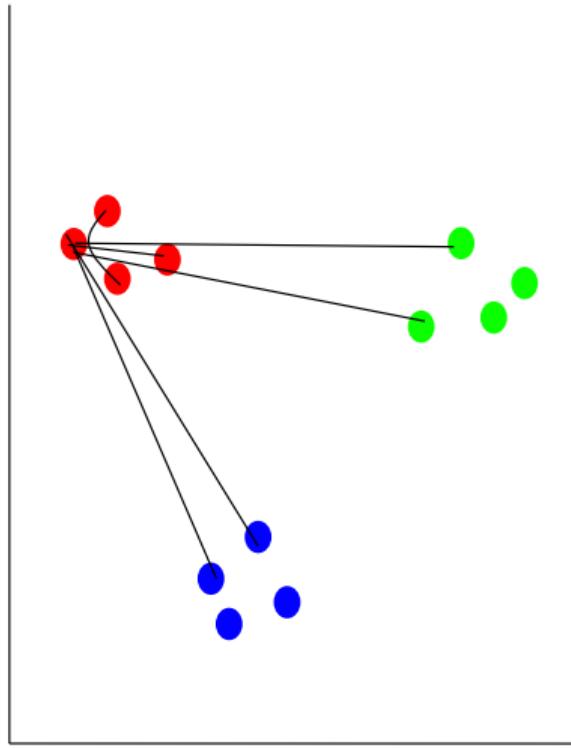
<sup>1</sup>Measure from Information Theory that computes the similarity among data distributions



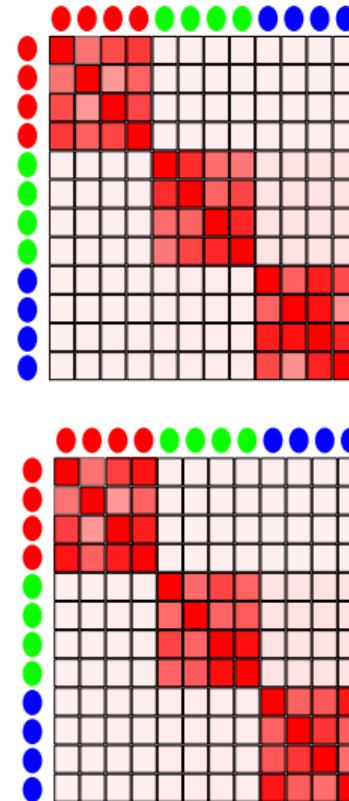
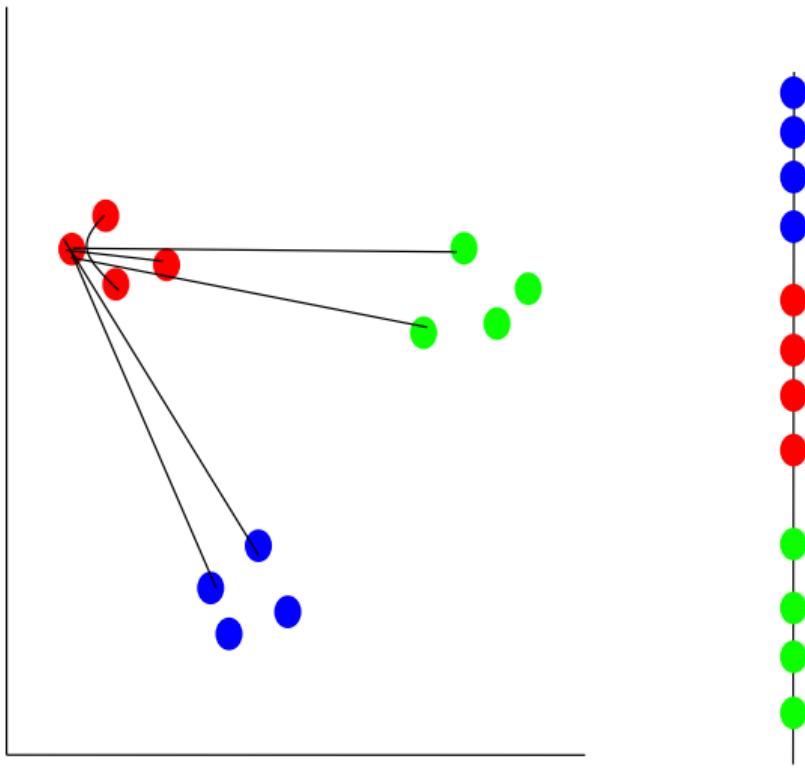
Each example has a distance probability distribution



A similarity distribution is obtained



We distribute the data in a lower dimensionality space



Data is moved so the similarity distributions get close

- ④ Autoencoders are fully connected neural networks able to learn a representation for a dataset
- ④ They are trained to reproduce their input on their output using a specific architecture that learn the representation
- ④ An autoencoder is defined by
  - An encoder that can transform the data to a possibly lower dimensional space
  - A decoder that recreates the original data from the representation obtained by the encoder
- ④ Only the encoder needs to be kept after training for performing the transformation

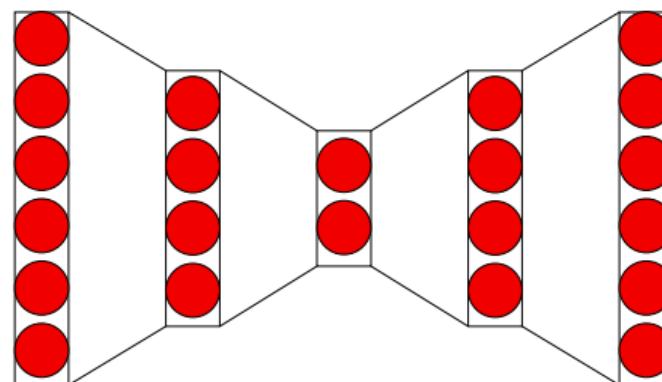
- ⑤ The training needs to minimize a loss function that measures the difference between the original data and the output of the network.
- ⑥ Encoder and decoder can be seen as two functions applied to the data ( $f$  and  $g$ ), so the loss would be:

$$\mathcal{L}(x) = L(x, g(f(x)))$$

where  $L$  can be the mean squared error

① The architecture of an autoencoder combines:

- An encoder that has several fully connected layers with progressively fewer units, with the number of units of the last layer the dimensionality of the resulting space
- An encoder defined symmetrically, so it increases the number of units until the original number of dimensions is reached



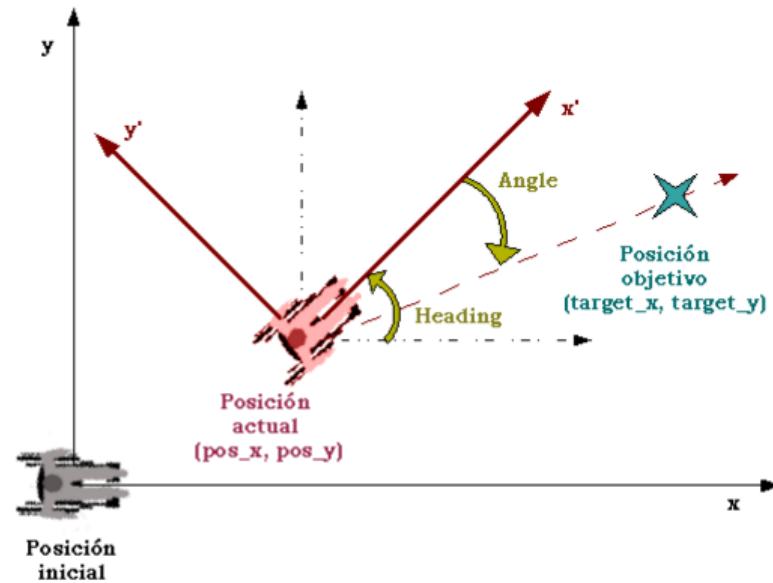
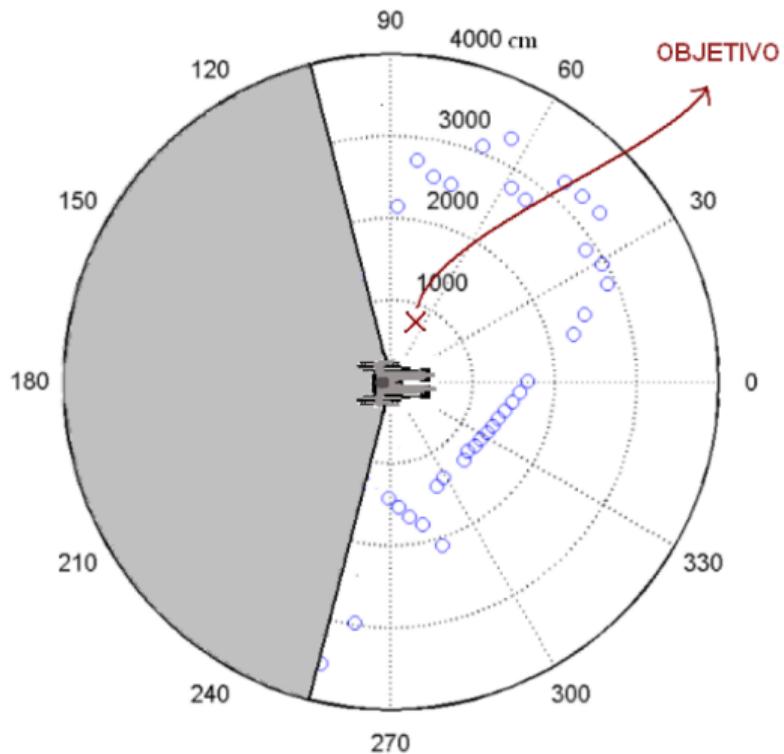
- When linear activation functions are used this architecture is enough to learn a representation that is equivalent to PCA
- Using non-linearities or having a decoder with more dimensionality than the input need for techniques that avoid trivial solutions or overfitting
  - **Regularization** is a common technique that constraints the values of the parameters of the model or bias the search towards smooth functions
  - **Denoising autoencoders** change the loss function by using during training corrupted examples, so the network learn how to reconstruct the original

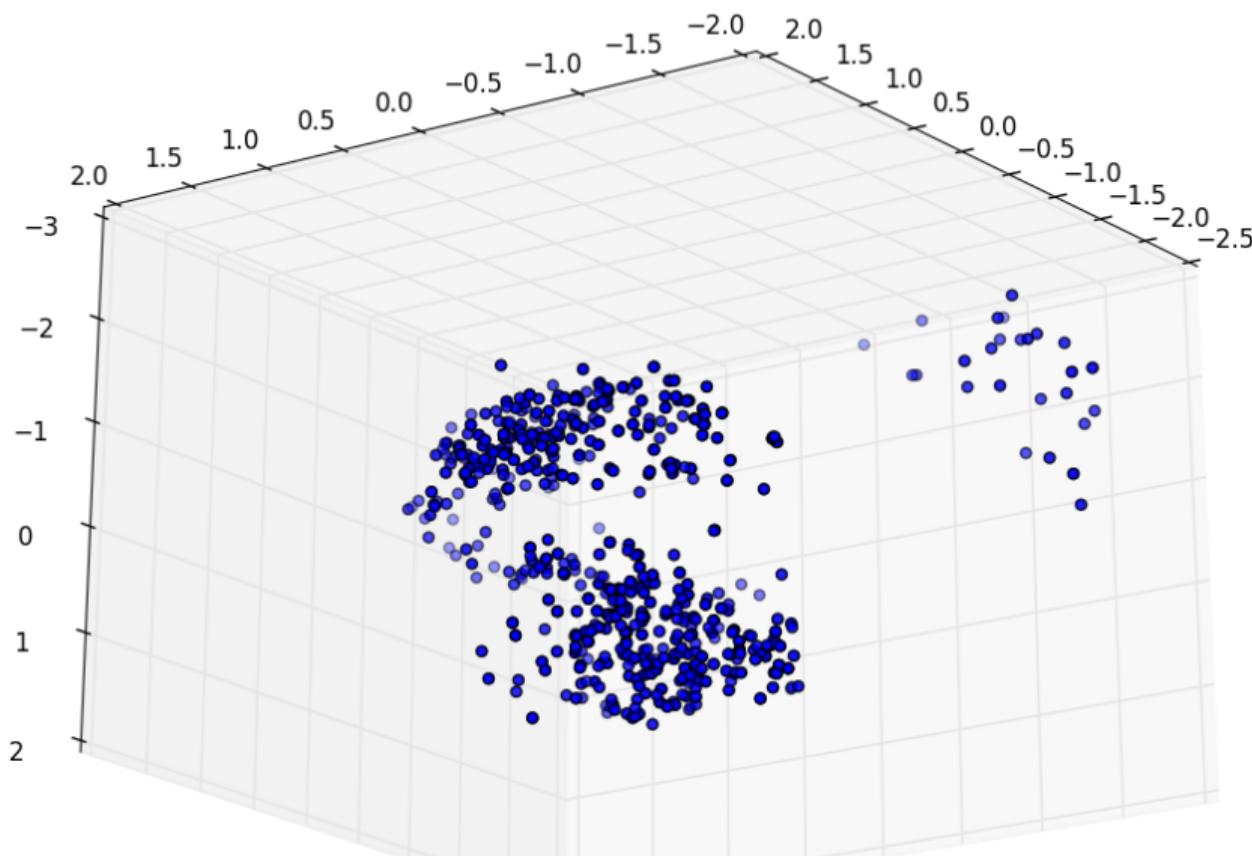
Application: Wheelchair control

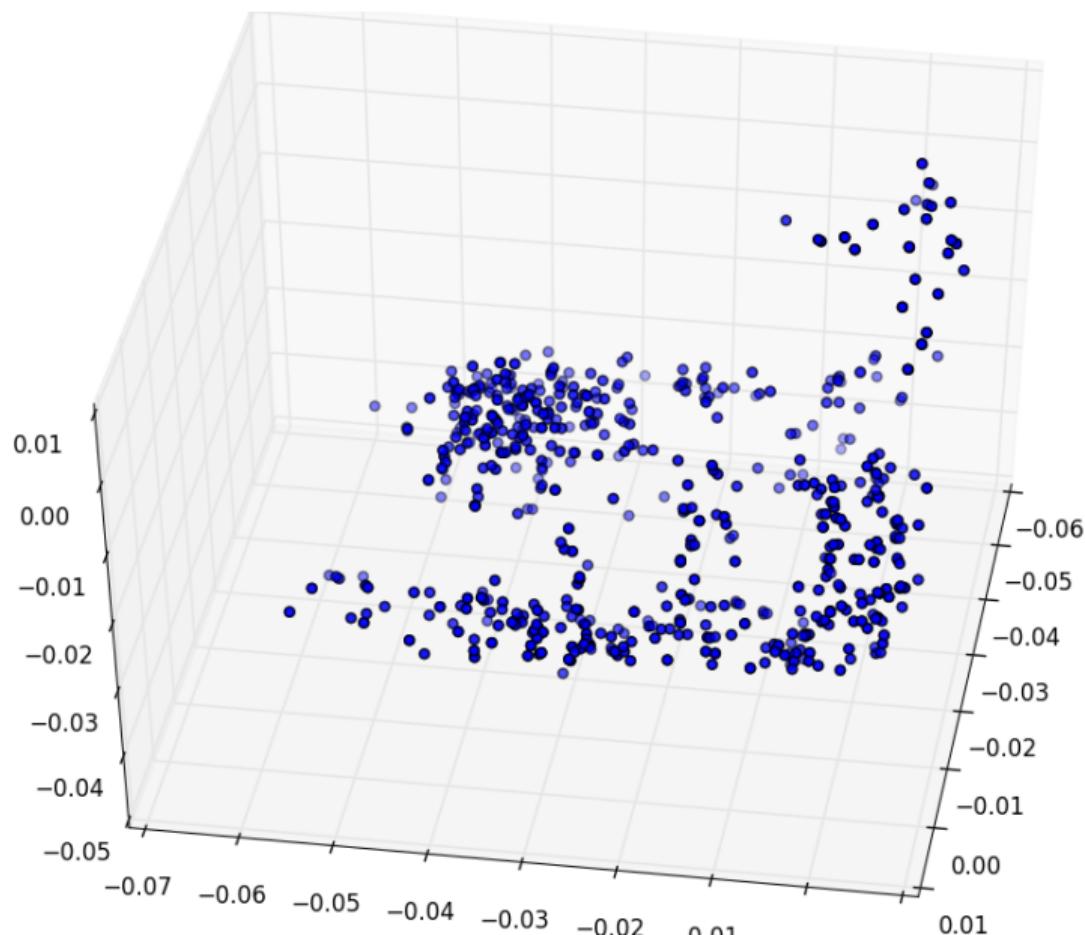
---

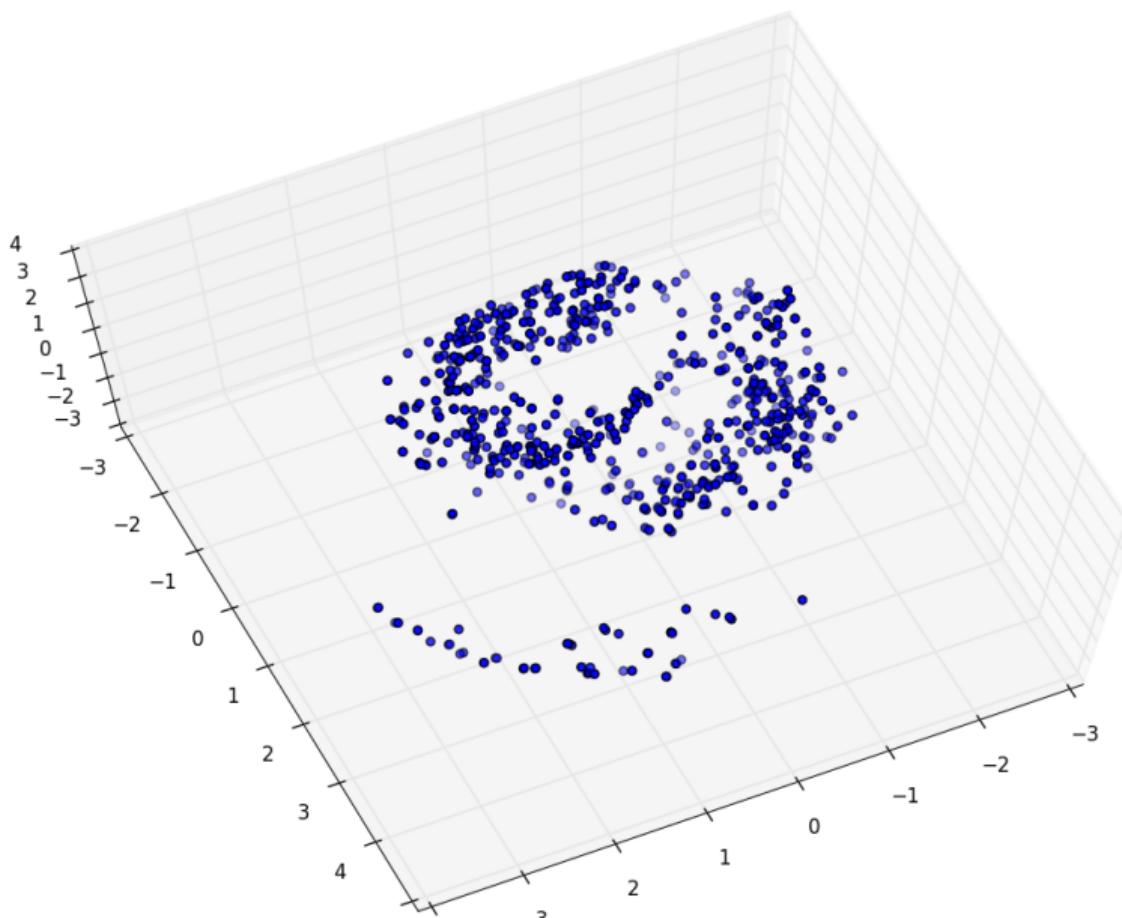
- ⑤ Wheelchair with shared control (patient/computer)
- ⑤ Recorded trajectories of several patients in different situations
  - Angle/distance to the goal, Angle/distance to the nearest obstacle from around the chair (210 degrees)
- ⑤ Characterization about how the computer helps the patients with different handicaps
- ⑤ Is there any structure in the trajectory data?

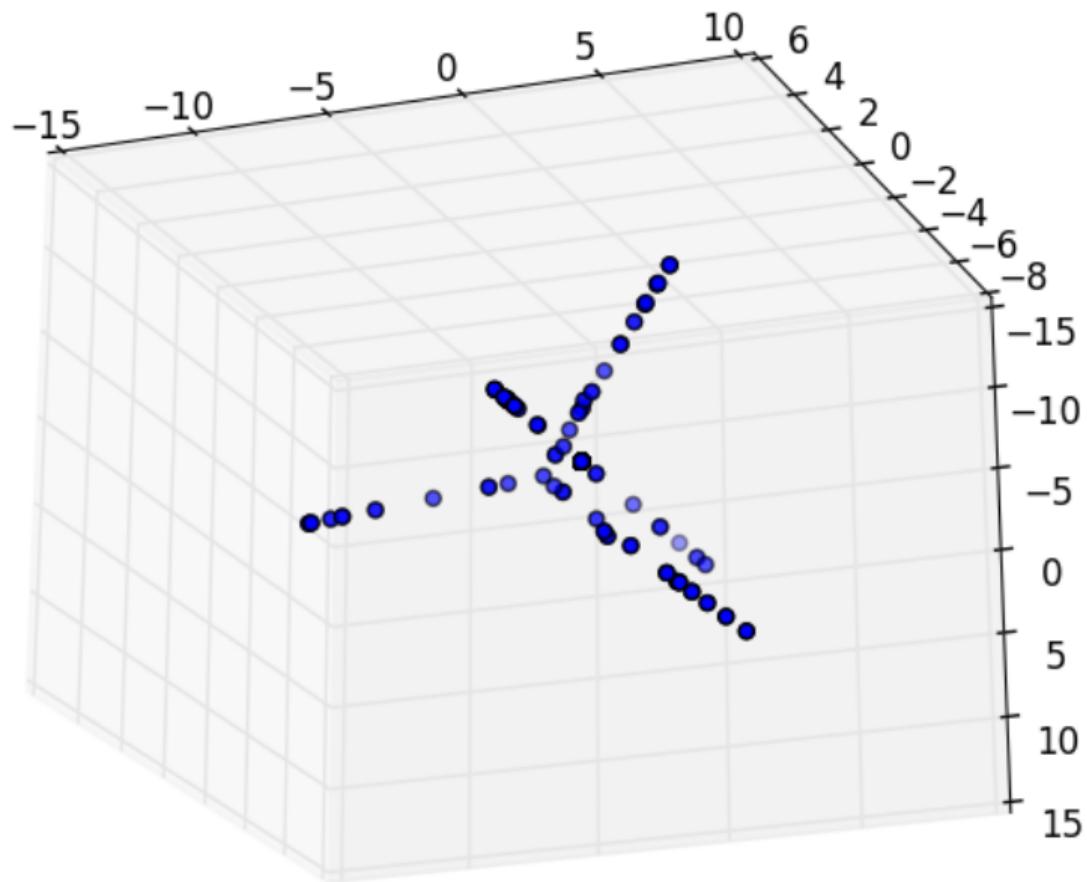


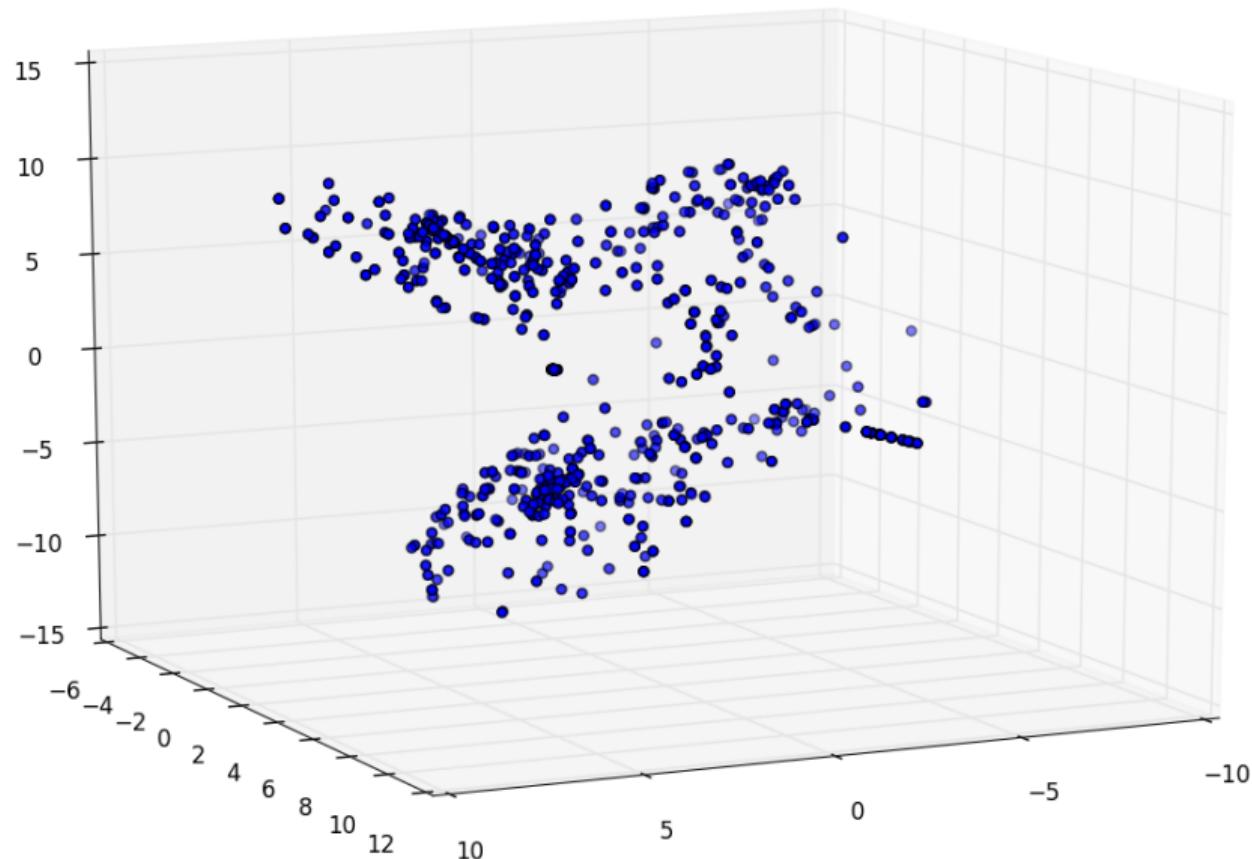












## Attribute Selection

---

- To **eliminate** from the dataset all the **redundant or irrelevant attributes**
- The original attributes are preserved
- Less developed than in Supervised Attribute Selection
  - **Problem:** An attribute can be relevant or not depending on the goal of the discovery process
- There are mainly two techniques for attribute selection: Wrapping and Filtering

- ⑤ A model evaluates the relevance of subsets of attributes
- ⑤ In supervised learning this is easy, in unsupervised learning it is very difficult
- ⑤ Results depend on the chosen model and on how well this model captures the actual structure of the data

- ④ Clustering algorithms that compute weights for the attributes based on probability distributions
- ④ Clustering algorithms with an objective function that penalizes the size of the model
- ④ Consensus clustering

- A measure evaluates the relevance of each attribute individually
- This kind of measures are difficult to obtain for unsupervised tasks
- The idea is to obtain a measure that evaluates the capacity of each attribute to reveal the structure of the data (e.g.: class separability, similarity of instances in the same class)

- ⑤ Measures of properties of the spatial structure of the data (Entropy, PCA, laplacian matrix)
- ⑥ Measures of the relevance of the attributes respect the inherent structure of the data
- ⑦ Measures of attribute correlation

- ④ The **Laplacian Score** is a filter method that ranks the features respect to their ability of preserving the natural structure of the data.
- ④ This method uses the spectral matrix of the graph computed from the near neighbors of the examples

- ④ The Similarity matrix is usually computed using a gaussian kernel (edges not present have a value of 0)

$$S_{ij} = e^{\frac{||x_i - x_j||^2}{\sigma}}$$

- ④ The Degree matrix is a diagonal matrix where the elements are the sum of the rows of  $S$
- ④ The Laplacian matrix is computed as

$$L = S - D$$

- ④ The score first computes for each attribute  $r$  and their values  $f_r$  the transformation  $\tilde{f}_r$  as:

$$\tilde{f}_r = f_r - \frac{f_r^T D \mathbf{1}}{\mathbf{1}^T D \mathbf{1}} \mathbf{1}$$

- ④ and then the score  $L_r$  is computed as:

$$L_r = \frac{\tilde{f}_r^T L \tilde{f}_r}{\tilde{f}_r^T D \tilde{f}_r}$$

- ④ This gives a ranking for the relevance of the attributes

# Notebooks

---

These two Python Notebooks show some examples of dimensionality reduction and feature selection

- ⑤ Dimensionality reduction and feature selection Notebook ([click here](#) to go to the url)
- ⑥ Linear and non-linear dimensionality reduction Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

## Similarity functions

---

- Unsupervised algorithms use similarity/distance to compare data
- This comparison will be obtained using functions of the attributes of the data
- We assume that data are embedded in a N-dimensional space where it can be defined a similarity/distance
- There are domains where this assumption is not true, so some other kind of functions will be needed to represent data relationships

⑤ The properties of a similarity function are:

1.  $s(p, q) = 1 \iff p = q$

2.  $\forall p, q \ s(p, q) = s(q, p)$

⑥ The properties of a distance function are:

1.  $\forall p, q \ d(p, q) \geq 0 \text{ and } \forall p, q \ d(p, q) = 0 \iff p = q$

2.  $\forall p, q \ d(p, q) = d(q, p)$

3.  $\forall p, q, r \ d(p, r) \leq d(q, p) + d(p, r)$

- ④ Minkowski metrics (Manhattan, Euclidean)

$$d(i, k) = \left( \sum_{j=1}^d |x_{ij} - x_{kj}|^r \right)^{\frac{1}{r}}$$

- ④ Mahalanobis distance

$$d(i, k) = (x_i - x_k)^T \cdot \varphi^{-1} \cdot (x_i - x_k)$$

where  $\varphi$  is the matrix covariance of the attributes

◎ **Chebyshev Distance**

$$d(i, k) = \max_j |x_{ij} - x_{kj}|$$

◎ **Camberra distance**

$$d(i, k) = \sum_{j=1}^d \frac{|x_{ij} - x_{kj}|}{|x_{ij}| + |x_{kj}|}$$

## ⑤ Cosine similarity

$$d(i, k) = \frac{x_i^T \cdot x_j}{\|x_i\| \cdot \|x_j\|}$$

## ⑥ Pearson correlation measure

$$d(i, k) = \frac{(x_i - \bar{x}_i)^T \cdot (x_j - \bar{x}_j)}{\|x_i - \bar{x}_i\| \cdot \|x_j - \bar{x}_j\|}$$

⑤ **Coincidence coefficient**

$$s(i, k) = \frac{a_{00} - a_{11}}{d}$$

⑥ **Jaccard coefficient**

$$s(i, k) = \frac{a_{11}}{a_{00} + a_{01} + a_{10}}$$

This Python Notebook shows examples of using different distance functions

- ⑤ Distance functions Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# Unsupervised Learning

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Unsupervised Learning

---

- Learning can be done in a supervised or unsupervised way
- There is a strong bias in the machine learning community towards supervised learning
- But a lot of concepts are learned unsupervisedly
- The discovery of new concepts is always unsupervised

- ① We assume that data is embedded in a N-dimensional space with a similarity/dissimilarity function
- ② Similarity defines how examples are related to each other
- ③ **Bias:**
  - Examples are more related to the nearest examples than to the farthest
  - Patterns are compact groups that are maximally separated from each other
- ④ **Areas:** Statistics, machine learning, graph theory, fuzzy theory, physics

- Discovery goals:
  - **Summarization:** To obtain representations that describe an unlabeled dataset
  - **Understanding:** To discover the concepts inside the data
- Difficult tasks because discovery is biased by context
  - Different answers could be valid depending of the discovery goal or the domain
  - There are few criteria to validate the results
- **Representation of the clusters:** Unstructured (partitions) or relational (hierarchies)

## Hierarchical algorithms

- Examples are organized as a binary tree
- Based on the relationship among examples defined by similarity/dissimilarity functions
- No explicit division in groups, has to be chosen a posteriori

## Partitional algorithms

- Only a partition of the dataset is obtained
- Based on the optimization of a criteria (assumptions about the characteristics of the cluster model)

# Hierarchical Algorithms

---

⑤ **Based on graph theory**

- The examples form a full connected graph
- Similarity defines the length of the edges
- Clustering is decided using a connectivity criteria

⑥ **Based on matrix algebra**

- A distance matrix is calculated from the examples
- Clustering is computed using the distance matrix
- The distance matrix is updated after each iteration (different updating criteria)

## ⑤ Graphs

- Single Linkage, Complete Linkage, MST
- Divisive, Agglomerative

## ⑥ Matrices

- Johnson algorithm
- Different update criteria (S-L, C-L, Centroid, minimum variance)

### Computational cost

From  $O(n\_inst^3 \times n\_dims)$  to  $O(n\_inst^2 \times n\_dims)$

**Algorithm:** Agglomerative graph algorithm

Compute distance/similarity matrix

**repeat**

    Find the pair of examples with smallest similarity

    Add an edge to the graph corresponding to this pair

**if** *Agglomeration criteria holds* **then**

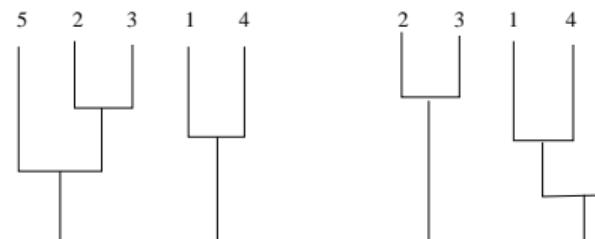
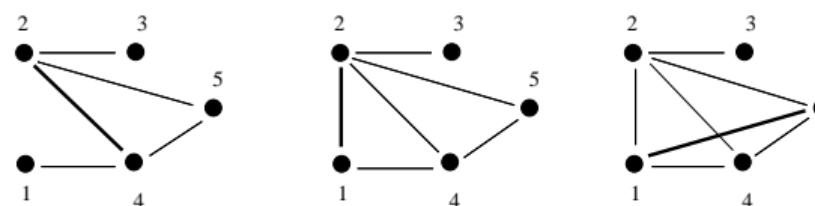
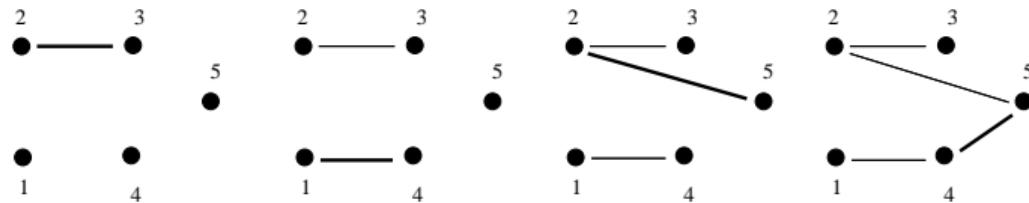
        Merge the clusters the pair belongs to

**until** *Only one cluster exists*

---

- **Single linkage** = New edge is between two disconnected graphs
- **Complete linkage** = New edge creates a clique with all the nodes of both subgraphs

	2	3	4	5
1	6	8	2	7
2		1	5	3
3			10	9
4				4



Single Link

Complete Link

**Algorithm:** Agglomerative Johnson algorithm

Compute Distance/similarity matrix

**repeat**

    Find pair of groups/examples with the smallest similarity

    Merge the pair of groups/examples

    Delete the rows and columns corresponding to the pair

    Add a new row and column with the new distances for the new group

**until** Matrix has one element

---

- Single linkage = Distance between the closest examples
- Complete linkage = Distance between the farthest examples
- Average linkage = Distance between group centroids

	2	3	4	5
1	6	8	2	7
2		1	5	3
3			10	9
4				4

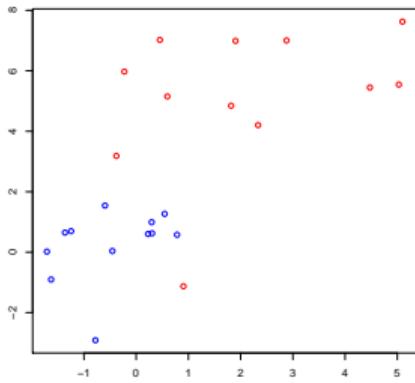
	2,3	4	5
1	7	2	7
2,3		7.5	6
4			4

	1,4	5
2,3	7.25	6
1,4		5.5

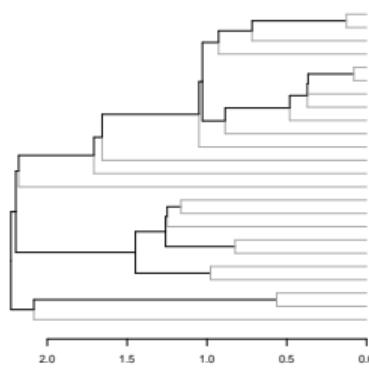
	1,4,5
2,3	6.725

- A partition of the data has to be decided a posteriori
- Some undesirable and strange behaviours could appear (chaining, inversions, breaking large clusters)
- Some algorithms have problems when with different sized and convex shaped clusters appear in the data
- Dendograms are not a practical representation for large amounts of data
- Computational cost is too high for large datasets
  - Time is  $O(n^2)$  in the best case,  $O(n^3)$  in general

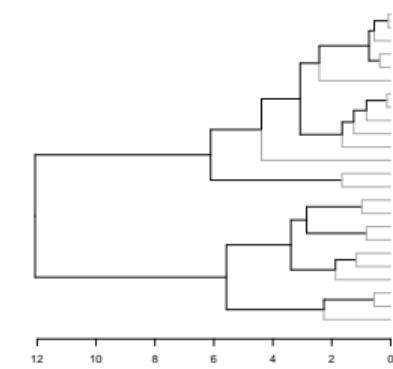
## Data



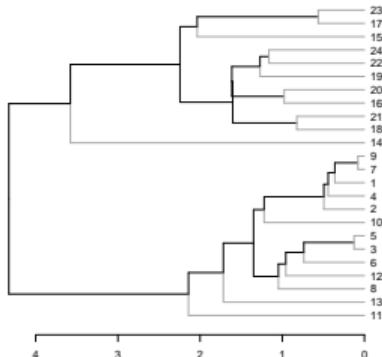
## Single Link



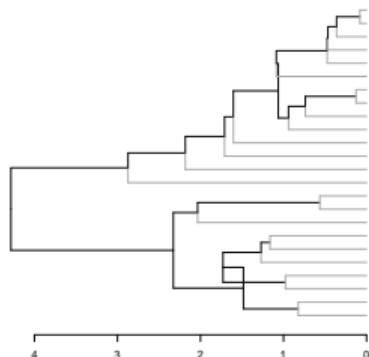
## Complete Link



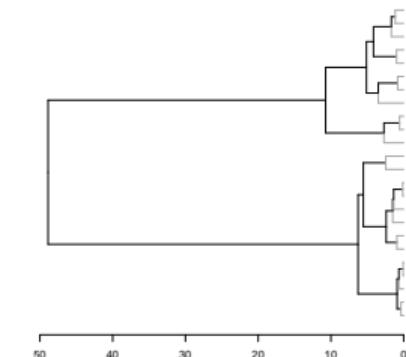
## Median



## Centroid



## Ward



This Python Notebook shows examples of using different hierarchical clustering algorithms

- ⑤ Hierarchical Clustering Algorithms Notebook ([click here](#) to go to the url)

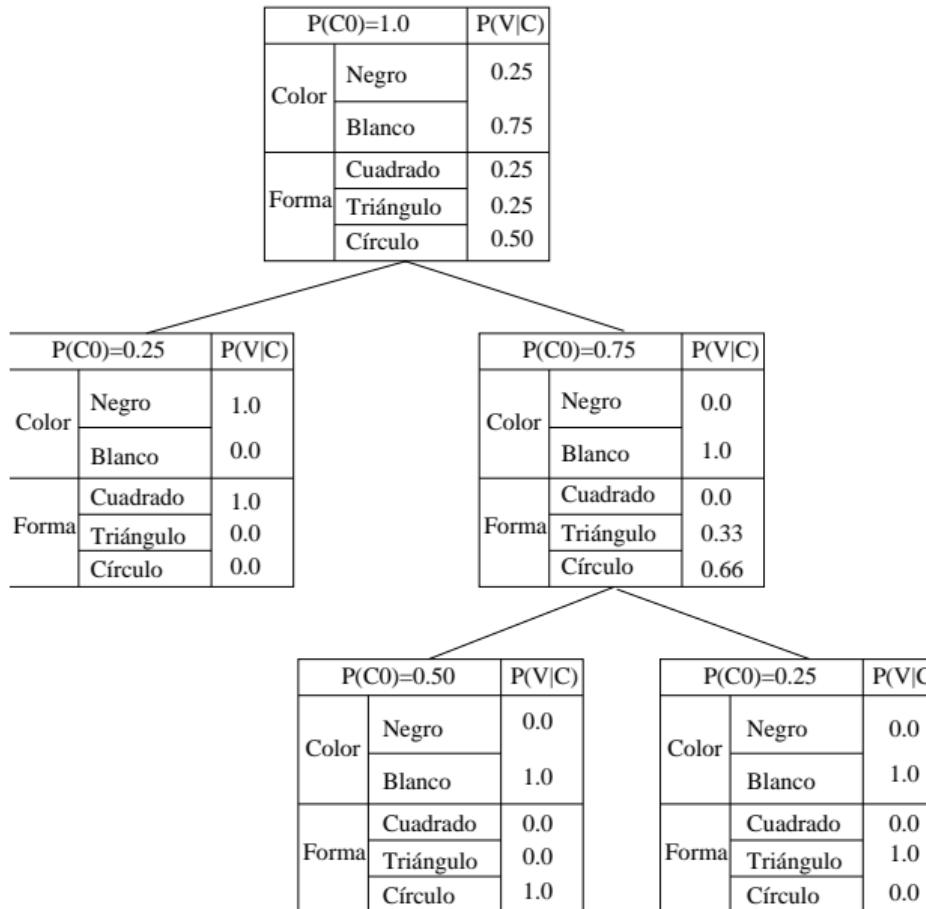
If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

- ④ Learning has an **incremental** nature (experience is acquired from continuous observation, not at once)
- ④ Concepts are learned at the same time as their **relationships** (polithetic hierarchies of concepts)
- ④ Learning is a search in the **space of hierarchies**
- ④ An objective function measures the **utility** of the structure
- ④ The updating of the structure is performed by a set of **conceptual operators**
- ④ The result **depends on the order** of the examples

JH Gennari, P Langley, D Fisher, **Models of incremental concept formation**,  
Artificial intelligence, 1989

- Based on ideas from cognitive psychology
  - Learning is incremental
  - Concepts are organized in a hierarchy
  - Concepts are organized around a prototype and described probabilistically
  - Hierarchical concept representation is modified via cognitive operators
- Builds a hierarchy top/down
- Four conceptual operators
- Heuristic measure to find the *basic level* (Category utility)

## Probabilistic hierarchy

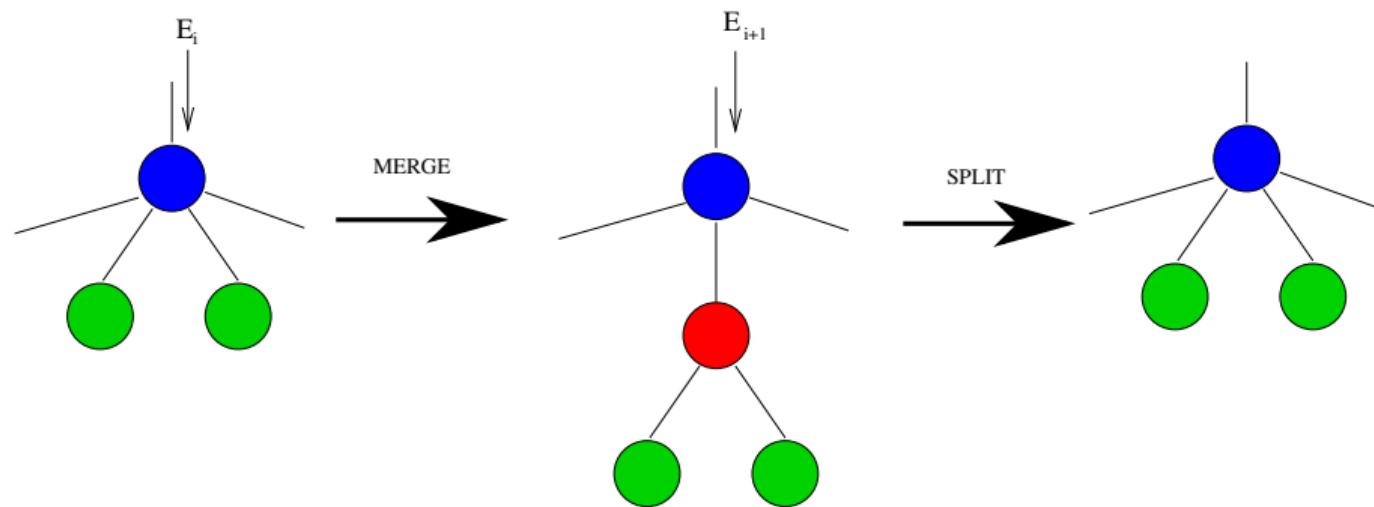


- ④ Category utility balances:
  - Intra class similarity:  $P(A_i = V_{ij} | C_k)$
  - Inter class similarity:  $P(C_k | A_i = V_{ij})$
- ④ It measures the difference between a partition of the data and no partition at all
- ④ For qualitative attributes and  $k$  categories  $\{C_1, \dots, C_k\}$  is defined as:

$$\frac{\sum_{k=1}^K P(C_k) \sum_{i=1}^I \sum_{j=1}^J P(A_i = V_{ij} | C_k)^2 - \sum_{i=1}^I \sum_{j=1}^J P(A_i = V_{ij})^2}{K}$$

(see the full derivation on the paper)

- **Incorporate:** Put the example inside an existing class
- **New class:** Create a new class at this level
- **Merge:** Two concepts are merged, and the example is incorporated inside the new class
- **Divide:** A concept is substituted by its children



---

**Procedure:** Depth-first limited search COBWEB (x: Example, H: Hierarchy)

Update the father with the new example

**if** we are in a leaf **then**

    Create a new level with this example

**else**

    Compute **CU** of incorporating the example to each class

    Save the two best **CU**

    Compute **CU** of merging the best two classes

    Compute **CU** of splitting the best class

    Compute **CU** of creating a new class with the example

    Recursive call with the best choice

## Partitional algorithms

---

To find the optimal partition of  $N$  objects in  $K$  groups is NP-hard, we need approximated algorithms

- Model/prototype based algorithms (K-means, Gaussian Mixture Models, Fuzzy K-means, Leader algorithm...)
- Density based algorithms (DBSCAN, DENCLUE...)
- Grid based algorithms (STING, CLIQUE...)
- Graph theory based algorithms (Spectral Clustering...)
- Other approaches:
  - Affinity Clustering
  - Unsupervised Neural networks
  - SVM clustering

# Model/Prototype Clustering

---

- Our model is a set of  $k$  hyperspherical clusters
- An iterative algorithm assigns each example to one of  $K$  groups ( $K$  is a parameter)
- Optimization criteria: Minimize the distance of each example to the centroid of the cluster (squared error)

$$Distortion = \sum_{k=1}^K \sum_{i \in C_k} \| x_i - \mu_k \|^2$$

- Optimization by a Hill Climbing/gradient descent search algorithm
- The algorithm converges to a local minima

---

**Algorithm:** K-means (X: Examples, k:integer)

Generate k initial prototypes (e.g. k random examples)

**repeat**

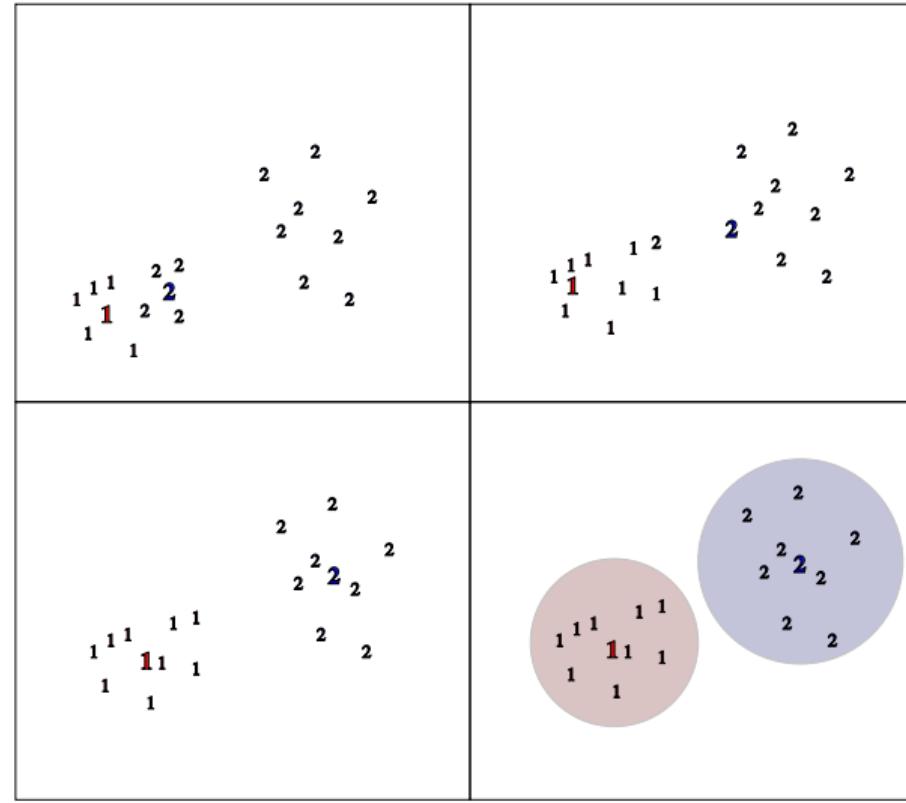
Reassign the examples to their nearest prototype

Recalculate prototypes (centroids)

**until** *no change in assignments or a number of iterations passes*

---

## K-means



- ④ The algorithm is **sensitive to initialization** (to run from several random initializations is a common practice)
- ④ Sensitive to clusters with **different sizes/densities** and **outliers**
- ④ To **find the value of  $k$**  is not a trivial problem
- ④ No guarantee about the **quality** of the solution
- ④ A solution is found even when not hyperspherical clusters exist
- ④ The spatial complexity makes it not suitable for large datasets

- ① K-means++ modifies the initialization strategy
- ② It tries to maximize distance among initial centers
- ③ **Algorithm:**
  1. Choose one center uniformly from among all the data
  2. For each data point  $x$ , compute  $d(x, c)$ , the distance between  $x$  and the nearest center already chosen
  3. Choose one new data point at random as a new center, using a weighted probability distribution where a point  $x$  is chosen with probability proportional to  $d(x, c)^2$
  4. Repeat Steps 2 and 3 until  $k$  centers have been chosen
  5. Proceed with the standard K-means algorithm

- ④ Bisecting K-means iteratively splits one of the current clusters into two until obtaining the desired number of clusters
- ④ Pros:
  - Reduces the effect of initialization
  - A hierarchy is obtained
  - It can be used to determine K
- ④ Con: Different criteria could be used to decide which cluster to split (the largest, the one with the largest variance... )

## ② Algorithm:

1. Choose a number of partitions
2. Apply K-means to the dataset with  $k=2$
3. Evaluate the quality of the current partition
4. Pick the cluster to be split using a quality criterion
5. Apply K-means to the cluster with  $k=2$
6. If the number of clusters is less than desired, repeat from step 3

- ④ Minimizes initialization dependence exploring all clusterings that can be generated using the examples as initialization points
- ④ For generating a partition with  $K$  clusters explores all the alternative partitions from 1 to  $K$  clusters.
- ④ Pro: Reduces the initialization problem/obtains all partitions from 2 to  $K$
- ④ Con: Computational cost (runs K-means  $K \times N$  times)

## ⑤ Algorithm:

- Compute the centroid of the partition with 1 cluster
- For  $C$  from 2 to  $k$ :
  - for each example  $e$ , compute K-means initialized with the  $C - 1$  centroids from the previous iteration and an additional one with  $e$  as the  $C$ -th centroid
  - Keep the clustering with the best objective function as the  $C$ -clusters solution

④ Kernel K-means:

- Distances are computed using a kernel
- **Pro:** Clusters that are non linearly separable can be discovered (non convex)
- **Con:** Centroids are in the feature space, no interpretation in the original space (image problem)

④ Fast K-means

- Use of the triangular inequality to reduce the number of distance computations for assigning examples

④ K-Harmonic means

- Uses the Harmonic mean of the squared distances instead of the distortion as objective function
- **Pro:** Less sensitive to initialization

- ⑤ K-means assumes a centroid can be computed
- ⑤ In some problems a centroid makes no sense (nominal attributes, structured data)
- ⑤ One or more examples for each cluster are maintained as a representative of the cluster (medoid)
- ⑤ The distance from each example to the medoid of their cluster is used as optimization criteria
- ⑤ **Pro:** It is not sensitive to outliers
- ⑤ **Con:** For one representative the cost per iteration is  $O(n^2)$ , for more it is NP-hard

## Partitioning Around Medoids (PAM):

1. Randomly select  $k$  of the  $n$  data points as the medoids
2. Associate each data point to the closest medoids
3. For each medoid  $m$ 
  - o For each non-medoid  $o$ : Swap  $m$  and  $o$  and compute the cost
4. Keep the best solution
5. If medoids change, repeat from step 2

- The previous algorithms need all the data from the beginning
- An incremental strategy is needed when data comes as a stream (**Leader Algorithm**):
  - A distance/similarity **threshold** ( $D$ ) **determines** the extent of a **cluster**
  - **Inside the threshold**: Incremental updating of the model (prototype)
  - **Outside the threshold**: A new cluster is created
- The threshold  $D$  determines the granularity of the clusters
- The clusters are dependent on the order of the examples

---

**Algorithm:** Leader Algorithm (X: Examples, D:double)

Generate a prototype with the first example

**while** *there are examples* **do**

    e= current example

    d= distance of e to the nearest prototype

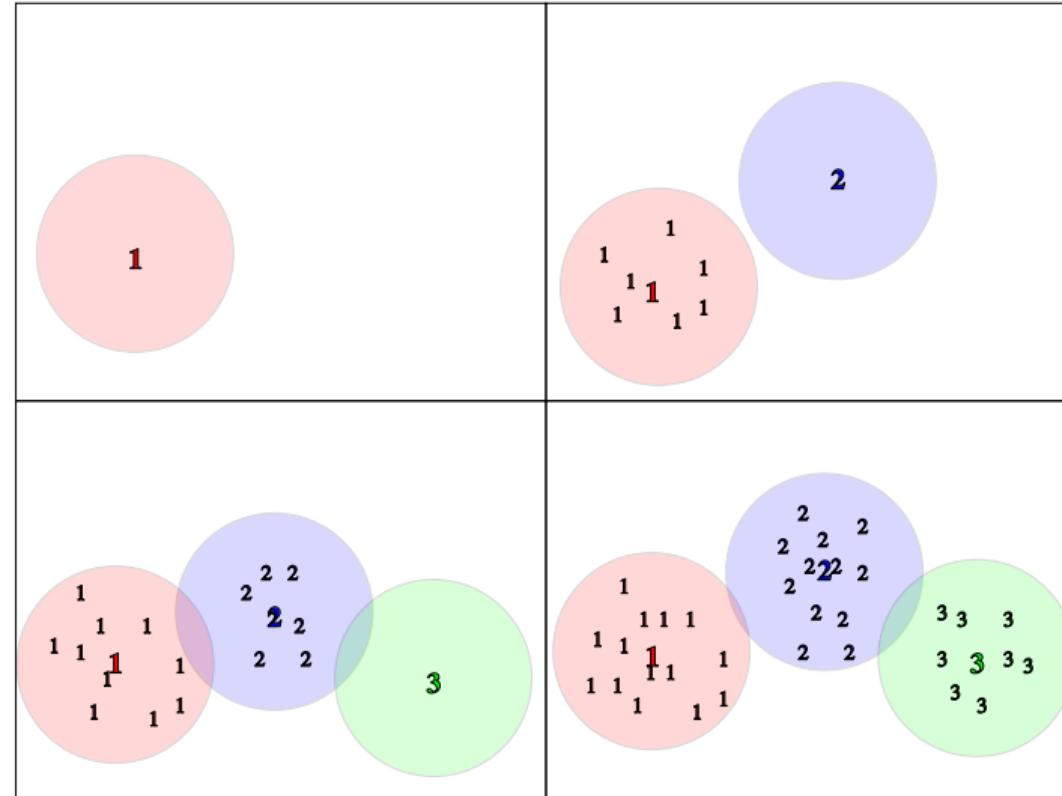
**if**  $d \leq D$  **then**

        Add the example to the cluster

        Recompute the prototype

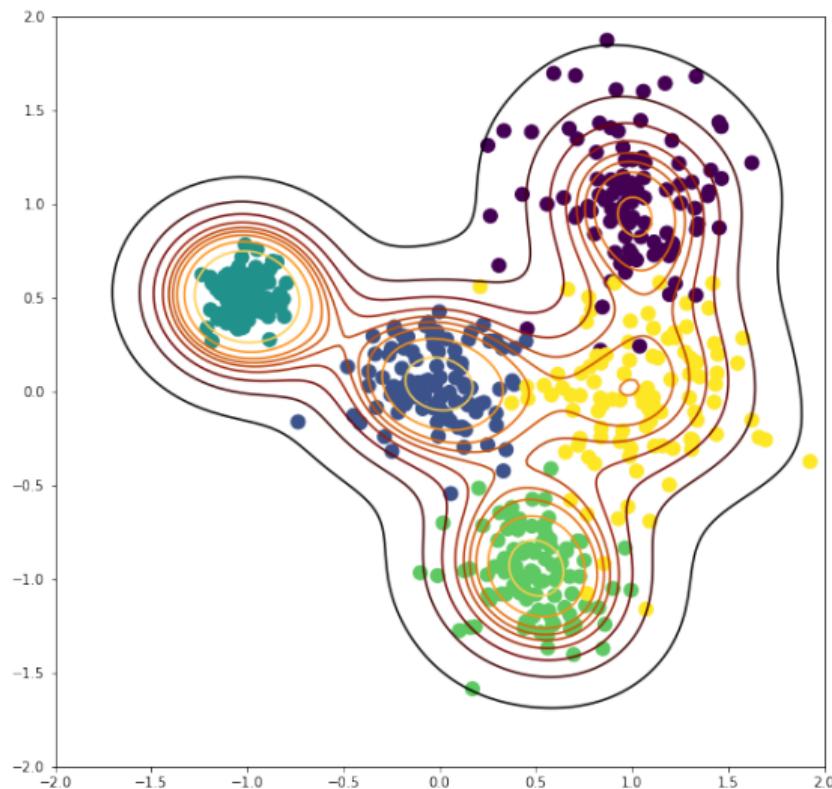
**else**

        Create a new cluster with this example



- ④ We can use a generative approach to clustering
- ④ We assume that data is a mixture of K probability distributions
- ④ The goal is to disentangle the distributions, assigning groups of the data to the distribution that generates them
- ④ Generating the groups we obtain an estimation of the parameters of the distributions
- ④ This give us a generative model that we can sample from

## Mixture Decomposition - EM algorithm



- ⑤ We assume that data are drawn from a mixture of probability distributions (usually Gaussian)
- ⑤ Search the space of parameters of the distributions to obtain the mixture that explains better the data (parameter estimation)
- ⑤ The model of the data is:

$$P(x|\theta) = \sum_{k=1}^K w_k P(x|\theta_k)$$

with  $K$  the number of clusters and  $\sum_{k=1}^K w_k = 1$ ,  $\pi_k$  represents the contribution of the distribution  $k$  to the mixture

- ⑤ Each example has a probability to belong to a cluster, **Soft partitions**

- ④ We can estimate the parameters of the mixed distribution using maximum likelihood
- ④ For the Gaussian case:

$$p(x|\theta) = \sum_{k=1}^K \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

Being  $\mu_k$  the vectors of means and  $\Sigma_k$  the covariance matrices of each Gaussian distribution

- ④ Computing the log likelihood of this distribution and equating the derivative to zero will give us the estimators for the  $\mu_k$  and  $\Sigma_k$  parameters

- The actual computations depend on the assumptions that we make about the attributes of the data and their distribution
- These assumptions result on different number of parameters to estimate and computational cost
  - Attributes are independent, so the covariance matrices are diagonal and different, but all the attributes of a component share the same variance ( $O(d)$  parameters, hyper spheres parallel to coordinate axis)
  - Attributes are independent, so the covariance matrices are diagonal, but with different variances ( $O(d)$  parameters, ellipsoids parallel to coordinate axis)
  - The same covariance matrices for all the components ( $O(d^2)$  parameters, identical hyper ellipsoids non-parallel to coordinate axis)
  - Full covariance matrix for all the components ( $O(d^2)$  parameters, hyper ellipsoids non-parallel to coordinate axis)

- There is not a closed form for the computation of the parameters, so an iterative algorithm has to be used, this is known as **Expectation-Maximization** (EM)
- The goal is to **estimate the parameters of the distribution** that describes each cluster (e.g.  $\mu$  and  $\Sigma$ )
- EM is a general algorithm that is used for the task of parameter estimation by Maximum Likelihood (it is not specific of GMM)
- The algorithm **maximizes the likelihood** of the distribution with respect to the data
- It performs iteratively two steps:
  - **Expectation:** We calculate a function that assigns to all the examples a degree of membership to all the  $K$  probability distributions
  - **Maximization:** We re-estimate the parameters of the distributions to maximize the memberships

- ⑤ For the case of  $A$  independent attributes:

$$p(x|\mu_k, \Sigma_k) = \prod_{j=1}^A \mathcal{N}(x|\mu_{kj}, \sigma_{kj})$$

- ⑥ The model to fit is

$$p(x|\mu, \sigma) = \sum_{k=1}^K \pi_k \prod_{j=1}^A \mathcal{N}(x|\mu_{kj}, \sigma_{kj})$$

we have a diagonal covariance matrix

- ⑦ We can derive the estimators of the parameters using Maximum Likelihood from the log-likelihood of  $p(x|\mu, \sigma)$  ( $\hat{\mu}$ ,  $\hat{\sigma}$  and  $\hat{\pi}$ )

- ④ The expectation step computes the weights for each example and component

$$\hat{\gamma}_{ik} = w_k P(x_i | \mu_k, \sigma_k)$$

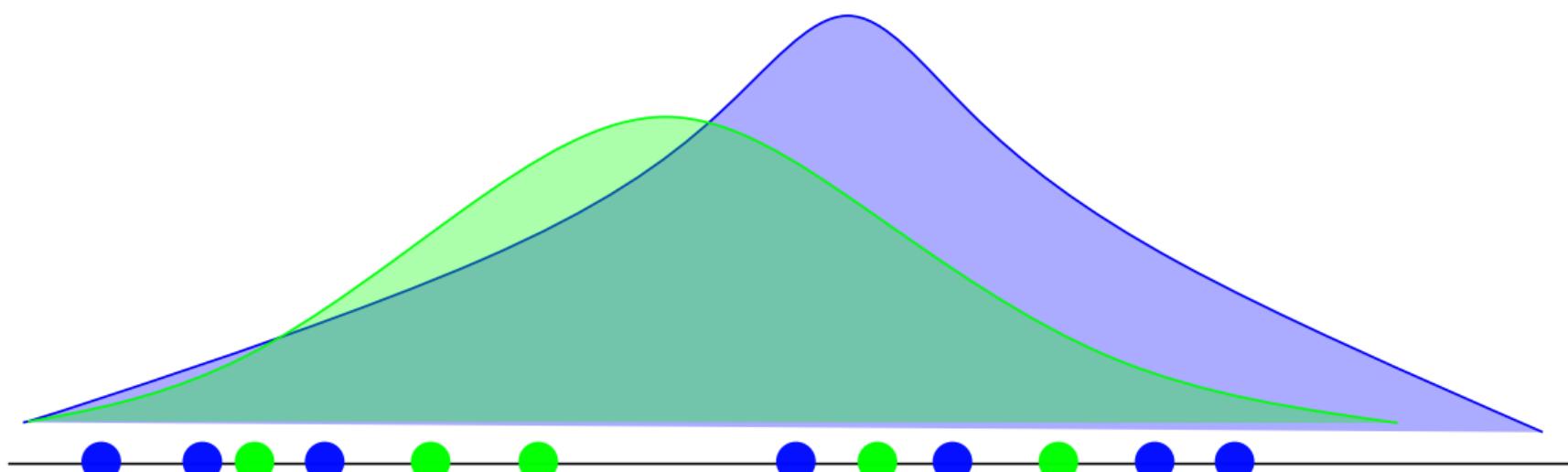
- ④ This represents the probability that an example  $x_i$  is generated by component  $C_k$

- ⑤ The maximization steps recomputes  $\mu$ ,  $\sigma$ , and  $w$  for each component proportionally to the weights computed in the expectation step

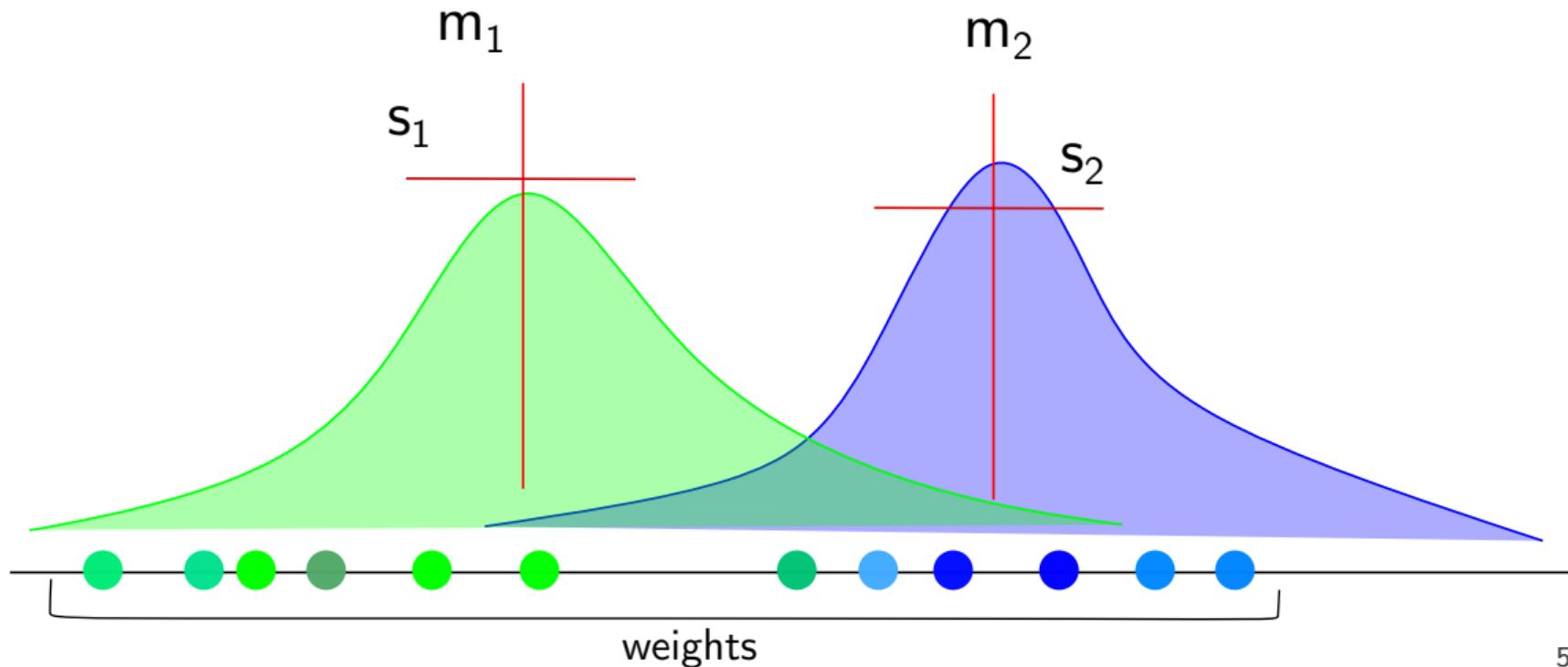
$$\begin{aligned}\hat{\mu}_k &= \frac{\sum_{i=1}^N \hat{\gamma}_{ik} x_i}{\sum_{i=1}^N \hat{\gamma}_{ik}} \\ \hat{\sigma}_k &= \frac{\sum_{i=1}^N \hat{\gamma}_{ik} (x_k - \hat{\mu}_i)^2}{\sum_{i=1}^N \hat{\gamma}_{ik}} \\ \hat{w}_k &= \frac{1}{N} \sum_{k=1}^N \hat{\gamma}_{ik}\end{aligned}$$

- ① K initial distributions are generated  $\mathcal{N}(\mu_k, \sigma_k)$ , defining their parameters  $\mu_k$ , and  $\sigma_k$ , K-means can be used as initial estimate
- ② Repeat until convergence (no log-likelihood improvement):
  1. **Expectation:** Compute the membership of each example to each mixture component
    - Each instance will have a weight ( $\hat{\gamma}_{ik}$ , **responsibility**) computed from the components computed by the previous iteration that indicate how likely the example is from a component
  2. **Maximization:** Recompute the parameters estimators using the weights from the previous step to obtain the new  $\hat{\mu}$ ,  $\hat{\sigma}$ , and  $\hat{\pi}$  for each distribution. Recompute the log-likelihood.
- ③ Finally, each example has a probability to each component that corresponds to the responsibility (soft assignment)

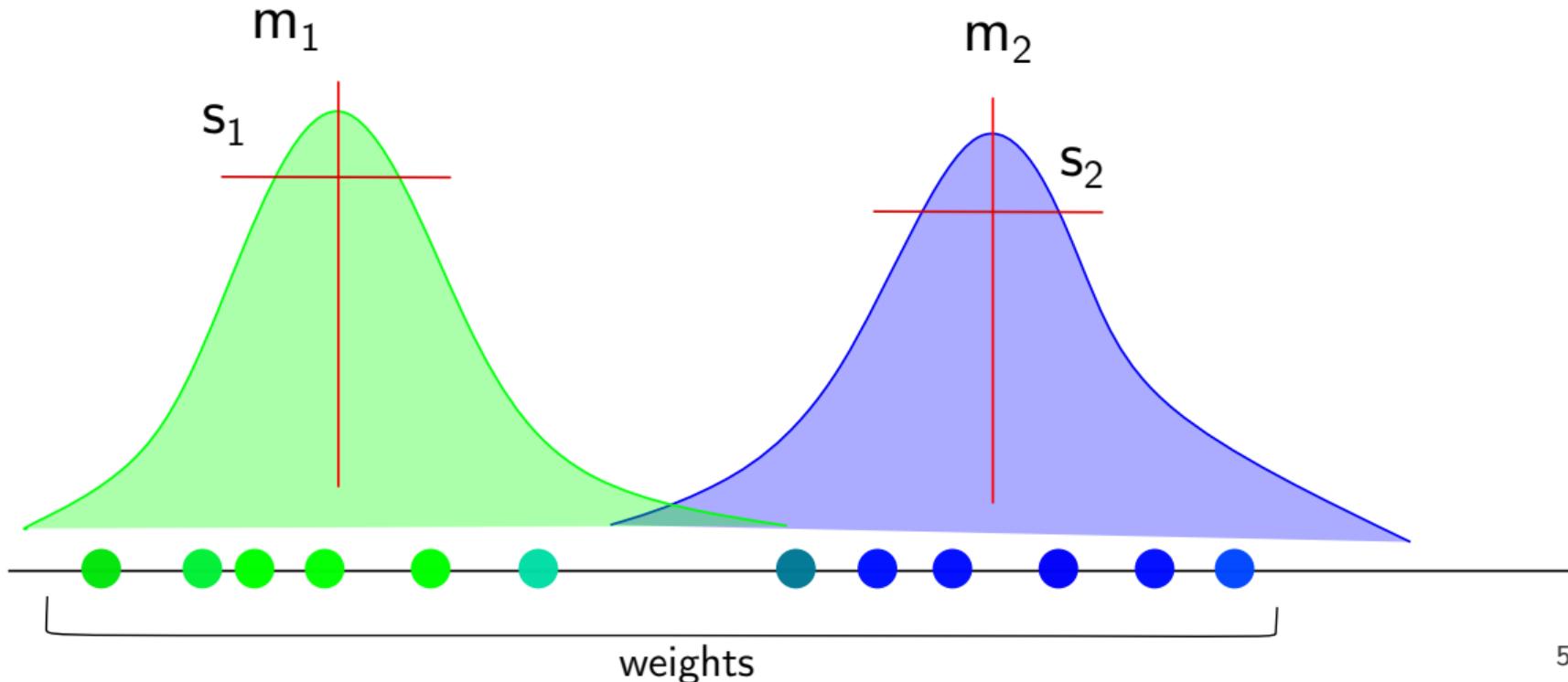
## Initial Assignment



Expectation + Maximization = new parameters

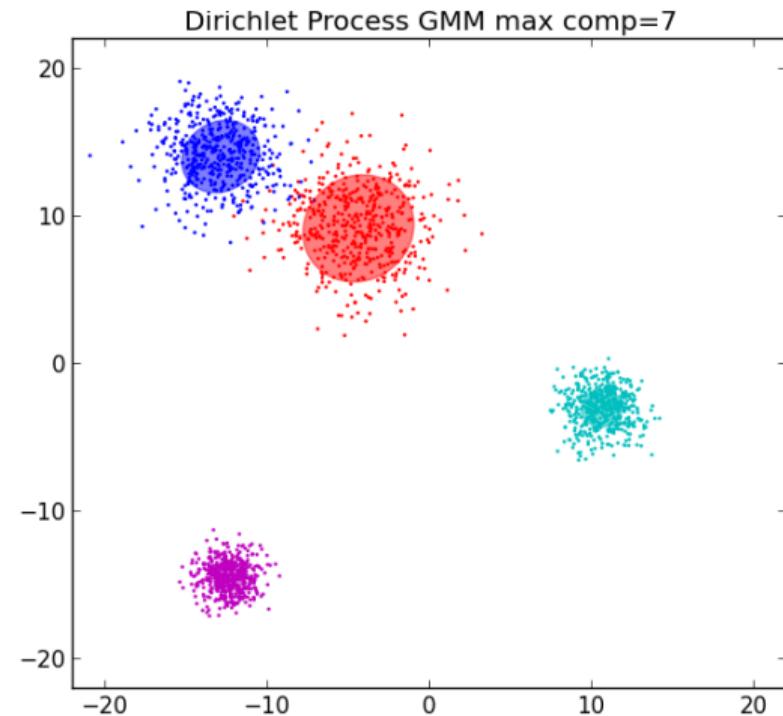
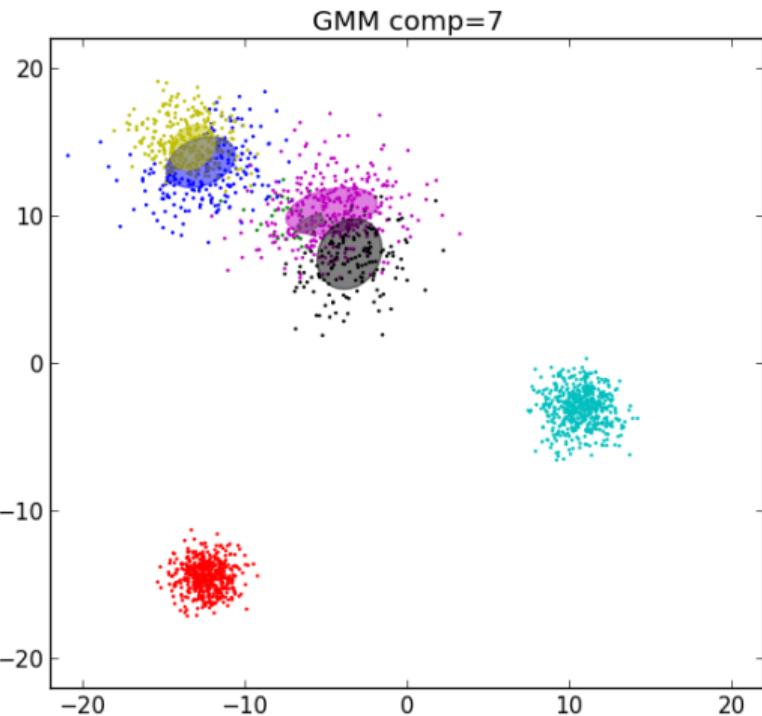


Expectation + Maximization = new parameters



- ⑤ K-means is a particular case of this algorithm (hard partition)
- ⑥ The main advantage is that we obtain a membership as a probability (soft assignments)
- ⑦ Using different probability distributions we can find different kinds of structures in the data
- ⑧ For each probability model we use we need to derive the calculations for the iterative updating of their parameters

- One of the problems of GMM is to decide a priori the number of components
- This can be included in the model following a bayesian approach using a mixture model that includes as priors for the weights of the components a Dirichlet Process distribution (represents the distribution of the number of mixtures and their weights)
- Dirichlet Process distribution assumes an unbound number of components
- A finite weight is distributed among all the components
- The fitting of the model will optimize the weights of the components, and some can be reduced to 0, eliminating some of them



- ⑤ Fuzzy clustering relax the hard partition constraint of K-means
- ⑤ Each example has a **continuous membership** to each partition
- ⑤ A new optimization function is introduced:

$$L = \sum_{i=1}^N \sum_{k=1}^K \delta(C_k, x_i)^b \|x_i - \mu_k\|^2$$

with  $\sum_{k=1}^K \delta(C_k, x_i) = 1$ , and  $b$  is a blending factor

- ⑤ When the clusters are overlapped this is an advantage over hard partition algorithms

- ④ C-means is the most known fuzzy clustering algorithm, it is the fuzzy version of K-means
- ④ Membership is computed as the normalized inverse distance to all the clusters
- ④ The updating of the cluster centers is computed as:

$$\mu_j = \frac{\sum_{i=1}^N \delta(C_j, x_i)^b x_i}{\sum_{i=1}^N \delta(C_j, x_i)^b}$$

- ④ And the updating of the memberships:

$$\delta(C_j, x_i) = \frac{(1/d_{ij})^{1/(1-b)}}{\sum_{k=1}^K (1/d_{ik})^{1/(1-b)}}, d_{ij} = \|x_i - \mu_j\|^2$$

Notice that this is exactly a **softmax** of the distances to the centroids

- ⑤ The C-means algorithm looks for spherical clusters, other alternatives:
  - **Gustafson-Kessel algorithm:** A covariance matrix is introduced for each cluster in the objective function that allows ellipsoid shapes and different cluster sizes
  - **Gath-Geva algorithm:** Adds to the objective function the size, and an estimation of the density of the cluster
- ⑥ Different objective functions can be used to detect specific shapes in the data (lines, rectangles...)

This Python Notebook shows examples of using different the K-means and GMM and their problems

- ⑤ Prototype Based Clustering Algorithms Notebook ([click here](#) to go to the url)

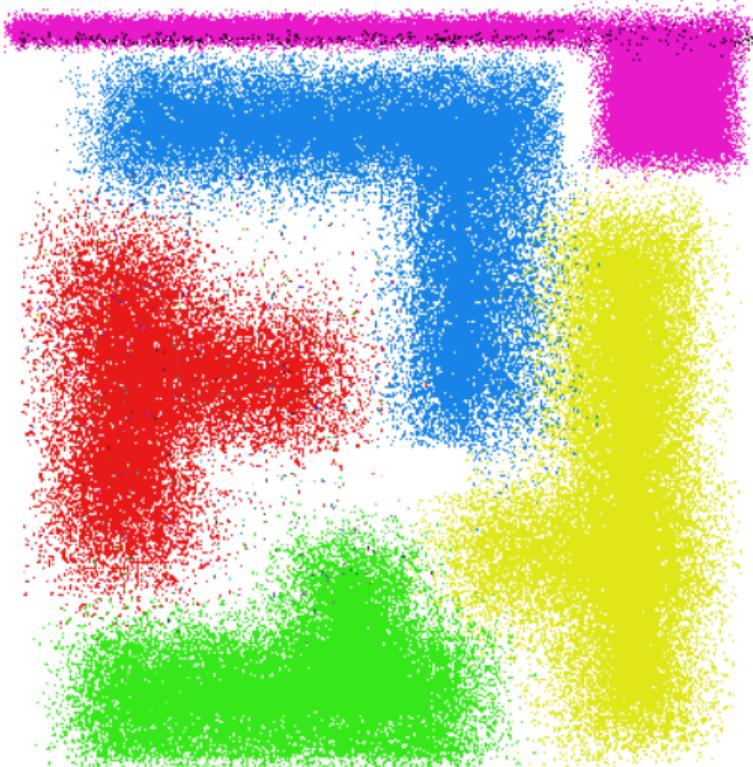
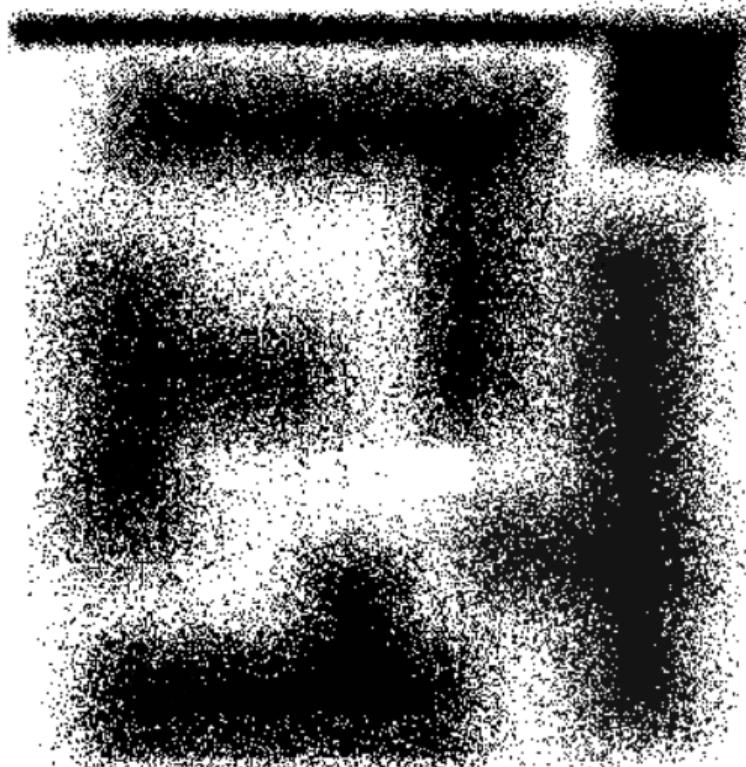
If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

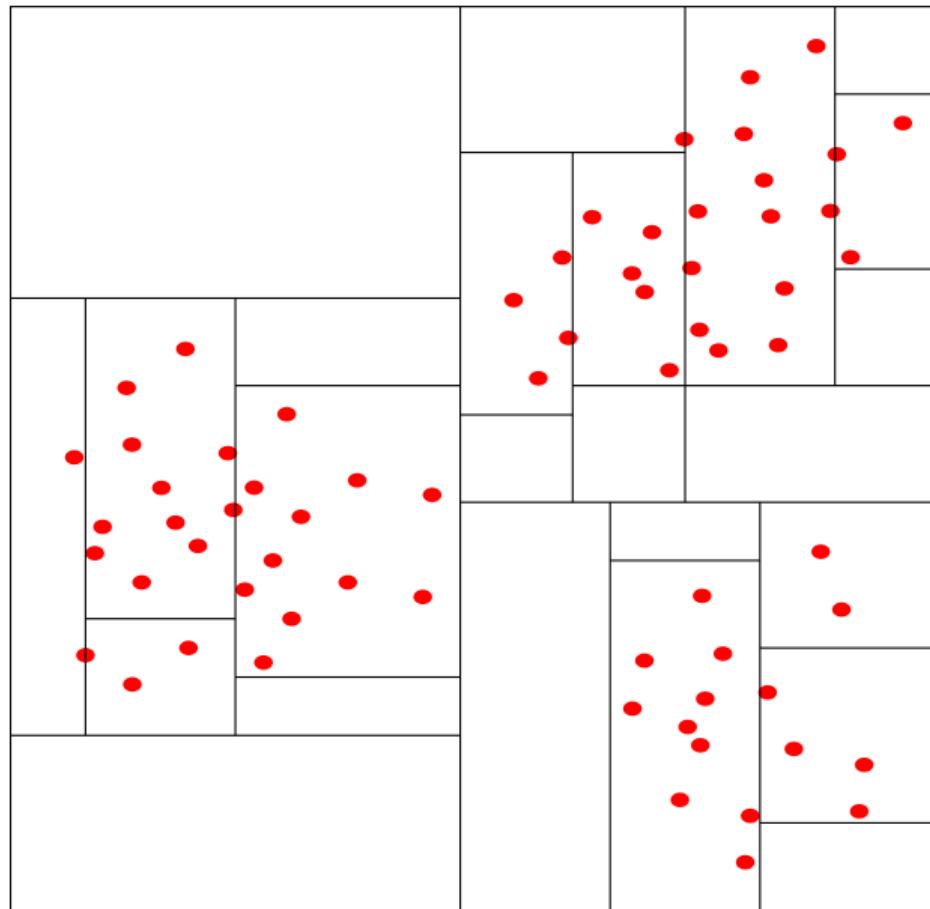
## Density/Grid Clustering

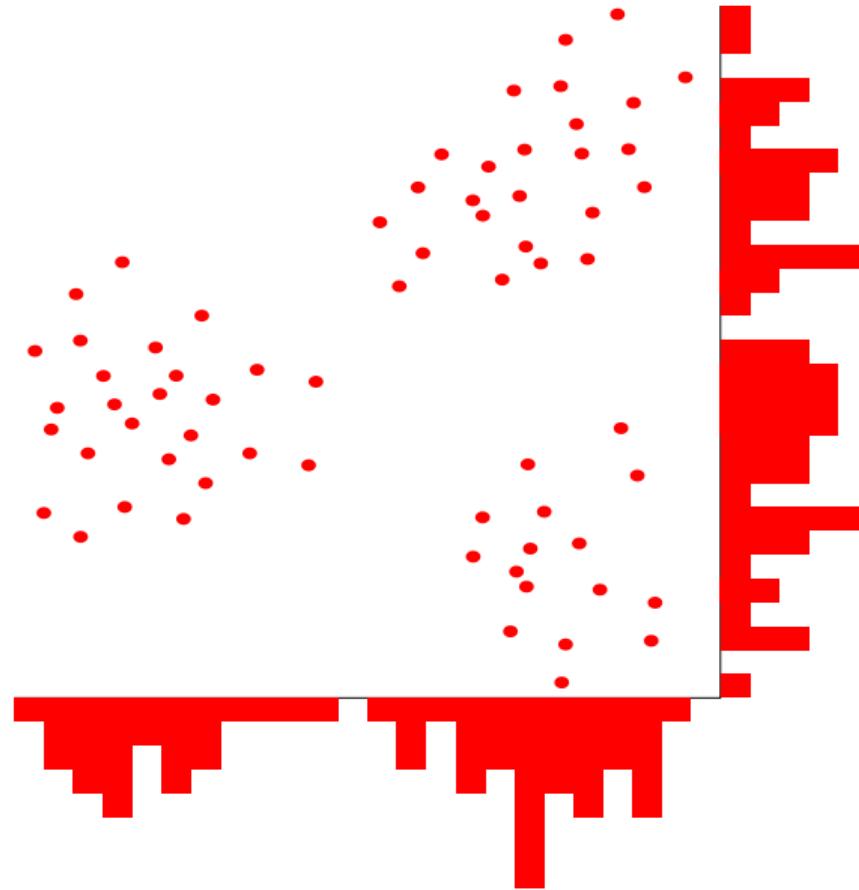
---

- The number of clusters is not decided beforehand
- We are looking for regions with high density of examples
- We are no limited to predefined shapes (there is no model)
- Different approaches:
  - Density estimation
  - Grid partitioning
  - Multidimensional histograms
- Usually applied to datasets with low dimensionality

## Density estimation







Ester, Kriegel, Sander, Xu **A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise (DBSCAN) (1996)**

Ankerst, Breunig, Kriegel, Sander **OPTICS: Ordering Points To Identify the Clustering Structure (2000)**

- Used in spatial databases, but can be applied to data with more dimensionality
- Based on finding areas of high density, it finds arbitrary shapes

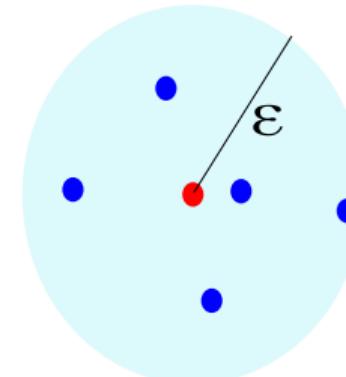
- ④ We define  **$\varepsilon$ -neighbourhood**, as the examples that are at a distance less than  $\varepsilon$  to a given instance

$$N_\varepsilon(x) = \{y \in X | d(x, y) \leq \varepsilon\}$$

- ④ We define **core point** as the examples that have a certain number of elements in  $N_\varepsilon(x)$

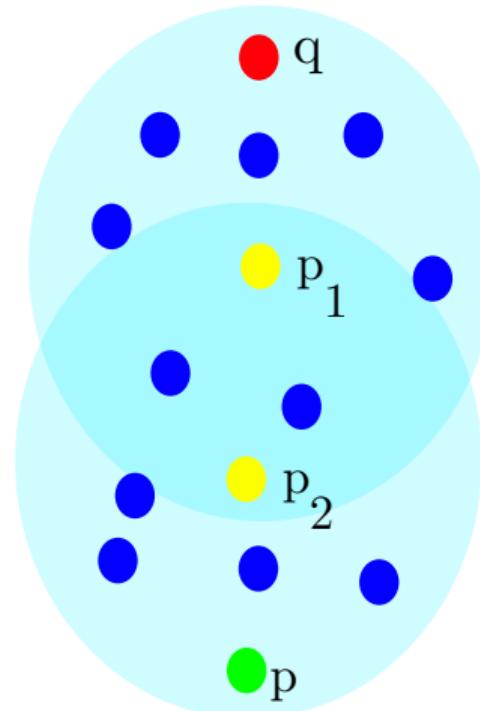
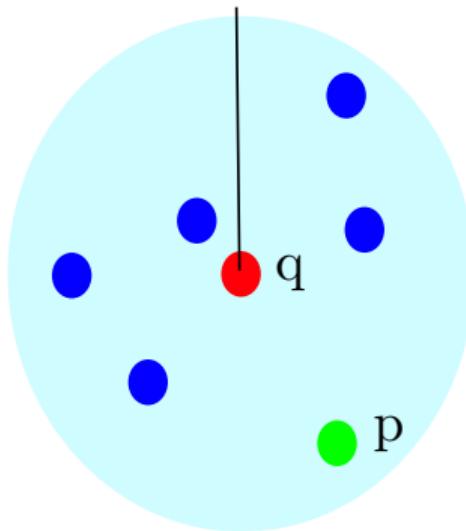
$$\text{Core\_point} \equiv |N_\varepsilon(x)| \geq \text{MinPts}$$

$\varepsilon$ -neighborhood



- Two examples  $p$  and  $q$  are **Direct Density Reachable** with respect to  $\varepsilon$ , and  $MinPts$  if:
  1.  $p \in N_\varepsilon(q)$
  2.  $|N_\varepsilon(q)| \geq MinPts$
- Two examples  $p$  and  $q$  are **Density Reachable** if there is a sequence of examples  $p = p_1, p_2, \dots, p_n = q$  where for all  $p_i, p_{i+1}$  is **Direct Density Reachable** from  $p_i$
- Two examples  $p$  and  $q$  are **Density connected** if there is an example  $o$  such that both  $p$ , and  $q$  are **Density Reachable** from  $o$

**p DR q**



## Cluster

Given a dataset  $D$ , a cluster  $C$  with respect  $\varepsilon$  and  $MinPts$  is any subset of  $D$  that:

1.  $\forall p, q \in C \wedge density\_reachable(q, p) \longrightarrow q \in C$
2.  $\forall p, q \in C \ density\_connected(p, q)$

Any **example that can not be connected** using these relationships is treated as **noise**

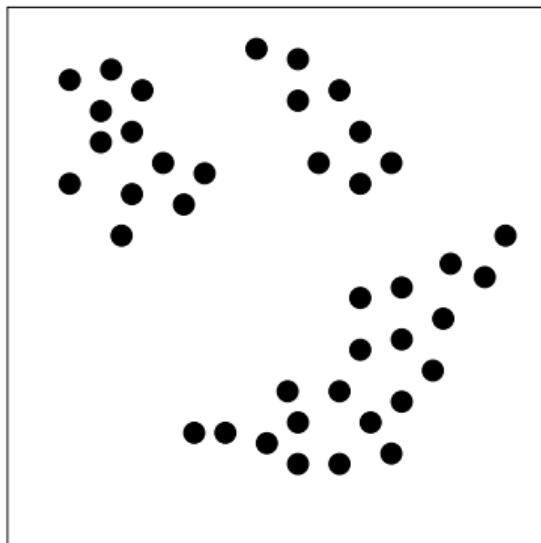
## Algorithm

1. We start with an arbitrary example and compute all density reachable examples with respect  $\varepsilon$  and  $MinPts$ .
2. If it is a core point we will obtain a group, otherwise, it is a border point and we will start from other unclassified instance

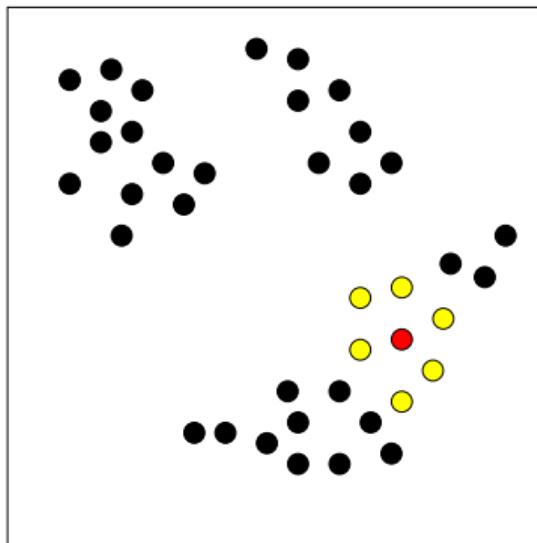
To decrease the computational cost R\* trees are used to store and compute the neighborhood of instances

$\varepsilon$  and  $MinPts$  are set from the thinnest cluster

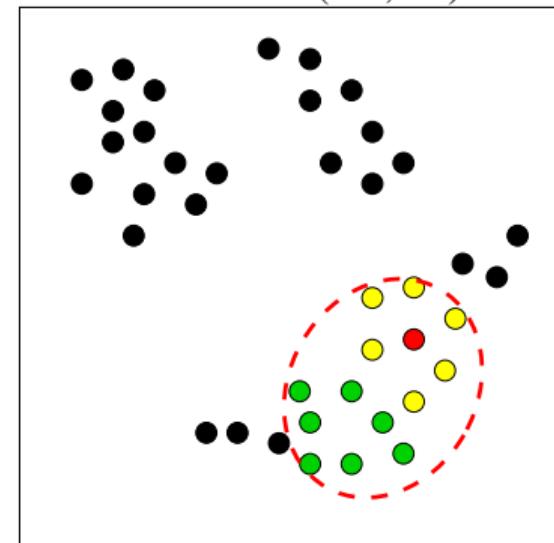
Datos Epsilon, MinPts=5



First Iteration (DDR)



First Iteration (DR,DC)



- The OPTICS algorithm is an improvement over DBSCAN to detect automatically the clusters in the data
- Two distances are defined used to compute the clusters that can be obtained for any value less than a given  $\varepsilon$ .
  - The **core distance** of an example, that is the smallest distance of a core point to the closest example in its  $\varepsilon$  neighborhood
  - The **reachability distance** between two examples  $p$  and  $o$  (core point), the smallest distance such  $p$  is directly density reachable from  $o$
- This information can also be used to order the examples by cluster and reachability distance obtaining a **reachability plot**
- The assignments can be decided using a parameter  $\xi$  that defines how much must change the reachability distance between two consecutive examples to consider that we have reached the border of the cluster

This Python Notebook compares Prototype Based and Density Based Clustering algorithm

- ⑤ Density Based Clustering Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

## Other Approaches

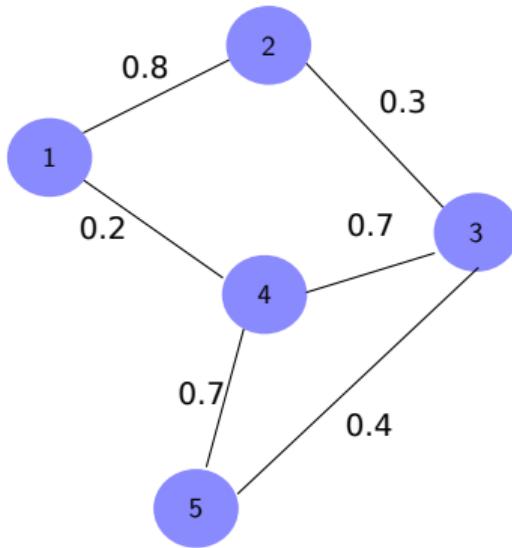
---

- ④ Spectral Graph Theory defines properties that hold the eigenvalues and eigenvectors of the adjacency matrix or Laplacian matrix of a graph
- ④ Spectral clustering uses the spectral properties of the similarity matrix
- ④ The distance matrix represents the graph that connects the examples
  - Complete graph
  - Neighborhood graph (different definitions)
- ④ Different clustering algorithms can be defined from the diagonalization of this matrix

- We start with the similarity matrix ( $W$ ) of the data
- The degree of a vertex is defined as:

$$d_i = \sum_{j=1}^n w_{ij}$$

- We define the degree matrix  $D$  as the diagonal matrix with values  $d_1, d_2, \dots, d_n$
- We can define different Laplace matrices:
  - Unnormalized:  $L = D - W$
  - Normalized:  $L_{sym} = D^{-1/2} L D^{-1/2}$  or also  $L_{rw} = D^{-1} L$



$$W = \begin{pmatrix} 0 & 0.8 & 0 & 0.2 & 0 \\ 0.8 & 0 & 0.3 & 0 & 0 \\ 0 & 0.3 & 0 & 0.7 & 0.4 \\ 0.2 & 0 & 0.7 & 0 & 0.7 \\ 0 & 0 & 0.4 & 0.7 & 0 \end{pmatrix} D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1.1 & 0 & 0 & 0 \\ 0 & 0 & 1.4 & 0 & 0 \\ 0 & 0 & 0 & 1.6 & 0 \\ 0 & 0 & 0 & 0 & 1.1 \end{pmatrix}$$

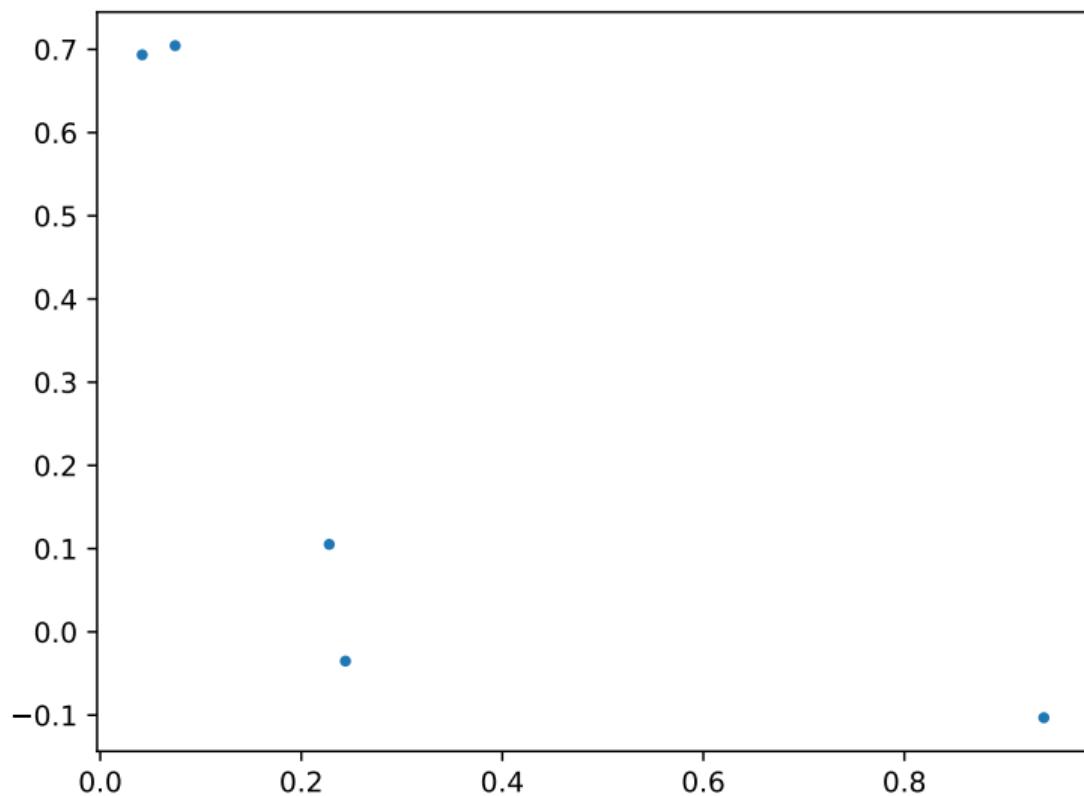
### ① Algorithm:

1. Compute the Laplace matrix from the similarity matrix
2. Compute the first K eigenvalues of the Laplace matrix
3. Use the eigenvectors as new datapoints
4. Apply K-means as clustering algorithm

② We are actually **embedding** the dataset in a space of lower dimensionality

$$L = \begin{pmatrix} 1 & -0.8 & 0 & -0.2 & 0 \\ -0.8 & 1.1 & -0.3 & 0 & 0 \\ 0 & -0.3 & 1.4 & -0.7 & -0.4 \\ -0.2 & 0 & -0.7 & 1.6 & -0.7 \\ 0 & 0 & -0.4 & -0.7 & 1.1 \end{pmatrix}$$

$$Eigvec(1, 2) = \begin{pmatrix} 0.07 & 0.70 \\ 0.04 & 0.69 \\ 0.22 & 0.10 \\ 0.93 & -0.10 \\ 0.24 & 0.03 \end{pmatrix}$$



- ④ From the similarity matrix and its Laplacian it is possible to formulate it as a **graph partitioning** problem
- ④ Given two disjoint sets of vertex  $A$  and  $B$ , we define:

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

- ④ We can partition the graph solving the mincut problem choosing a partition that minimizes :

$$\text{cut}(A_1, \dots, A_k) = \sum_{i=1}^k \text{cut}(A_i, \overline{A_i})$$

- Using directly the weights of the Laplacian not always gives good results, alternative objective functions are:

$$\text{RatioCut}(A_1, \dots, A_k) = \sum_{i=1}^k \frac{\text{cut}(A_i, \overline{A_i})}{|A_i|} \quad (1)$$

$$\text{Ncut}(A_1, \dots, A_k) = \sum_{i=1}^k \frac{\text{cut}(A_i, \overline{A_i})}{\text{vol}(A_i)} \quad (2)$$

- Where  $|A_i|$  is the size of the partition and  $\text{vol}(A_i)$  is the sum of the degrees of the vertex in  $A_i$

- ④ Affinity clustering is a **message passing** algorithm related to graph partitioning and belief propagation in probabilistic graphical models
- ④ It chooses a set of examples as the **cluster prototypes** and computes how the rest of examples are attached to them
- ④ Each pair of examples have a **similarity** defined  $s(i, k)$
- ④ Each example has a value  $r(k, k)$  that represents the **preference** for each point to be an exemplar
- ④ The algorithm **does not set apriori the number of clusters**,

- The examples pass two kind of messages
  - **Responsibility**  $r(i, k)$ , this is a message that an example  $i$  passes to the candidate to exemplars  $k$  of the point. This represents the evidence of how good is  $k$  for being the exemplar of  $i$
  - **Availability**  $a(i, k)$ , sent from candidate to exemplar  $k$  to point  $i$ . It represents the accumulated evidence of how appropriate would be for point  $i$  to choose point  $k$  as its exemplar

- ① All availabilities are initialized to 0
- ② The responsibilities are updated as:

$$r(i, k) = s(i, k) - \max_{k' \neq k} \{a(i, k') + s(i, k')\}$$

- ③ The availabilities are updated as:

$$a(i, k) = \min\{0, r(k, k) + \sum_{i' \notin \{i, k\}} \max(0, r(i', k))\}$$

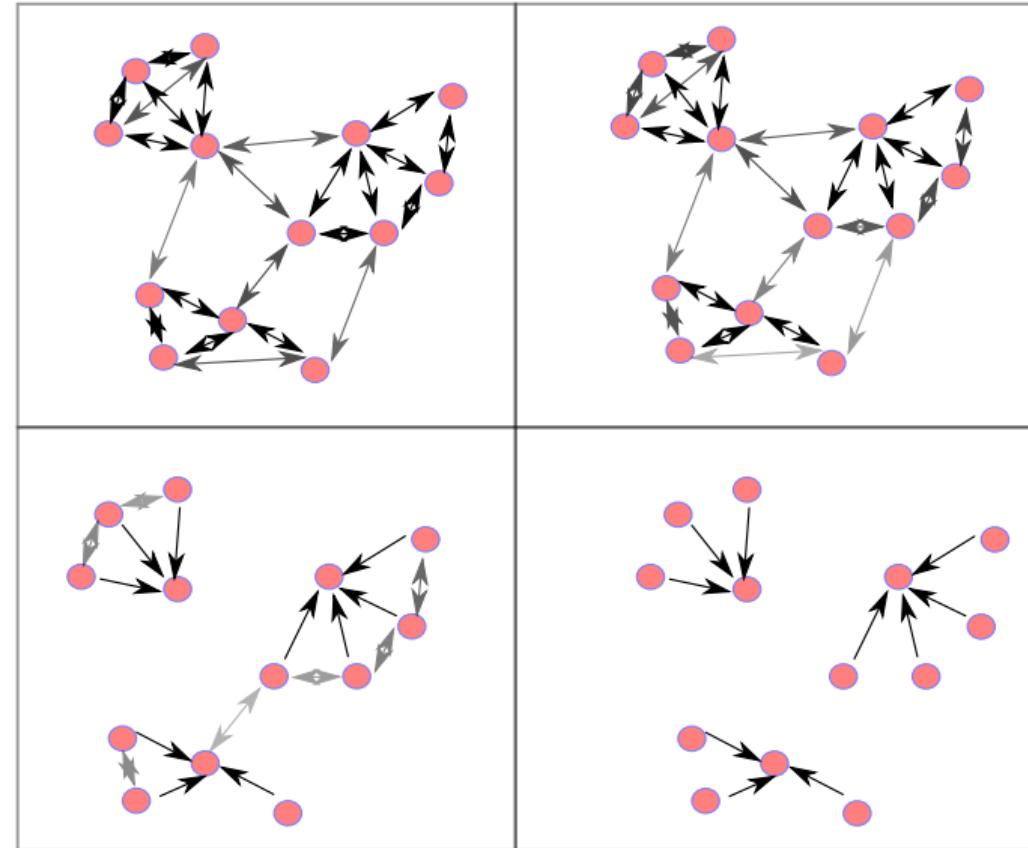
- ⑤ The self availability  $a(k, k)$  is updated as:

$$a(k, k) = \sum_{i' \neq k} \max(0, r(i', k))\}$$

- ⑥ The exemplar for a point is identified by the point that maximizes  $a(i, k) + r(i, k)$ , if this point is the same point, then it is an exemplar.

1. Update the responsibilities given the availabilities
2. Update the availabilities given the responsibilities
3. Compute the exemplars
4. Terminate if the exemplars do not change in a number of iterations

## Affinity Propagation Clustering - Algorithm



- Self-organizing maps are unsupervised neural networks
- Can be seen as an on-line constrained version of K-means
- The data is transformed to fit in a 1-d or 2-d regular mesh
- The nodes of this mesh are the prototypes
- This algorithm can be used as a dimensionality reduction method (from  $N$  to 2 dimensions)

- To build the map we have to decide the size and shape of the mesh (rectangular/hexagonal)
  - Each node a multidimensional prototype of  $p$  features
- 

### Algorithm: Self-Organizing Map algorithm

Initial prototypes are distributed regularly on the mesh

**for** *Predefined number of iterations* **do**

**foreach** *Example*  $x_i$  **do**

        Find the nearest prototype ( $m_j$ )

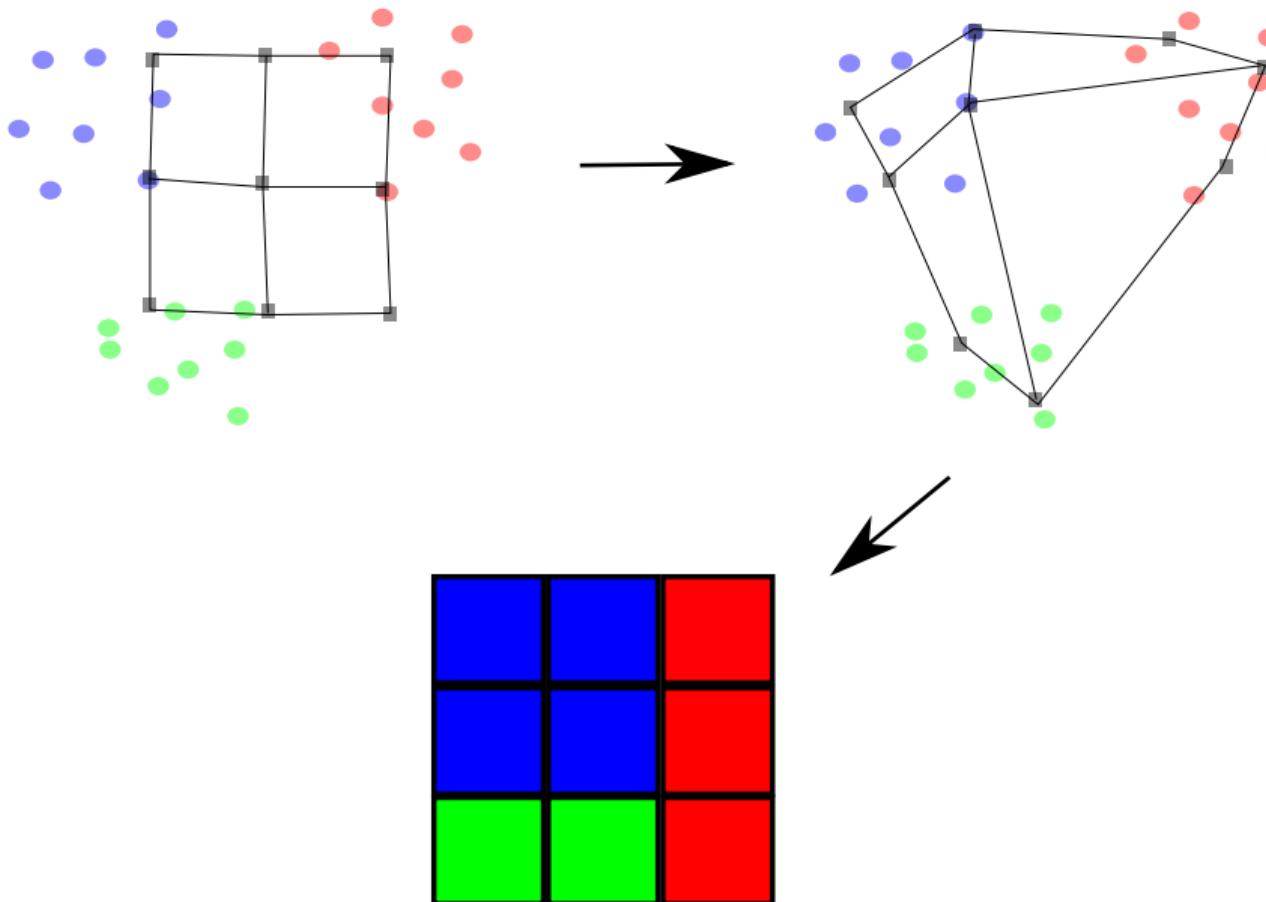
        Determine the neighborhood of  $m_j$  ( $\mathcal{M}$ )

**foreach** *Prototype*  $m_k \in \mathcal{M}$  **do**

$m_k = m_k + \alpha(x_i - m_k)$

- ⑤ Each iteration **transforms the mesh** closer to the data, maintaining the 2D relationship between prototypes
- ⑥ The neighborhood of a prototype is defined by the adjacency of the cells and the distance of the prototypes
- ⑦ Performance depends on the learning rate  $\alpha$ , that is usually decreased from 1 to 0 during the iterations
- ⑧ The number of neighbors used in the update is decreased during the iterations from a predefined number to 1
- ⑨ Some variations of the algorithm use the distance of the prototypes as weights for the update

## Self-Organizing Maps



This Python Notebook has examples for Spectral and Affinity Propagation Clustering

- ⑤ Spectral and Affinity Propagation Clustering Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# Applications

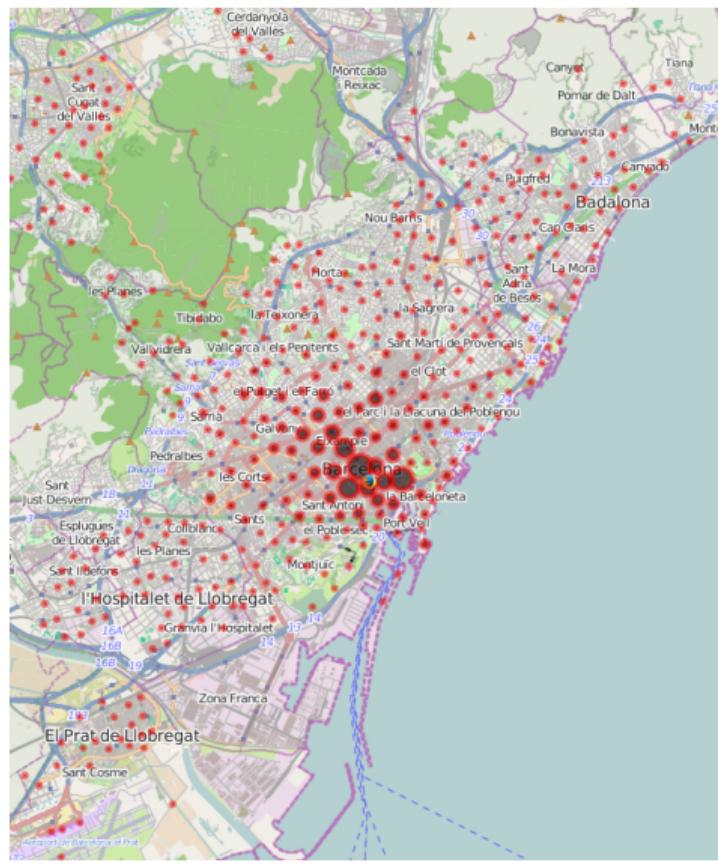
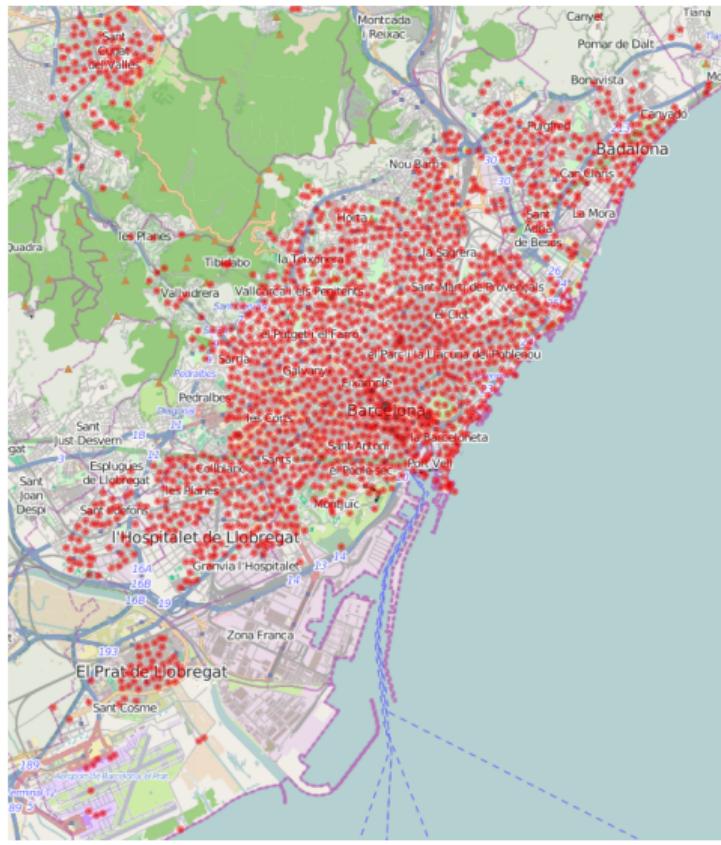
---

- ① **Goal:** To analyze the geographical behavior of people living/visiting a city
- ① **Dataset:** Tweets/posts inside a geographical area
- ① **Attributes:** geographical information / time stamp of the post
- ① **Processes:**
  - Geographical discretization for user representation
  - Discovery of behavior profiles

- ⑤ Focus on the **discovery of different groups of people**
- ⑥ The hypothesis is that users can be segmented according to where they are at different times of the day
- ⑦ We could answer questions like:
  - people from one part of the city stay usually in that part?
  - people from outside the city go to the same places?
  - public transportation is preferred by different profiles? ...

- ④ Raw geolocation and timestamp are **too fine grained**
- ④ Even when we have millions of events the probability of having two events at the same place and at the same time is low
- ④ Discretizing localization and time will increase the probability of finding patterns
- ④ How do we discretize space and time? **Clustering**

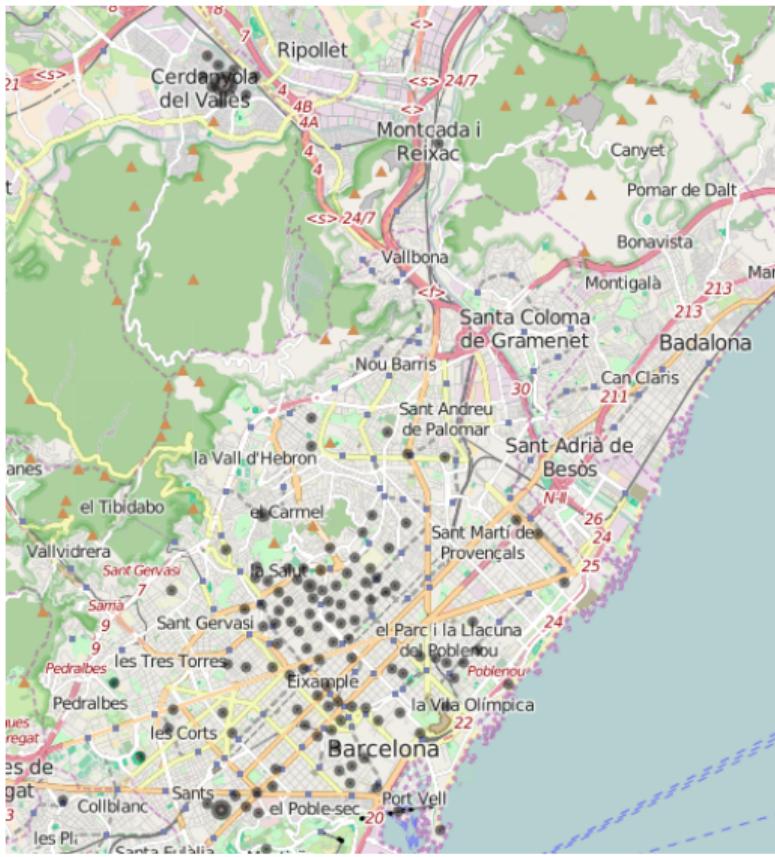
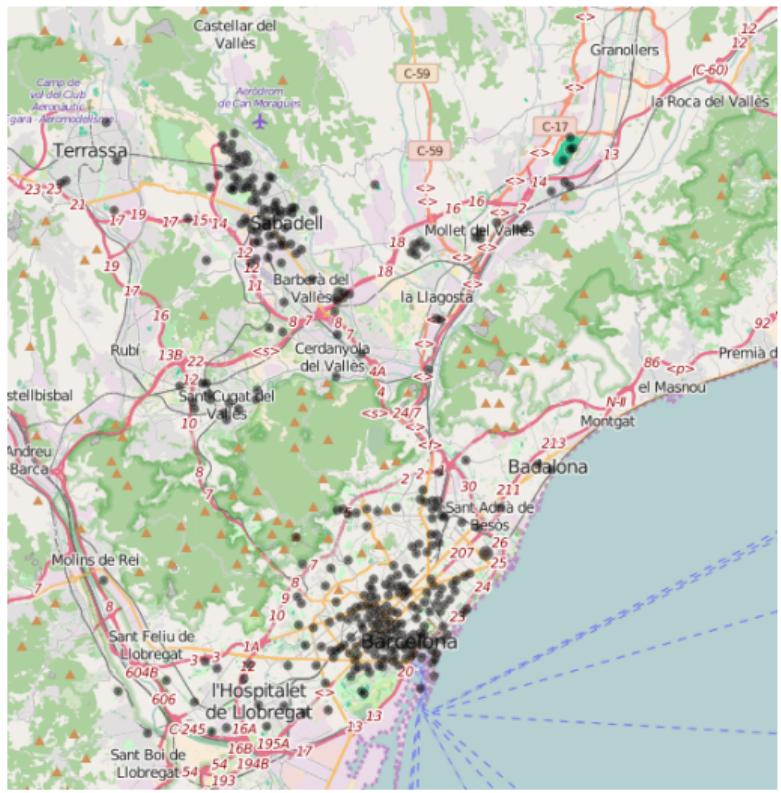
## Clustering of the events - Leader (200m/500m radius)

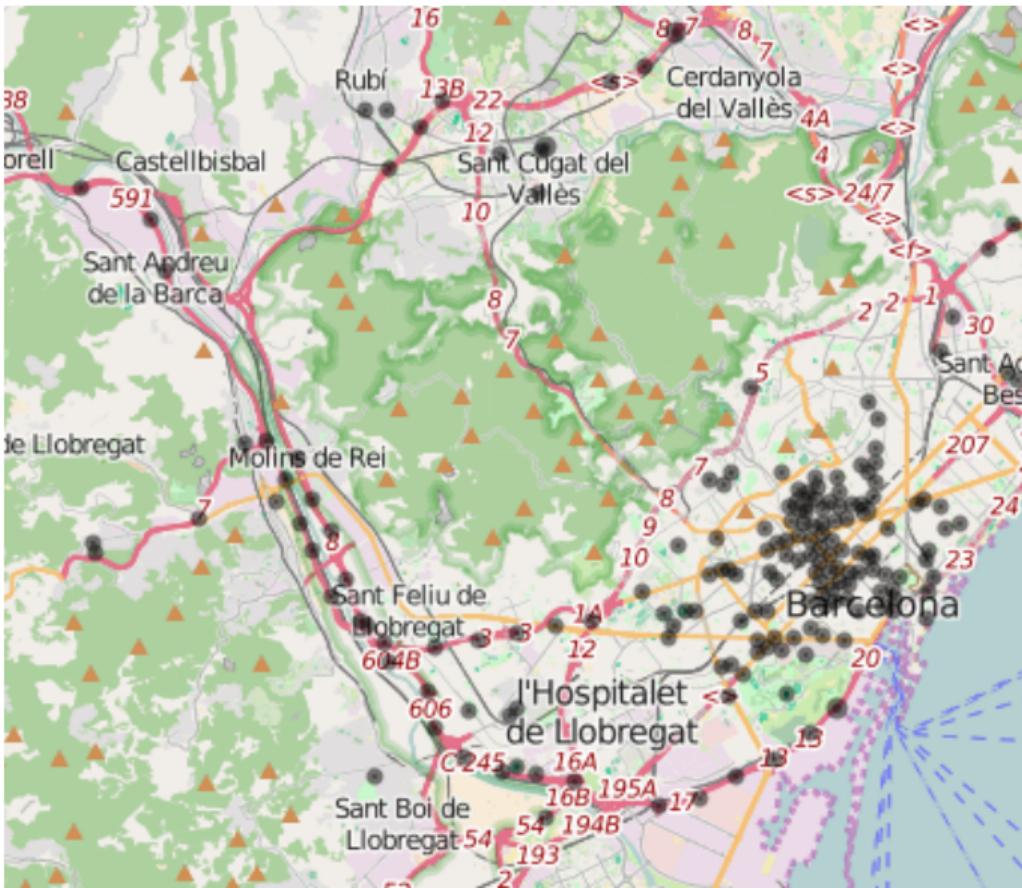


- We could explore the **aggregated behavior** of a user for a long period (month, year)
- Each example represents the places/times of the events of a user for the period
- We obtain a representation similar to the *Bag of Words* used in text mining
  - User  $\Rightarrow$  Document
  - Time  $\times$  Place  $\Rightarrow$  Word

- ④ Difficult to choose the adequate clustering algorithm
- ④ Some choices will depend on:
  - Types of attributes: continuous or discrete values, sparsity of the data
  - Size of the dataset: Aggregating the events reduces largely the size of the data (no scalability issues)
  - Assumptions about the model that represents our goals: Shape of the clusters, Separability of the clusters/Distribution of examples
  - Interpretability/Representability of the clusters

# K-means - Twitter - Moving from near cities to Barcelona





# Clustering Evaluation

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Cluster Evaluation

---

- ① The **evaluation** of unsupervised learning is **difficult**
- ① There is no goal model to compare with
- ① The true result is unknown, it **may depend on the context**, the task to perform...
- ① Why do we want to evaluate them?
  - To avoid finding patterns in noise
  - To compare clustering algorithms
  - To compare different models/parameters

- Cluster tendency, there are clusters in the data?
- Compare the clusters to the true partition of the data
- Quality of the clusters without reference to external information
- Compare the results of different clustering algorithms
- Evaluate algorithm parameters
  - For instance, to determine the *correct* number of clusters

- Before clustering a dataset we can test if there are actually clusters
- We have to test the hypothesis of the existence of patterns in the data versus a dataset uniformly distributed (homogeneous distribution)

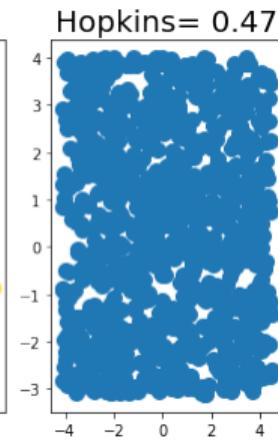
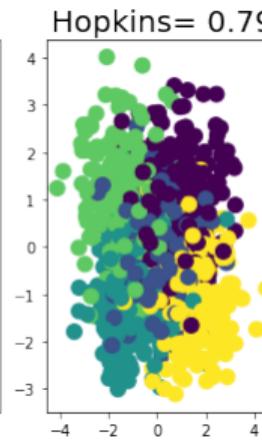
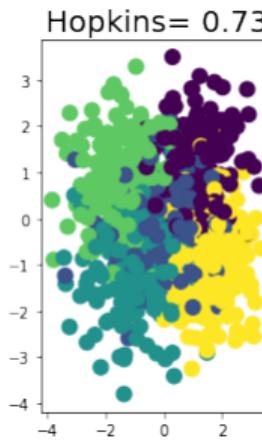
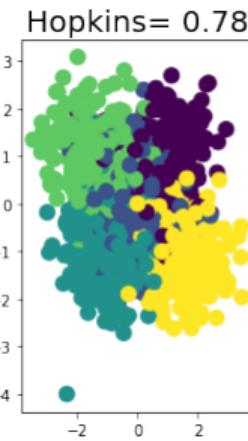
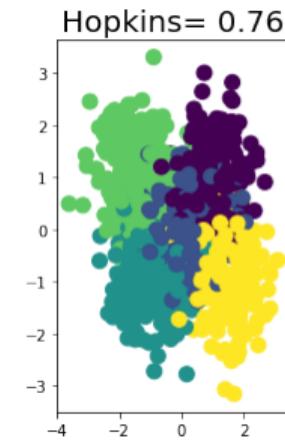
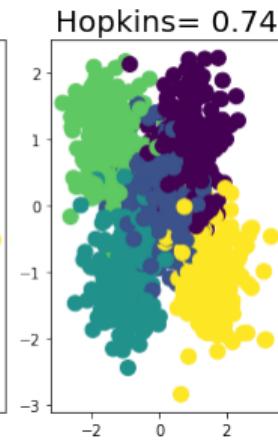
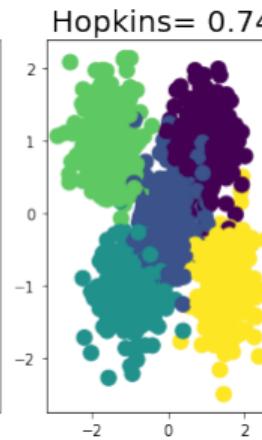
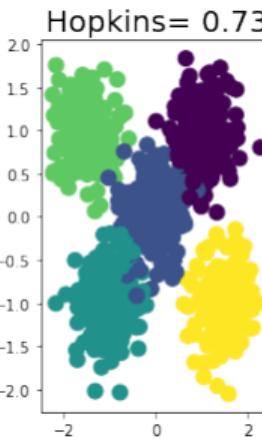
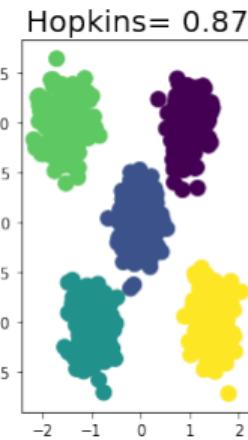
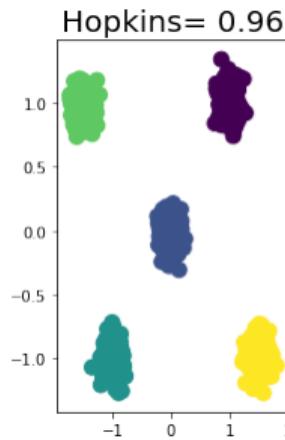
## ⑤ Hopkins Statistic

1. Sample n points ( $p_i$ ) from the dataset (D) uniformly and compute the distance to their nearest neighbor ( $d(p_i)$ )
2. Generate n points ( $q_i$ ) uniformly distributed in the space of the dataset and compute their distance to their nearest neighbors in D ( $d(q_i)$ )
3. Compute the quotient:

$$H = \frac{\sum_{i=1}^n d(p_i)}{\sum_{i=1}^n d(p_i) + \sum_{i=1}^n d(q_i)}$$

4. If data are uniformly distributed the value of  $H$  will be around 0.5

## Hopkins Statistic - Example



- ④ We can use different methodologies/criterion to evaluate the quality of a clustering:
  - **External criteria:** Comparison with a model partition/labeled data
  - **Internal criteria:** Quality measures based on the examples/quality of the partition
  - **Relative criteria:** Comparison with other clusterings

## Internal criteria

- Measure properties expected in a good clustering
  - Compact groups
  - Well separated groups
- The indices are based on the model of the groups
- We can use indices based on the attributes values measuring the properties of a good clustering
- These indices are based on statistical properties of the attributes of the model
  - Values distribution
  - Distances distribution

- ⑤ Some indices correspond directly to the objective function optimized:
  - Quadratic error/Distortion (k-means)

$$SSE = \sum_{k=1}^k \sum_{\forall x_i \in C_k} \| x_i - \mu_k \|^2$$

- Log likelihood (Mixture of gaussians/EM)

- ⑤ For prototype based algorithms several measures can be used to compute quality indices
- ⑥ Scatter matrices: interclass distance, intraclass distance, separation

$$S_{W_k} = \sum_{\forall x_i \in C_k} (x_i - \mu_k)(x_i - \mu_k)^T$$

$$S_{B_k} = |C_k|(\mu_k - \mu)(\mu_k - \mu)^T$$

$$S_{M_{k,l}} = \sum_{\forall i \in C_k} \sum_{\forall j \in C_l} (x_i - x_j)(x_i - x_j)^T$$

- ④ Trace criteria (lower overall intracluster distance/higher overall intercluster distance)

$$Tr(S_W) = \frac{1}{K} \sum_{i=1}^K S_{W_k} \quad Tr(S_B) = \frac{1}{K} \sum_{i=1}^K S_{B_k}$$

- ④ Calinski-Harabasz index (interclass-intraclass distance ratio)

$$CH = \frac{\sum_{i=0}^K |C_i| \times \|\mu_i - \mu\|^2 / (K - 1)}{\sum_{k=1}^K \sum_{i=0}^{|C_i|} \|x_i - \mu_i\|^2 / (N - K)}$$

## ④ Davies-Bouldin criteria (maximum interclass-intraclass distance ratio)

$$\bar{R} = \frac{1}{K} \sum_{i=1}^K R_i$$

where

$$R_{ij} = \frac{S_{W_i} + S_{W_j}}{S_{M_{ij}}}$$
$$R_i = \max_{j:j \neq i} R_{ij}$$

④ Silhouette index (maximum class spread/variance)

$$S = \frac{1}{N} \sum_{i=0}^N \frac{b_i - a_i}{\max(a_i, b_i)}$$

Where

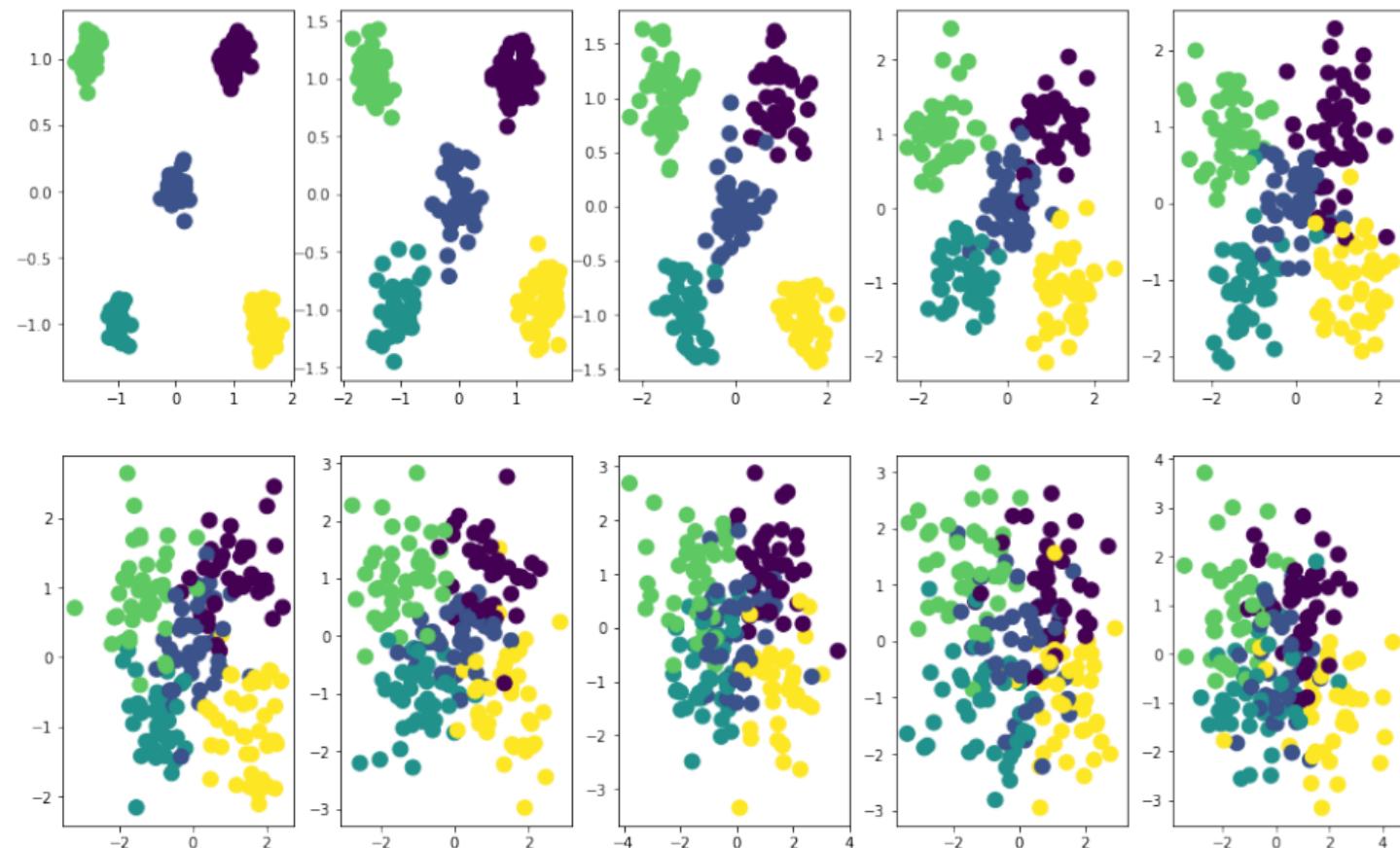
$$a_i = \frac{1}{|C_j| - 1} \sum_{y \in C_j, y \neq x_i} \|y - x_i\|$$

$$b_i = \min_{l \in H, l \neq j} \frac{1}{|C_l|} \sum_{y \in C_l} \|y - x_i\|$$

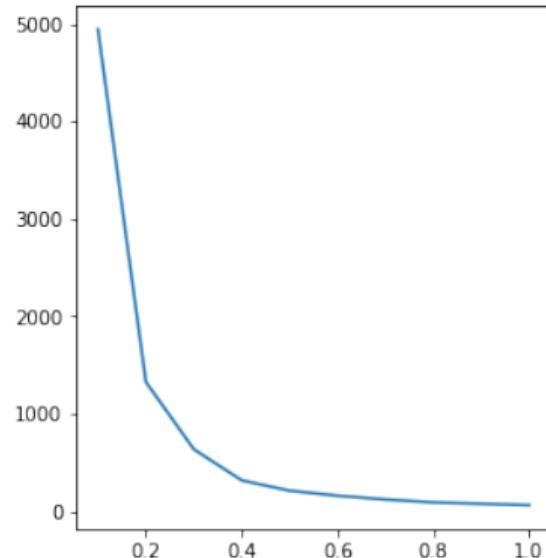
with  $x_i \in C_j, H = \{h : 1 \leq h \leq K\}$

- More than 30 indices can be found in the literature
- Several studies and comparisons have been performed
- Recent studies (Arbelatiz et al, 2013) have exhaustively tested these indices, some have a performance significantly better than others
- Some indices show a similar performance (not statistically different)
- The study concludes that Silhouette, Davies-Bouldin and Calinski Harabasz perform well in a wide range of situations

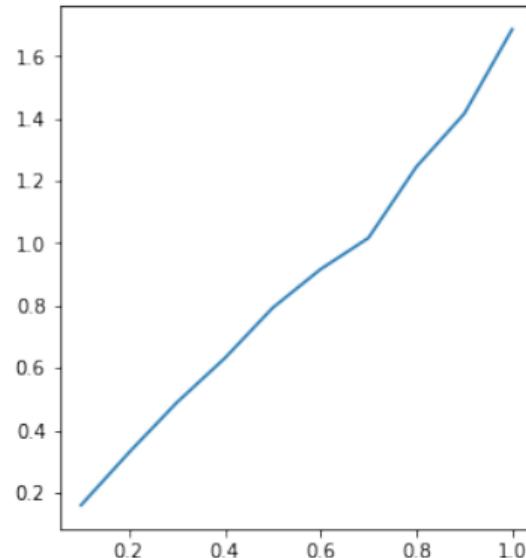
Internal criteria - 5 clusters different variance



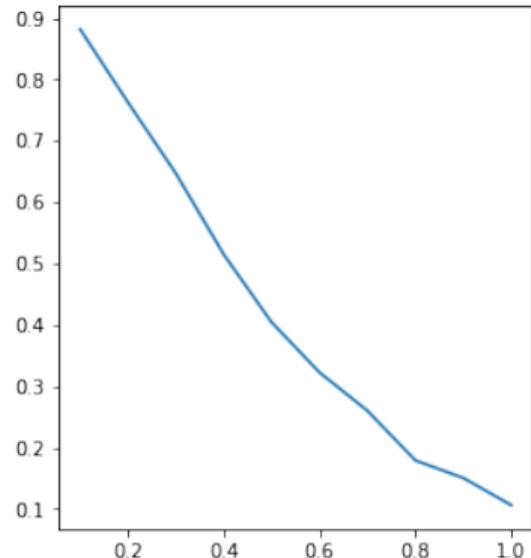
Calinski-Harabaz



Davies-Bouldin



Silhouette



## External criteria

- These indices measure the similarity of a clustering to a model partition  $P$
- Without a model they can be used to compare the results of using different parameters or different algorithms
  - For instance, can be used to assess the sensitivity to initialization
- The main advantage is that these indices are independent of the examples/cluster description
- That means that they can be used to assess any clustering algorithm

- All the indices are based on the coincidence of each pair of examples in the groups of two clusterings
- The computations are based on four values:

		Clustering 1	Same	Different
		Clustering 2		
Clustering 2	Same	a	b	
	Different	c	d	

## ⑤ Rand/Adjusted Rand statistic:

$$Rand = \frac{(a + d)}{(a + b + c + d)}; \quad ARand = \frac{a - \frac{(a+c)(a+b)}{a+b+c+d}}{\frac{(a+c)+(a+b)}{2} - \frac{(a+b)(a+c)}{a+b+c+d}}$$

## ⑥ Jaccard Coefficient:

$$J = \frac{a}{(a + b + c)}$$

## ⑦ Folkes and Mallow index:

$$FM = \sqrt{\frac{a}{a + b} \cdot \frac{a}{a + c}}$$

- ⑤ Defining Mutual Information between two partitions as:

$$MI(Y_i, Y_k) = \sum_{X_c^i \in Y_i} \sum_{X_{c'}^k \in Y_k} \frac{|X_c^i \cap X_{c'}^k|}{N} \log_2 \left( \frac{N |X_c^i \cap X_{c'}^k|}{|X_c^i| |X_{c'}^k|} \right)$$

- ⑥ and Entropy of a partition as

$$H(Y_i) = - \sum_{X_c^i \in Y_i} \frac{|X_c^i|}{N} \log_2 \left( \frac{|X_c^i|}{N} \right)$$

where  $X_c^i \cap X_{c'}^k$  is the number of objects that are in the intersection of the two groups

## ⑤ Normalized Mutual Information:

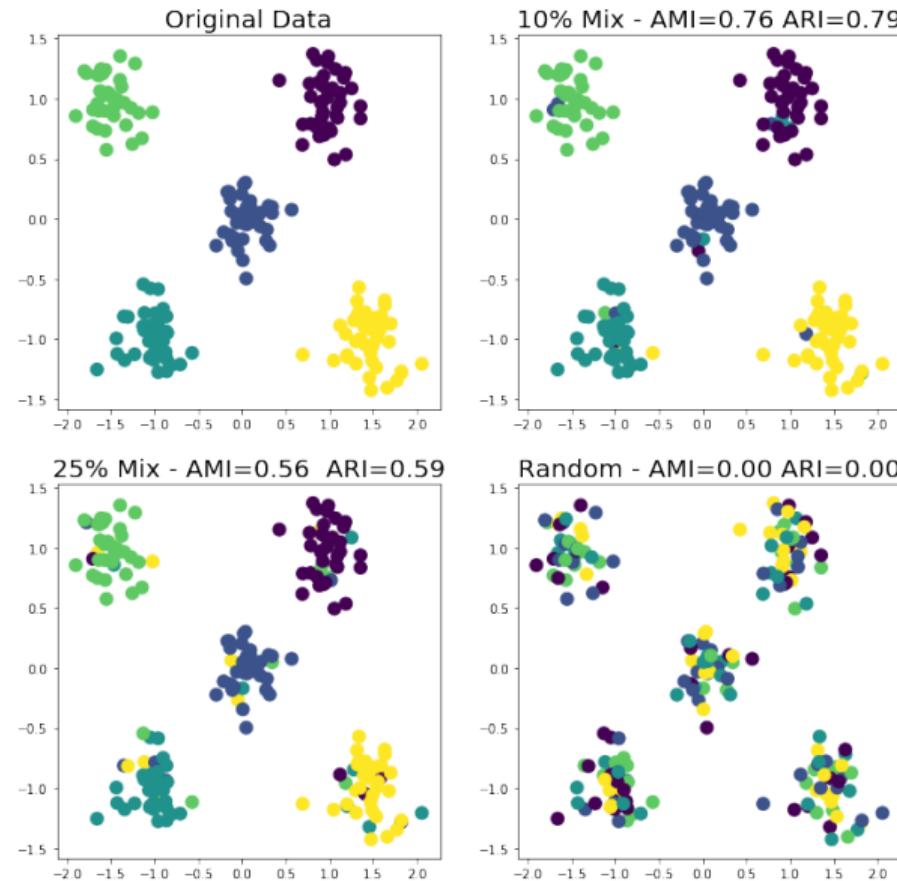
$$NMI(Y_i, Y_k) = \frac{MI(Y_i, Y_k)}{\sqrt{H(Y_i)H(Y_k)}}$$

## ⑥ Variation of Information:

$$VI(C, C') = H(C) + H(C') - 2I(C, C')$$

## ⑦ Adjusted Mutual Information:

$$AMI(U, V) = \frac{MI(U, V) - E(MI(U, V))}{\max(H(U), H(V)) - E(MI(U, V))}$$



Number of clusters

---

- ⑤ A topic related to cluster validation is to decide if the number of clusters obtained is the correct one
- ⑤ This point is important specially for the algorithms that need this value as a parameter
- ⑤ The usual procedure is to compare the characteristics of clusterings of different sizes
- ⑤ Usually internal criteria indices are used in this comparison
- ⑤ A graphic of these indices for different number of clusters can show what number of clusters is more probable

- Some internal validity indices can be used for this purpose: Calinsky Harabasz index, Silhouette index
- Using the within class scatter matrix ( $S_W$ ) other criteria can be defined:
  - Hartigan index:

$$H(k) = \left[ \frac{S_W(k)}{S_W(k+1)} - 1 \right] (n - k - 1)$$

- Krzanowski Lai index:

$$KL(k) = \left| \frac{DIFF(k)}{DIFF(k+1)} \right|$$

being  $DIFF(k) = (k-1)^{2/p} S_W(k-1) - k^{2/p} S_W(k)$

- Assess the number of clusters comparing a clustering with the expected distribution of data given the null hypothesis (no clusters)
- Computes different clusterings of the data increasing the number of clusters and compare them to clusters of data (B) generated with a **uniform distribution**
- The interclass distance matrix  $S_W$  is computed for both and compared.
- The correct number of clusters is where the **widest gap** appears between the  $S_W$  of the data and the uniform data

- ④ The Gap statistic:

$$Gap(k) = (1/B) \sum_b \log(S_W(k)_b) - \log(S_W(k))$$

The first term is the mean of  $S_W$  for the clusters obtained from the uniform distributed data

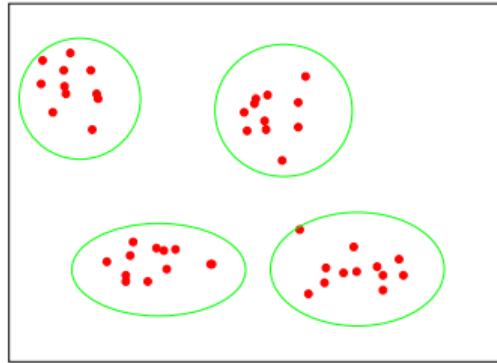
- ④ From the st. dev. ( $sd_k$ ) of  $\sum_b \log(S_W(k)_b)$  is defined  $s_k$  as:

$$s_k = sd_k \sqrt{1 + 1/B}$$

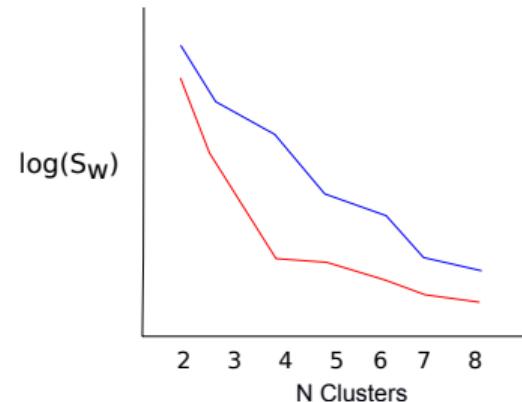
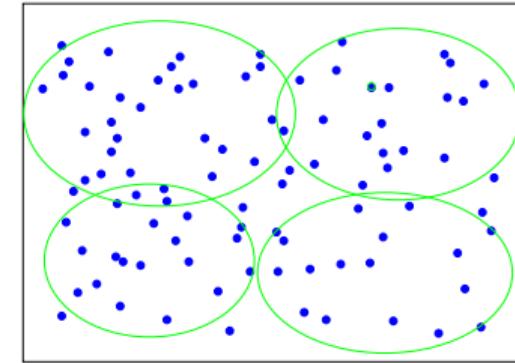
- ④ The probable number of clusters is the smallest number that holds:

$$Gap(k) \geq Gap(k+1) - s_{k+1}$$

Clustered Data



Uniform Distributed Data



- ⑤ The idea is that if the model chosen for clustering a dataset is correct, it should be stable for different samplings of the data
- ⑥ The procedure is to obtain different subsamples of the data, cluster them and test their stability

◎ Using disjoint samples:

- Dataset divided in two disjoint samples that are clustered separately
- Indices can be defined to assess stability, for example using the distribution of the number of neighbors that belong to the complementary sample

◎ Using non disjoint samples:

- Dataset divided in three disjoint samples ( $S_1, S_2, S_3$ )
- Two clusterings are obtained from  $S_1 \cup S_3, S_2 \cup S_3$
- Indices can be defined about the coincidence of the common examples in both partitions

This Python Notebook has examples for Measures of Clustering Validation

- ⑤ Clustering Validation Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

- ⑤ In the code from the repository inside subdirectory Validation you have the python program ValidationAuthors,
- ⑥ The authors dataset is clustered with different algorithms (K-means, GMM, Spectral) and different validity indices are plotted for the number of clusters

# Clustering in KDD

---

Javier Béjar

URL - 2021 Spring Semester

CS - MIA



# Introduction

---

- ⑤ One of the main tasks in the KDD process is the analysis of data when we do not know its structure
- ⑤ This task is very different from the task of prediction where we know the goal, and we try to approximate it
- ⑤ A great part of the KDD tasks are non supervised problems (KD Nuggets poll, 2-3 most frequent task)
- ⑤ Problems: Scalability, arbitrary cluster shapes, limited types of data, finding the correct parameters...
- ⑤ There are some new algorithms that deal with these kinds of problems

# Scalability Strategies

---

- ① One-pass
  - Process data as a stream
- ① Summarization/Data compression
  - Compress examples to fit more data in memory
- ① Sampling/Batch algorithms
  - Process a subset of the data and maintain/compute a global model
- ① Approximation
  - Avoid expensive computations by approximate estimation
- ① Paralelization/Distribution
  - Divide the task in several parts and merge models

- ④ This strategy is based on incremental clustering algorithms
- ④ They are cheap but order of processing affects greatly their quality
- ④ Although they can be used as a preprocessing step
- ④ Two steps algorithms
  1. Many clusters are generated using the one-pass algorithm
  2. A more accurate algorithm clusters the preprocessed data

- Not all the data is necessary to discover the clusters
- Discard sets of examples and summarize by:
  - Sufficient statistics
  - Density approximations
- Discard data irrelevant for the model (do not affect the result)

- ⑤ Not using all the information available to make decisions
  - Using K-neighbours (data structures for computing k-neighbours)
- ⑥ Preprocessing the data using a cheaper algorithm
  - Generate batches using approximate distances (eg: canopy clustering)
- ⑦ Use approximate data structures
  - Use of hashing or approximate counts for distances and frequency computation

- ① Process only data that fits in memory
- ② Obtain from the data set:
  - Samples (process only a subset of the dataset)
    - Determine the size of the sample so all the clusters are represented
  - Batches (process all the dataset)

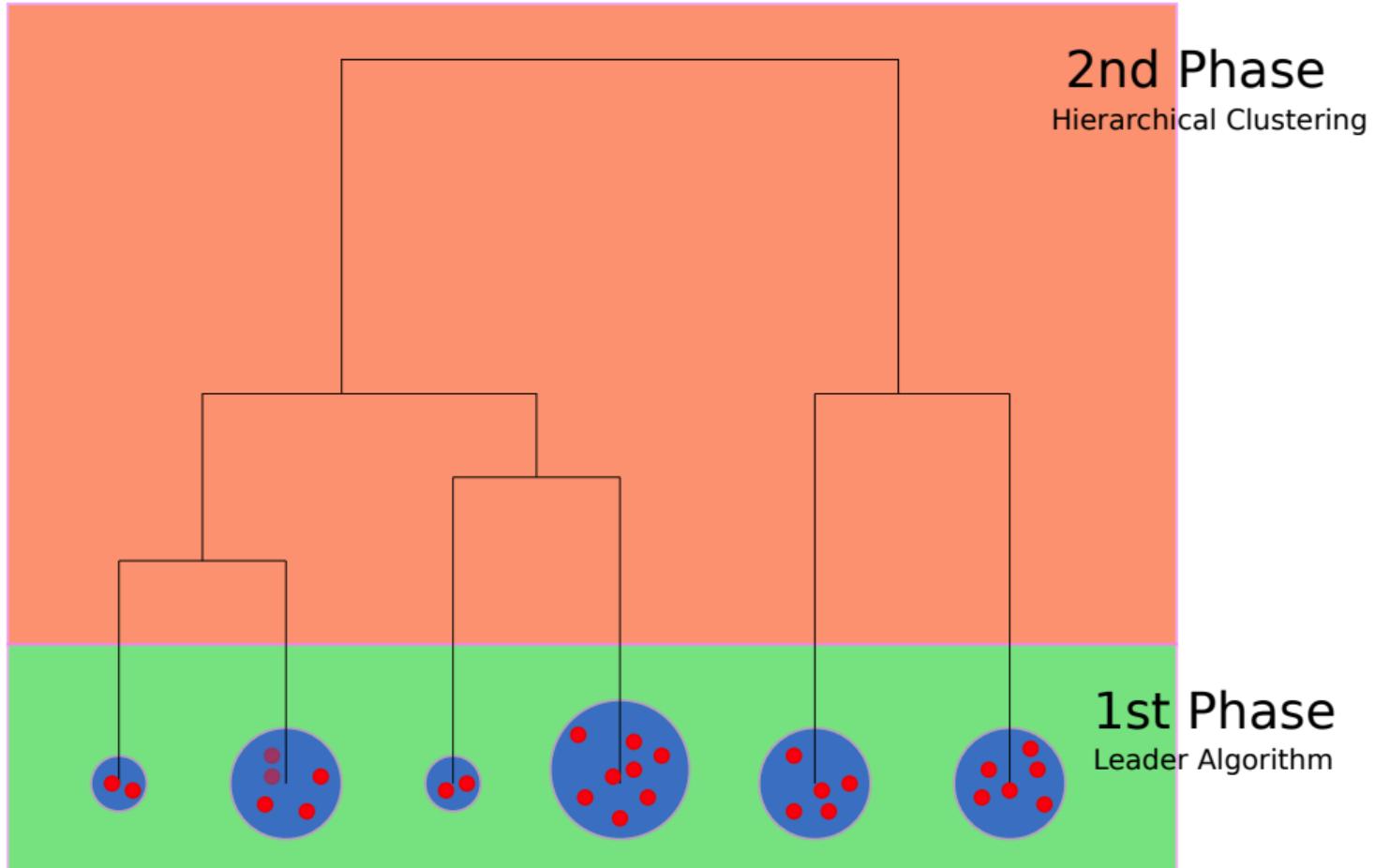
- ⑤ Paralelization usually depends on the specific algorithm
- ⑤ Some not easy to parallelize (eg: hierarchical clustering)
- ⑤ Some have specific parts that can be solved in parallel or by Divide&Conquer
  - Distance computations in k-means
  - Parameter estimation in EM algorithms
  - Grid density estimations
  - Space partitioning
- ⑤ Batches and sampling are more general approaches
  - The problem is how to merge all the different partitions

# Scalable Algorithms

---

Patra, Nandi, Viswanath **Distance based clustering method for arbitrary shaped clusters in large datasets** Pattern Recognition, 2011, 44, 2862-2870

- ④ **Strategy:** One pass + Summarization
- ④ The leader algorithm is used as a one pass summarization using Leader algorithm (many clusters)
- ④ Single link is used to cluster the summaries
- ④ Guarantees the equivalence to SL at top levels
- ④ Summarization makes the algorithm independent of the dataset size (depends on the radius used on the leader algorithm and the volume of the data)
- ④ Complexity  $O(c^2)$



**2nd Phase**  
Hierarchical Clustering

**1st Phase**  
Leader Algorithm

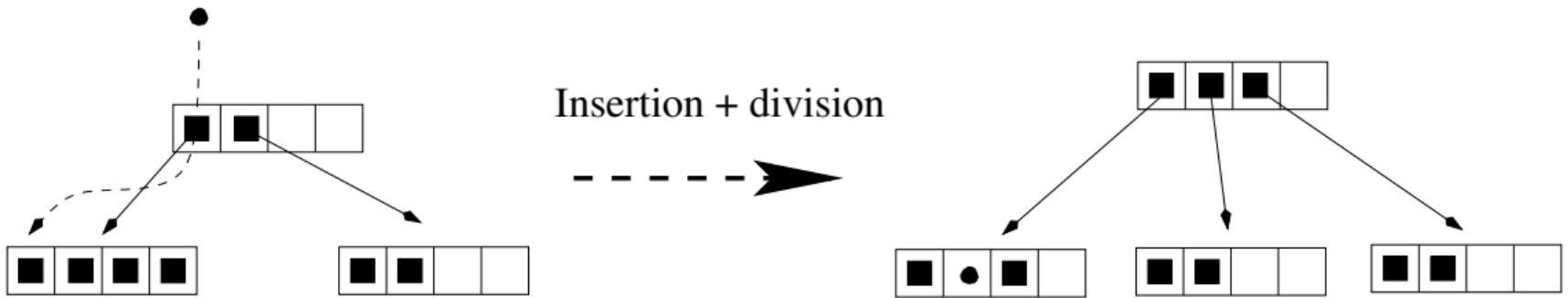
Zhang, Ramakrishnan, Livny **BIRCH: An Efficient Data Clustering Method for Very Large Databases (1996)**

- ① **Strategy:** One-pass + Summarization
- ② Hierarchical clustering with limited memory
- ③ Incremental algorithm
- ④ Based on probabilistic prototypes and distances
- ⑤ We need two passes from the database
- ⑥ Based on a specialized data structure named CF-tree (Clustering Feature Tree)

- ④ Balanced n-ary tree containing clusters represented by probabilistic prototypes
- ④ Leaves have capacity  $L$  prototypes and clusters radius can not be more than  $T$
- ④ Non-terminal nodes have a fixed branching factor ( $B$ ), each element summarizes its subtree
- ④ Choice of parameters is crucial because available space could be filled during the process
- ④ This is solved by changing the parameters (basically  $T$ ), and recompressing the tree ( $T$  determines the granularity of the final groups)

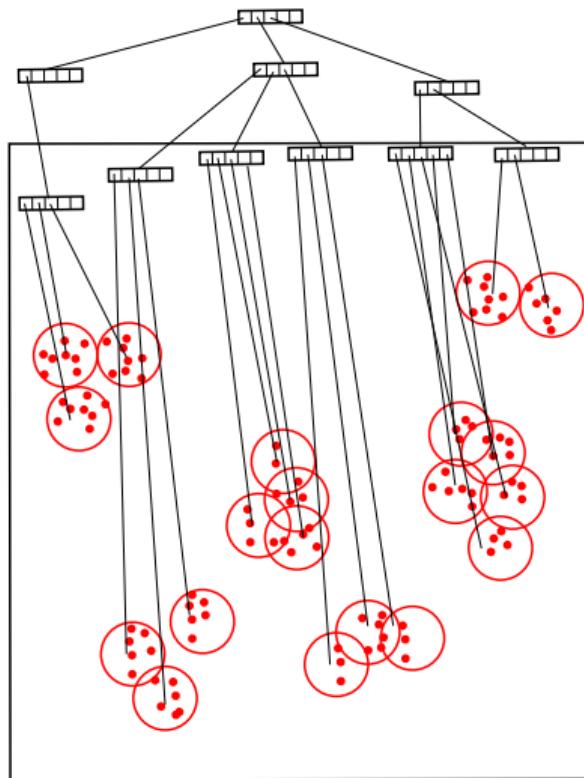
1. Traverse the tree until reaching a leave and choose the nearest prototype
2. On this leave we could introduce the instance in an existing group or create a new prototype depending on if the distance is larger than parameter  $T$
3. If the current leave has **no space** for the new prototype, then **create a new terminal node**, and **distribute the prototypes** among the current node and the new node

4. The distribution is performed choosing the two most different prototypes and dividing the rest using their proximity to these two prototypes
5. This creates a new node in the ascendant node, if the new node exceeds the capacity of the father then it is split, and the process is continued until the root of the tree is reached if necessary
6. Additionally, we could perform merge operations to compact the tree and reduce space

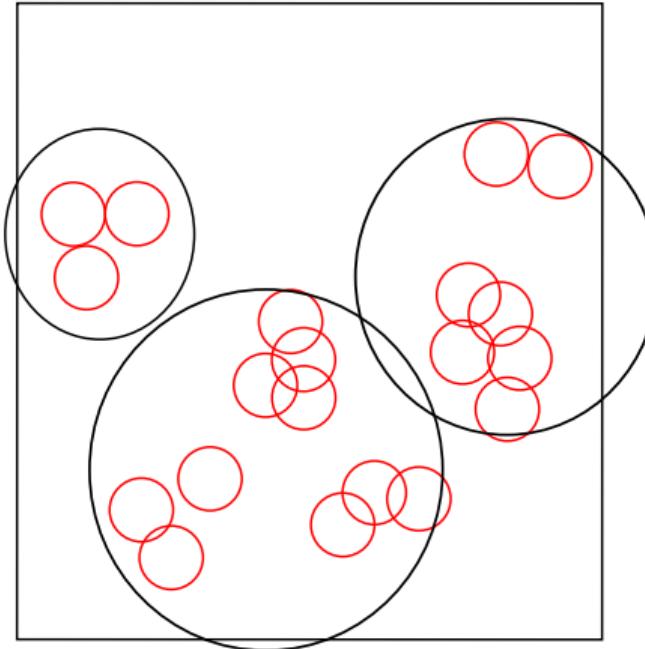


1. **Phase 1:** Construction of the CF-tree, we obtain a hierarchy that summarizes the database as a set of groups which granularity is defined by  $T$
2. **Phase 2:** Optionally we modify the CF-tree in order to reduce its size by merging near groups and deleting outliers
3. **Phase 3:** We use the prototypes inside the leaves of the trees as new instances, and we run a clustering algorithm with them (for instance K-means)
4. **Phase 4:** We refine the groups assigning the instances from the original database to the prototypes obtained in the previous phase

1st Phase - CFTree



2nd Phase - Kmeans



Bradley, Fayyad, Reina **Scaling Clustering Algorithms to Large Databases** Knowledge Discovery and Data Mining (1998)

- **Strategy:** Sampling + Summarization
- Clustering algorithms need to have all data in main memory to perform their task
- We try to obtain scalability looking for an algorithm that:
  - Only look at the data one time
  - To be anytime (a result is available always)
  - To be incremental (more data not to start from scratch)
  - To be suspendable (continue from current solution)
  - To use limited memory

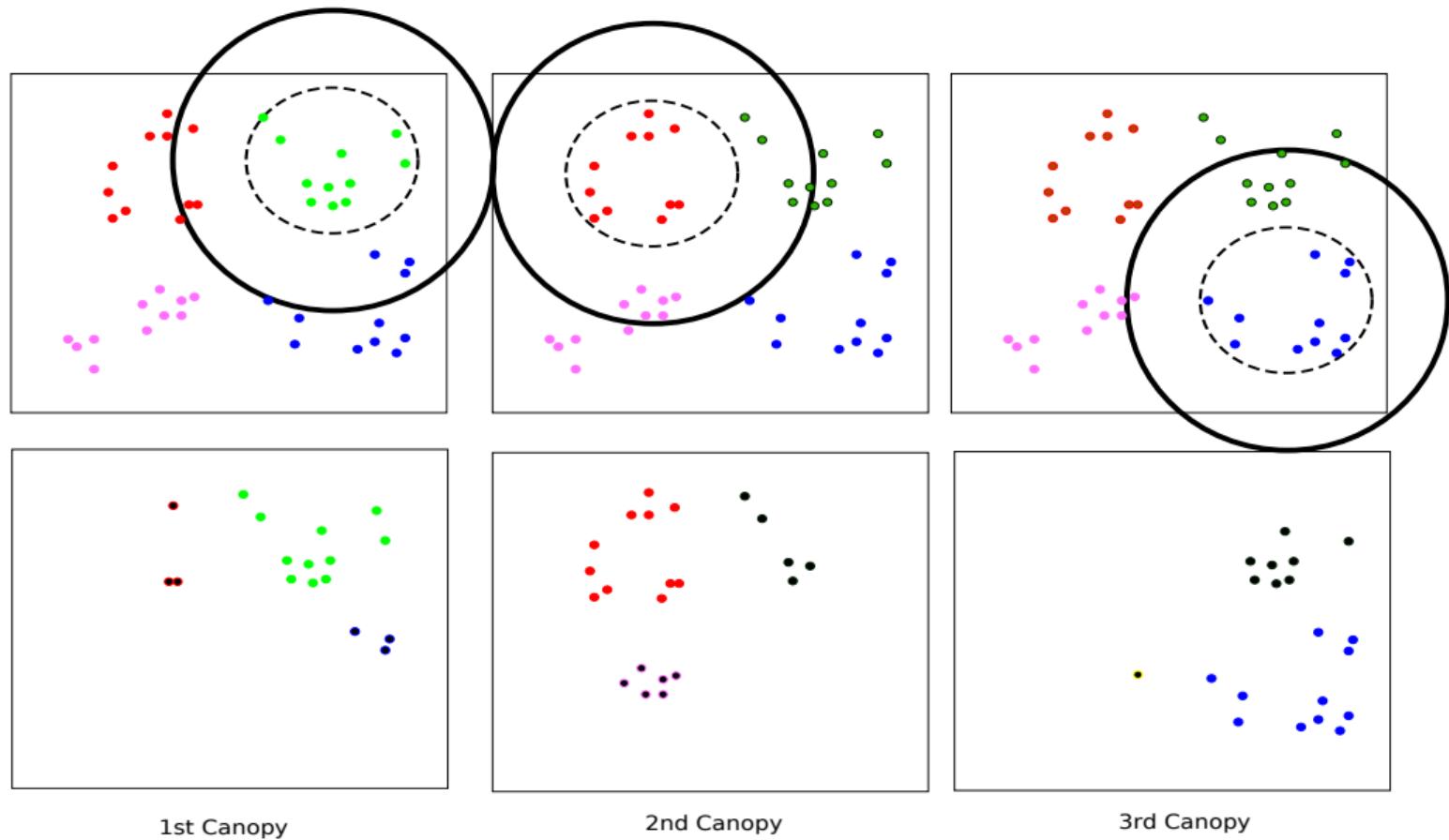
- ⑤ Obtain a sample that fits in memory
- ⑥ Update the actual model
- ⑦ Classify new instances as:
  - Necessary
  - Discardable (We keep their information as sufficient statistics)
  - Summarizable using data compression
- ⑧ Decide if the model is stable, or we keep clustering more data

McCallum, Nigam, Ungar **Efficient clustering of high-dimensional data sets with application to reference matching** (2002)

- ① **Strategy:** Divide & Conquer + Approximation
- ① The approach is based on a two stages clustering
- ① The first stage can be seen as a preprocess to determine the neighborhood of the densities and reducing the number of distances to compute on the second stage
- ① This first stage is the called *canopy clustering*, relies on a cheap distance and two parameters  $T_1 > T_2$
- ① These parameters are used as two centered spheres that determine how to classify the examples.

- Algorithm:
  1. One example is picked at random, and the cheap distance from this example to the rest is computed
  2. All the examples that are at less than  $T_2$  are deleted and included in the canopy
  3. The points at less than  $T_1$  are added to the canopy of these examples without deleting them
  4. The procedure is repeated until the example list is empty
  5. Canopies can share examples
- After that the data can be clustered with different algorithms
- For agglomerative clustering only the distances among the examples in the canopies have to be computed

## Canopy Clustering - Algorithm



Sculley Web-scale k-means clustering Proceedings of the 19th international conference on World wide web, 2010, 1177-1178

- ① **Strategy:** Sampling
- ① Apply K-means to a sequence of bootstrap samples of the data
- ① Each iteration the samples are assigned to prototypes, and the prototypes are updated with the new sample
- ① Each iteration the weight of the samples is reduced (learning rate)
- ① The quality of the results depends on the size of the batches
- ① Convergence is detected when prototypes are stable

---

**Given:** k, mini-batch size b, iterations t, data set X

Initialize each  $c \in C$  with an  $x$  picked randomly from X

$v \leftarrow 0$

**for**  $i \leftarrow 1$  to  $t$  **do**

$M \leftarrow b$  examples picked randomly from X

**for**  $x \in M$  **do**

$d[x] \leftarrow f(C, x)$

**for**  $x \in M$  **do**

$c \leftarrow d[x]$

$v[c] \leftarrow v[c] + 1$

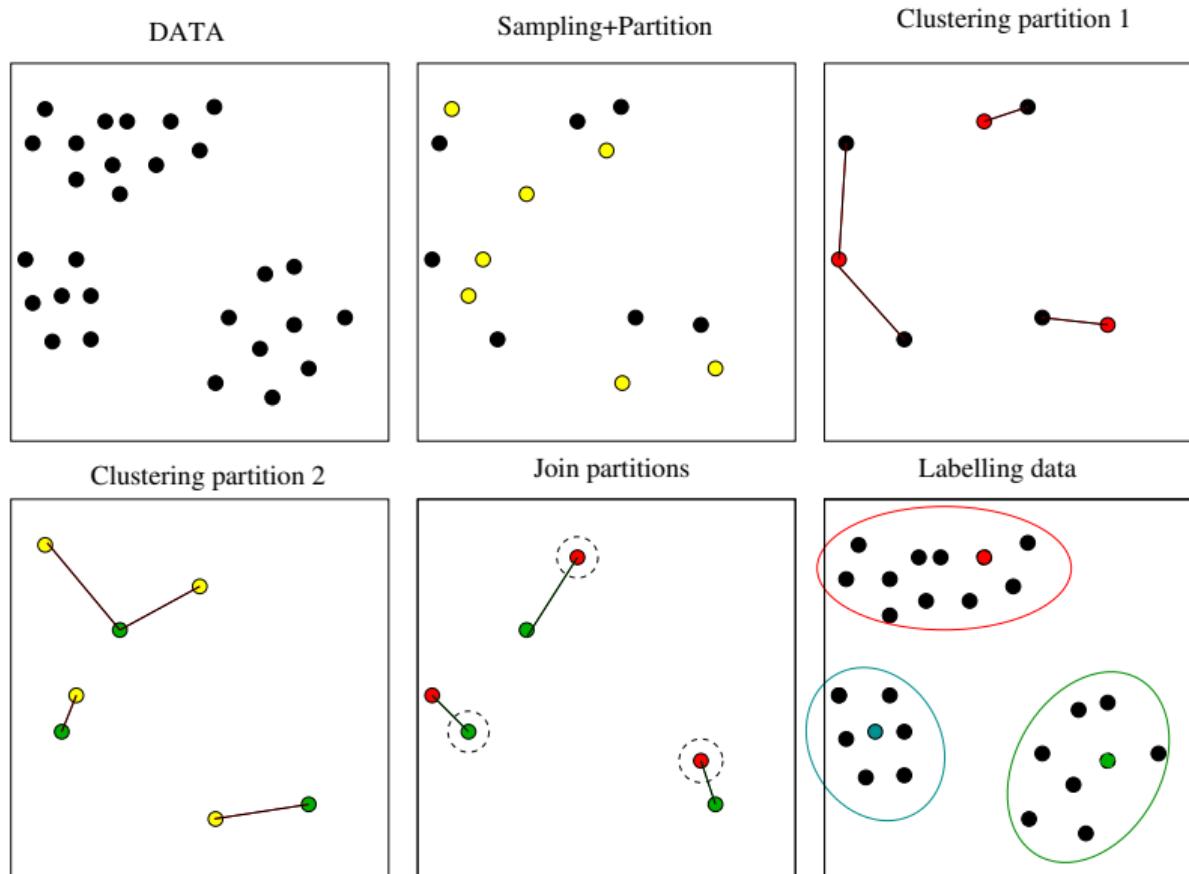
$\eta \leftarrow \frac{1}{v[c]}$

$c \leftarrow (1-\eta)c + \eta x$

Guha, Rastogi, Shim **CURE: An efficient clustering algorithm for large databases** (1998)

- ① **Strategy:** Sampling + Divide & Conquer
- ① Hierarchical agglomerative clustering
- ① Scalability is obtained by using sampling techniques and partitioning the dataset
- ① Uses a set of representatives ( $c$ ) for a cluster instead of centroids (non-spherical groups)
- ① Distance is computed as the nearest pair of representatives among groups
- ① The clustering algorithm is agglomerative and merges pairs of groups until  $k$  groups are obtained

1. Draws a random sample from the dataset
2. Partitions the sample in  $p$  groups
3. Executes the clustering algorithm on each partition
4. Deletes outliers
5. Runs the clustering algorithm on the union of all groups until it obtains  $k$  groups
6. Label the data accordingly to the similarity to the  $k$  groups

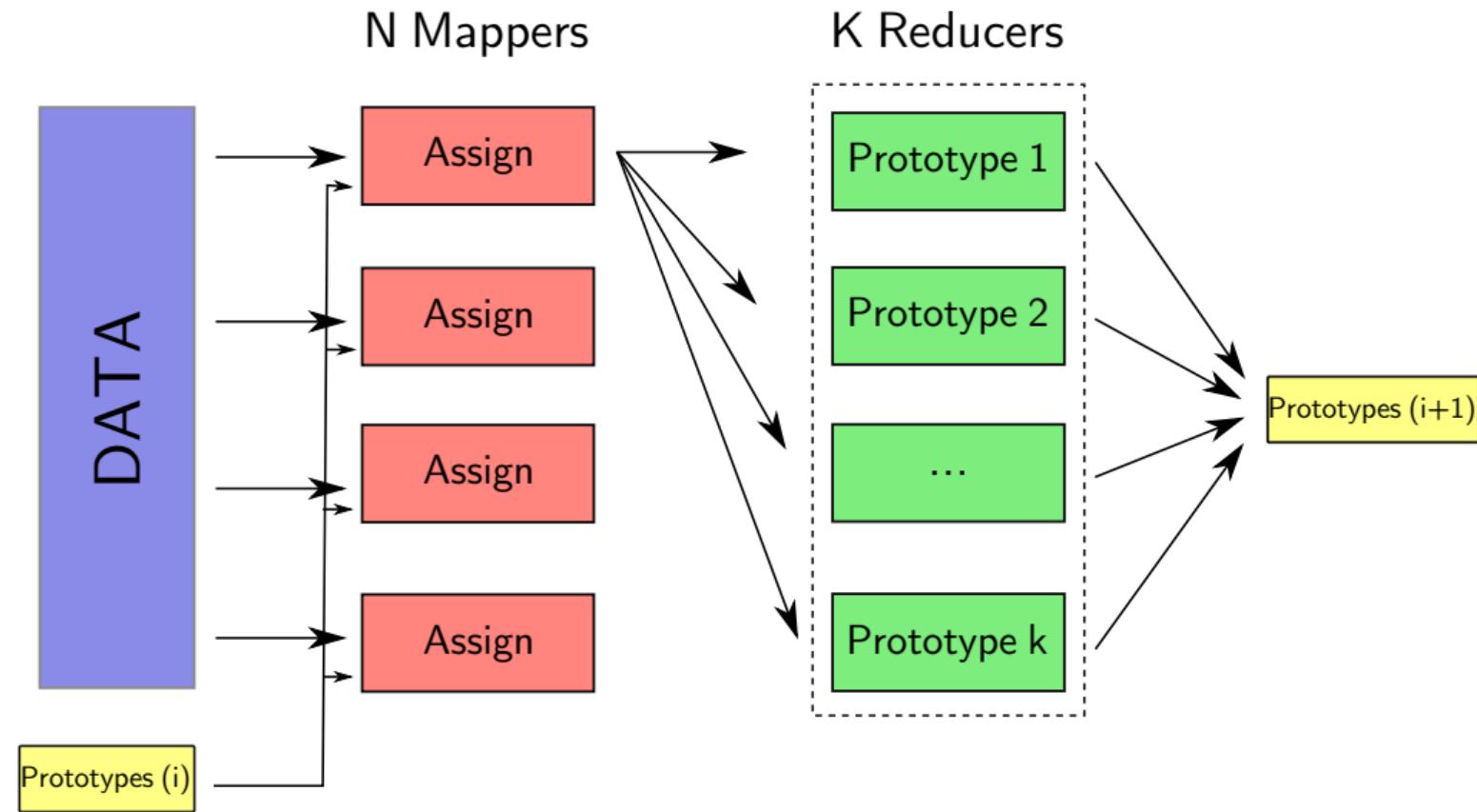


Viswanath, Babu **Rough-DBSCAN: A fast hybrid density based clustering method for large data sets** Pattern Recognition Letters, 2009, 30, 1477 - 1488

- ① **Strategy:** One-pass + Summarization
- ② Two stages algorithm:
  1. Preprocess using the leader algorithm
    - Determine the instances that belong to the higher densities and their neighbours
  2. Apply DBSCAN algorithm
    - Determine the densities for the selected instances
    - Approximate the values of the densities from their distances, and the sizes of the neighbor
    - Assign the neighbors accordingly to the found densities

Zhao, W., Ma, H., He, Q. **Parallel K-Means Clustering Based on MapReduce Cloud Computing**, LNCS 5931, 674-679, Springer 2009

- **Strategy:** Distribution/Divide & Conquer
- Applied to K-means and GMM
- Mappers have a copy of the current centroids and assigns the closest one to examples
- Reducers compute the new centroids according to the assignments



- Use Peer2Peer networks as a divide and conquer strategy
- Each member of the network process a chunk of the data
- Peers interchange messages with data or intermediate results
- Strategies:
  - **Synchronous**: All peers interchange information at timed intervals
  - **Asynchronous**: All peers work independently and interchange information randomly
- Messages:
  - Messages consist of prototypes that are integrated (possibly with a weighting strategy)
  - Messages consist of examples or summarized examples

This Python Notebook has examples comparing K-means algorithm with two scalable algorithms Mini Batch K-means and BIRCH

- ⑤ Clustering DM Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# Consensus Clustering

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



- The ensemble of classifiers is a well established strategy in supervised learning
- Unsupervised learning aims the same goal: Consensus clustering or clustering ensemble
- The idea is to merge **complementary perspectives** of the data into a more **stable partition**

- ⑤ Given a set of partitions of the same data  $\mathcal{X}$ :

$$\mathbb{P} = \{P^1, P^2, \dots, P^n\}$$

with:

$$P^1 = \{C_1^1, C_2^1, \dots, C_{k_1}^1\}$$

⋮

$$P^n = \{C_1^n, C_2^n, \dots, C_{k_n}^n\}$$

to obtain a new partition that uses the information of all  $n$  partitions

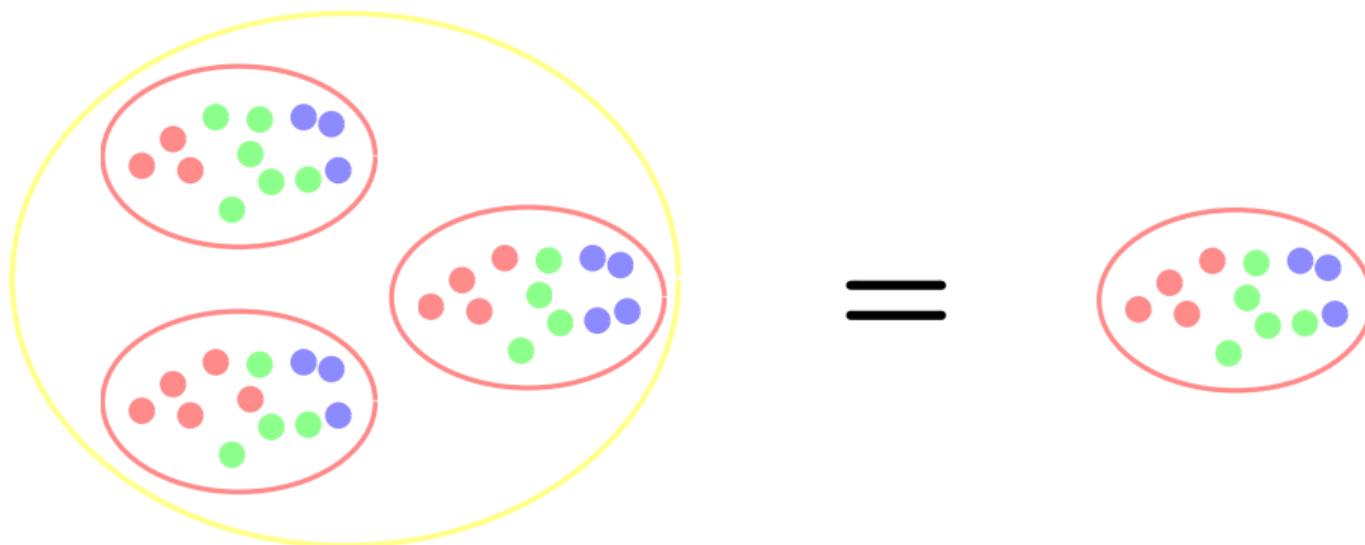
- ④ **Robustness**, the combination has a **better performance** than each individual partition in some sense
- ④ **Consistency**, the combination is **similar** to the individual partitions
- ④ **Stability**, the resulting partition is **less sensitive to outliers and noise**
- ④ **Novelty**, the combination is able to obtain **different partitions** that can not be obtained by the clustering methods that generated the individual partitions

- ④ **Knowledge reuse**, the consensus can be computed from the partition assignments so previous partitions using the same or different attributes can be used
- ④ **Distributed computing**, the individual partitions can be obtained independently
- ④ **Privacy**, only the assignments of the individual partitions are needed for the consensus

# Consensus Process

---

- ⑤ Consensus clustering is based generally in a two steps process:
  1. Generate the individual partitions to be combined
  2. Combine the partitions to generate the final partition



- ⑤ **Different example representations:** Diversity by generating partitions with different subsets of attributes
- ⑤ **Different clustering algorithms:** Take advantage that all clustering algorithms have a different biases
- ⑤ **Different parameter initialization:** Use clustering algorithms able to produce different partitions using different parameters
- ⑤ **Subspace projection:** Use dimensionality reductions techniques
- ⑤ **Subsets of examples:** Use random subsamples of the dataset (bootstrapping)

- ④ **Cooccurrence based methods:** Use the labels obtained from each individual clustering, and the coincidence of the labels for the examples
  - Relabeling and voting, co-association matrix, graph and hypergraph partitioning, information theory measures, finite mixture models
- ④ **Median partition based methods:** Given a set of partitions ( $\mathcal{P}$ ) and a similarity function ( $\Gamma(P_i, P_j)$ ), find the partition ( $P_c$ ) that maximizes the similarity to the set:

$$P_c = \arg \max_{P \in \mathbb{P}_x} \sum_{P_i \in \mathcal{P}} \Gamma(P, P_i)$$

## Cooccurrence based methods

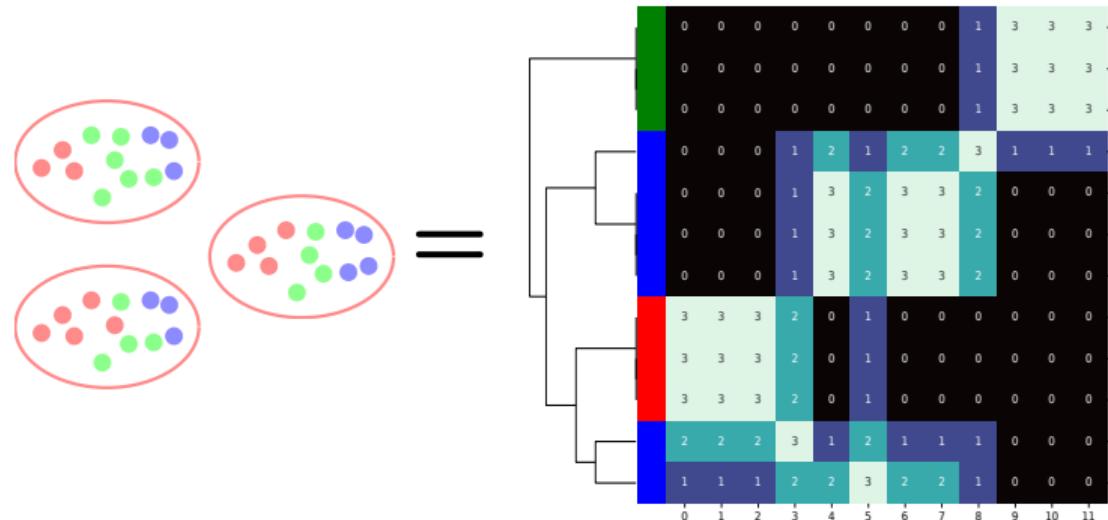
---

- ① First, solve the **labeling correspondence** problem
- ② After that, determine the consensus using different strategies of **voting**

Dimitriadou Weingessel, Hornik **Voting-Merging: An Ensemble Method for Clustering**  
Lecture Notes in Computer Science, 2001, 2130

1. Generate a clustering
2. Determine the correspondence with the current consensus
3. Each example gets a vote from their cluster assignment
4. Update the consensus

- **Co-Association matrix:** Count how many times a pair of examples are in the same cluster
  - Use the matrix as a similarity or a new set of characteristics
  - Apply a cluster algorithm to the information from the co-association matrix



Fred **Finding Consistent Clusters in Data Partitions** Multiple Classifier Systems, 2001, 309-318

Fred, Jain **Combining Multiple Clusterings Using Evidence Accumulation** IEEE Trans. Pattern Anal. Mach. Intell., 2005, 27, 835-850

1. Compute the co-association matrix
2. Apply hierarchical clustering (different criteria)
3. Use a heuristic to cut the resulting dendrogram

- Define consensus as a graph partitioning problem
- Different methods to build a graph or hypergraph from the partitions

Strehl, Ghosh **Cluster ensembles- A knowledge reuse framework for combining multiple partitions** Journal of Machine Learning Research, MIT Press, 2003, 3, 583-617

- Cluster based Similarity Partitioning Algorithm (CSPA)
- HyperGraph-Partitioning Algorithm (HGPA)
- Meta-CLustering Algorithm (MCLA)

- Compute a similarity matrix from the clusterings
- **Hyperedges matrix:** For all clusterings, compute an indicator matrix ( $H$ ) that represents the links among examples and clusters (Hypergraph)
- Compute the similarity matrix as:

$$S = \frac{1}{r} HH^T$$

where  $r$  is the number of clusterings

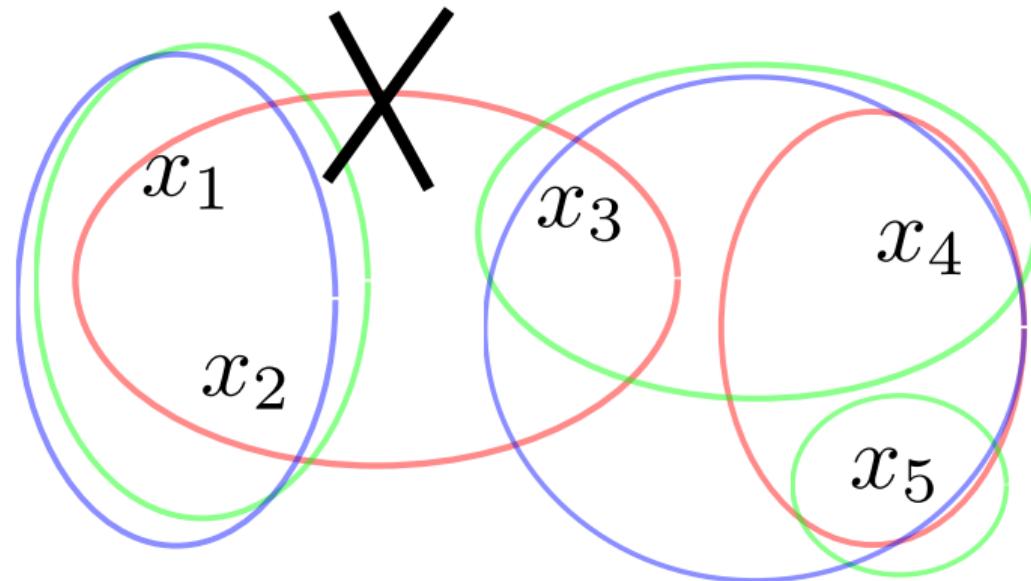
- Apply a graph partitioning algorithm to the distance matrix (METIS)
- **Drawback:** Quadratic cost in the number of examples  $O(n^2kr)$

	$C_1$	$C_2$	$C_3$		$C_{1,1}$	$C_{1,2}$	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$	$C_{3,1}$	$C_{3,2}$
$x_1$	1	2	1	$x_1$	1	0	0	1	0	1	0
$x_2$	1	2	1	$x_2$	1	0	0	1	0	1	0
$x_3$	1	1	2	$x_3$	1	0	1	0	0	0	1
$x_4$	2	1	2	$x_4$	0	1	1	0	0	0	1
$x_5$	2	3	2	$x_5$	0	1	0	0	1	0	1

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
$x_1$	1	1	1/3	0	0
$x_2$	1	1	1/3	0	0
$x_3$	1/3	1/3	1	2/3	1/3
$x_4$	0	0	2/3	1	2/3
$x_5$	0	0	1/3	2/3	1

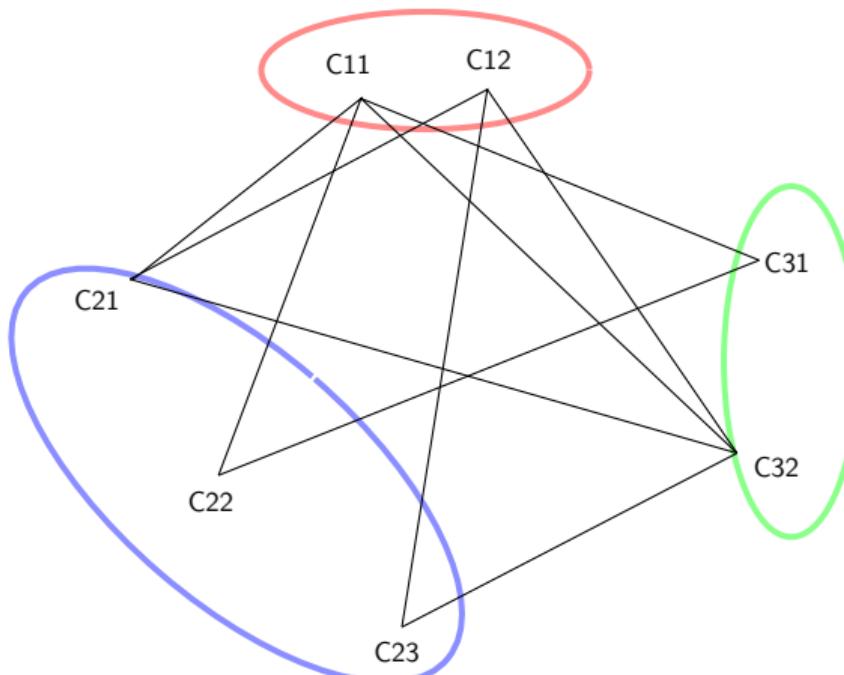
$$S =$$

- Partitions the hypergraph generated by the examples and their clusterings
- The indicator matrix is partitioned into  $k$  clusters of approximately the same size
- The METIS hypergraph partitioning algorithm is used
- Linear in the number of examples  $O(nkr)$



- ⑤ Group and collapse hyperedges and assign the objects to the hyperedge in which they participate the most
- ⑥ Algorithm
  1. Build a meta-graph with the hyperedges as vertices (edges have the vertices similarities as weights, Jaccard)
  2. Partition the hyperedges into  $k$  metaclusters
  3. Collapse the hyperedges of each metacluster
  4. Assign examples to their most associated metacluster
- ⑦ Linear in the number of examples  $O(nk^2r^2)$

	$C_{1,1}$	$C_{1,2}$	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$	$C_{3,1}$	$C_{3,2}$
$x_1$	1	0	0	1	0	1	0
$x_2$	1	0	0	1	0	1	0
$x_3$	1	0	1	0	0	0	1
$x_4$	0	1	1	0	0	0	1
$x_5$	0	1	0	0	1	0	1



- Information Theory measures are used to assess the similarity among the clusters of the partitions
- For instance:
  - Normalized Mutual Information
  - Category Utility
- The labels can be transformed to a new set of features for each example (measuring example coincidence)
- The new features can be used to partition the examples using a clustering algorithm

- ④ The problem is transformed into the estimation of the probability of assignment
- ④ The mixture is composed by the product of multinomial distributions, one for each clustering
- ④ Each example is described by the set of assignments of each clustering
- ④ An EM algorithm is used to find the probability distribution that maximized the agreement

## Median partition based methods

---

- Given a set of partitions ( $\mathcal{P}$ ) and a similarity function among partitions  $\Gamma(P_i, P_j)$ , the **Median Partition**  $P_c$  is the one that maximizes the similarity to the set

$$P_c = \arg \max_{P \in \mathbb{P}_x} \sum_{P_i \in \mathcal{P}} \Gamma(P, P_i)$$

- Has been proven to be a NP-hard problem for some similarity functions  $\Gamma$

- Based on the **agreements and disagreements of pairs** of examples between two partitions
  - Rand index, Jaccard coefficient, Mirkin distance (and their randomness adjusted versions)
- Based on **set matching**
  - Purity, F-measure
- Based on **information theory measures** (how much information two partitions share)
  - NMI, Variation of Information, V-measure

- ④ Best of k (the partition of the set that minimizes the distance)
- ④ Optimization using local search: Hill Climbing, Simulated Annealing, Genetic Algorithms
  - Perform a movement of examples between two clusters of the current solution to improve the partition
- ④ Non Negative Matrix Factorization
  - Find the partition matrix closest to the averaged association matrix of a set of partitions

This Python Notebook has examples of consensus clustering

- ⑤ Consensus clustering Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# Clustering of Structured Data

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Introduction

---

- There are some domains where patterns are more complex
- In these domains examples are related to each other
- Mining these relationships is more interesting than obtaining patterns from the examples individually
- For instance:
  - Temporal domains
  - Relational databases
  - Structured instances (trees, graphs)
- Usually these domains need specific methods

# Sequences

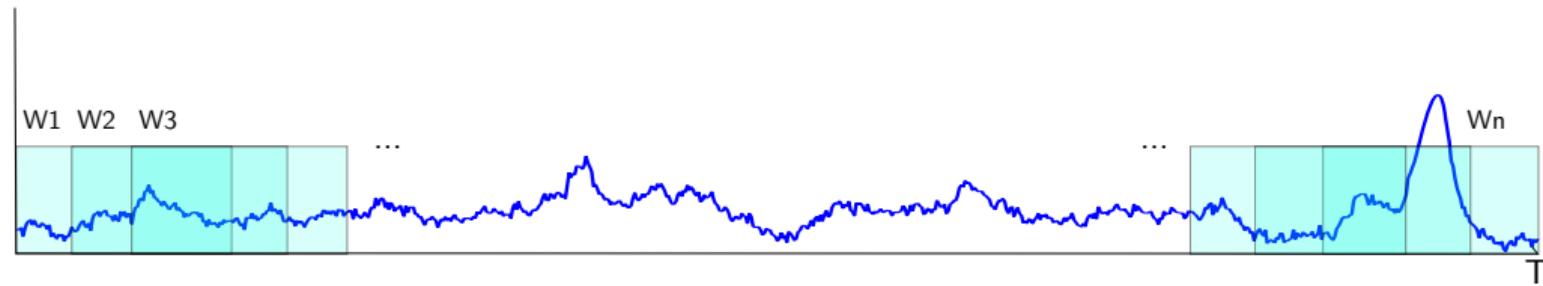
---

- Data have a sequential relationship among examples
- We can have a unique sequence, or a set of sequences
- The classical techniques from time series analysis do not apply (AR, ARIMA, GARCH, Kalman filter...)
- What makes different these data?
  - Usually qualitative data
  - Very short series or long series that have to be segmented
  - Interest in the relationships among series
  - Interest only in a part of the series (episodes, anomalies, novelty... )

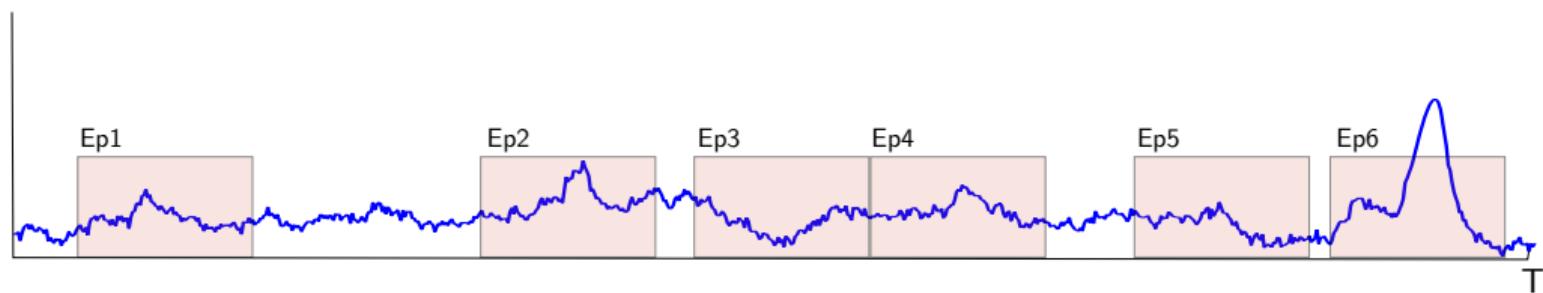
- ⑤ **Clustering of temporal series:** Clustering algorithms applied to a set of short series
  - How to segment a unique series in a set of series? what parts are interesting?
  - Representation of the series, representation of the groups
  - New distance/similarity measures (scale invariant, shape distances...)

- ⑤ A unique series is provided, and we must divide it into a set of subseries (series segmentation)
- ⑤ Extract subseries using a sliding window
  - Width of the window
  - overlapping/non overlapping
- ⑤ Only some parts of the series are the target (episodes)
  - Anomaly Detection
  - Change Detection
- ⑤ Be careful with unbalanced datasets

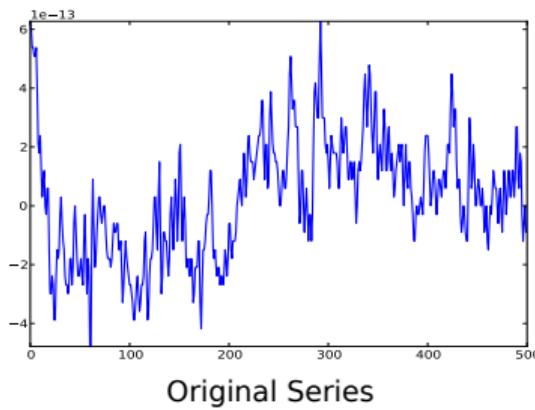
## Sliding Window



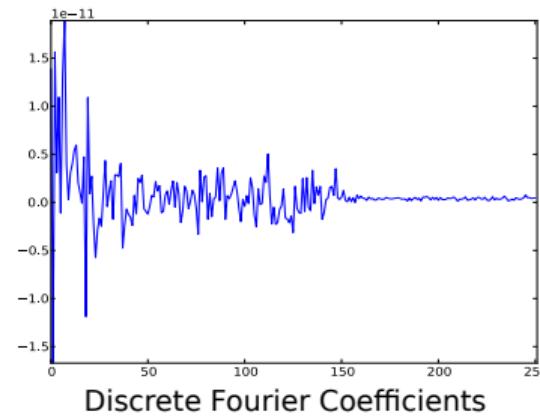
## Episodes



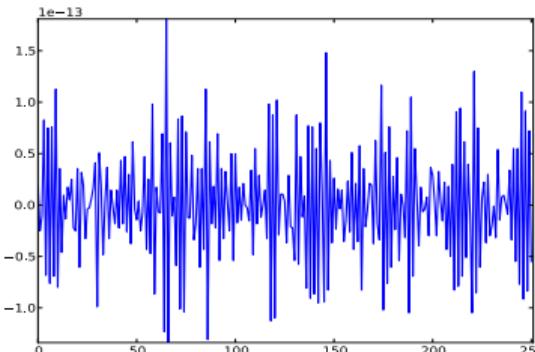
- ⑤ Raw time series are not always the best input
- ⑥ **Feature extraction:** Generate informative features
  - Frequency/Time domain features (Fourier, Wavelets. . . )
  - Extreme points (maximum, minima, inflection points)
  - Probabilistic models (Hidden Markov Models, ARIMA)
  - Symbolic representation: SAX, SFA



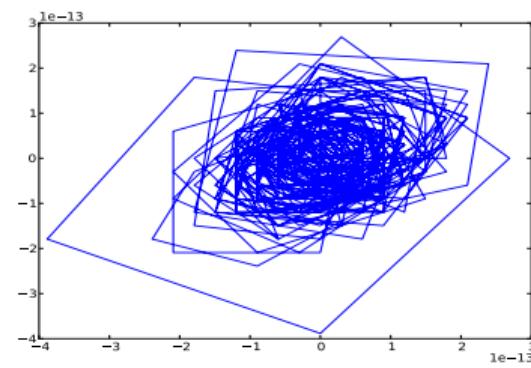
Original Series



Discrete Fourier Coefficients

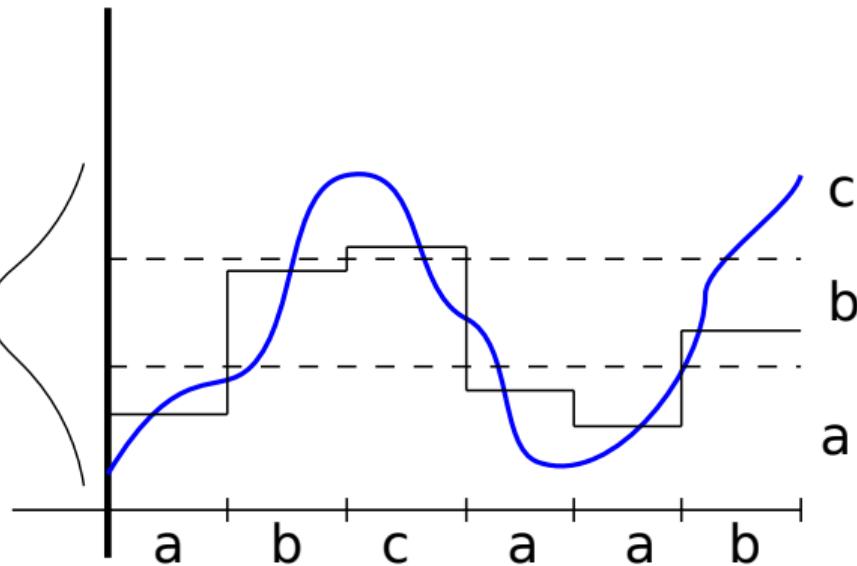
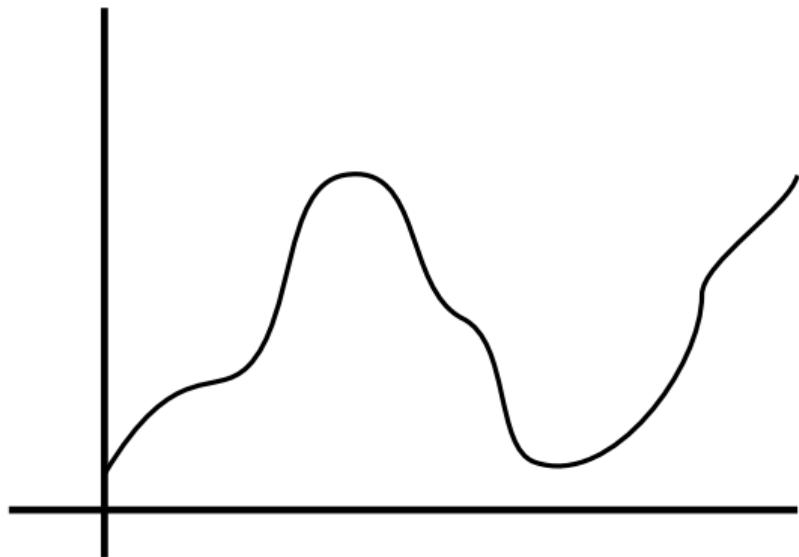


Discrete Wavelet Coefficients

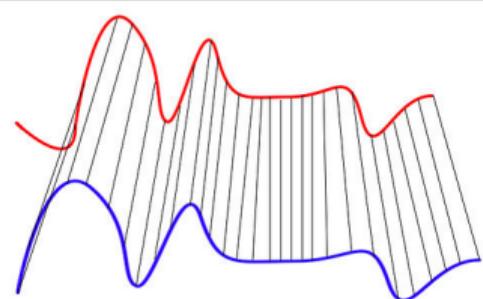
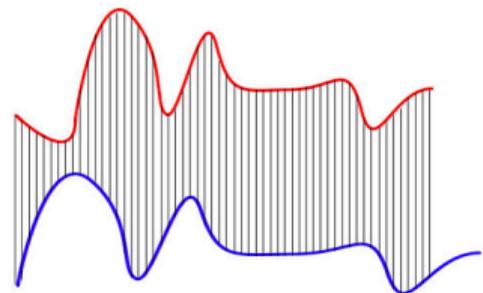


1-Lag Series

- Transforms a time series into a set of discrete symbols
- Data are discretized to strings of length  $M$  with a vocabulary of size  $N$
- Algorithm:
  - Standardize the series ( $\mathcal{N}(0, 1)$ )
  - For each of the  $M$  subwindows compute their mean
  - Map each mean to a Gaussian distribution discretized to  $N$  segments of equal frequency
- Transformed data can be used as a string, or an integer valued series

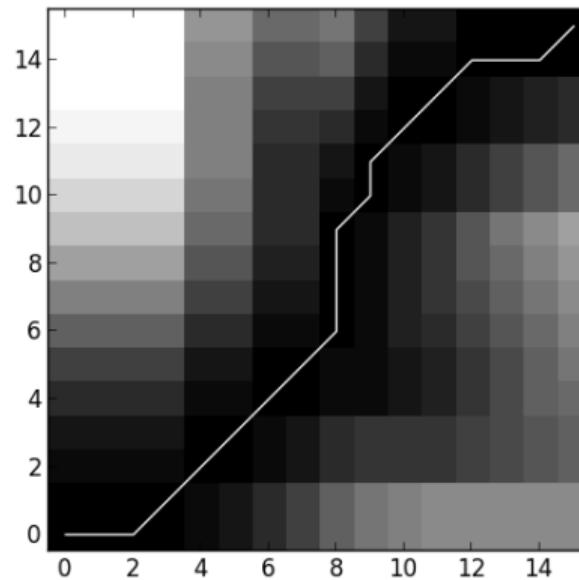
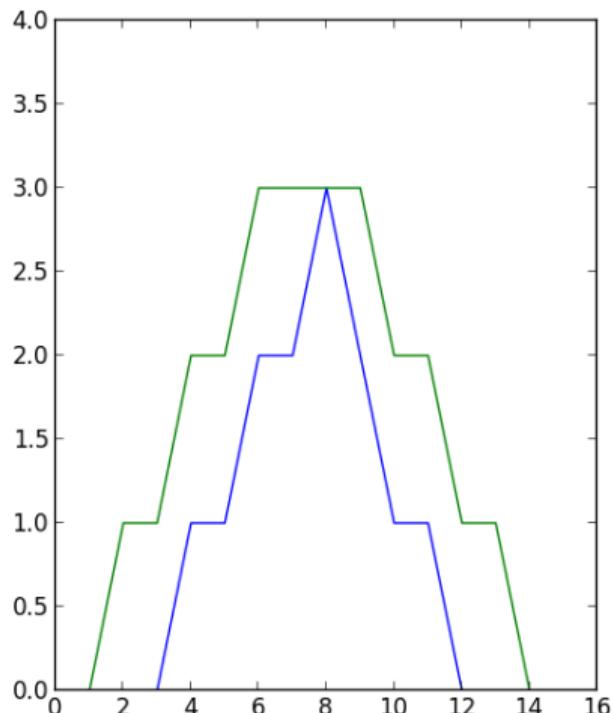


- Usual distance functions ignore time dynamic
  - Euclidean, hamming...
- Patterns in series contain noise, time/amplitude scaling, translations
  - Dynamic Time Warping (DTW)
  - Longest Common Subsequence (LCSS)
  - Edit Distance with Real Penalty (ERP)
  - Edit Distance on Real Sequence (EDR)
  - Spatial Assembly Distance (SpADe)



Dynamic Time Warping Matching

- Matches the dynamic of the series
- Series can be of different lengths
- The cost of matching two points is their distance (e.g. euclidean)
- A point from one series can be matched to multiple points of the other
- DTW is the minimum cost of the possible matchings
- Computing the distance is  $O(n^2)$ , but it can be reduced by limiting the number of points able to match to a point



# Clustering of Data Streams

---

## Data streams: Modeling an on-line continuous series of data

- Each item of the series is an example (one value, a vector of values, structured data)
  - For instance, sensory data (one or multiple synchronized data), stream of documents (twitter/news)
- Data are generated from a set of clusters (stable or changing over time)
  - For instance, states from a process or semantic topics

- ⑤ Data are processed incrementally (model changes with time)
  - Only the current model
  - Periodic snapshots
- ⑥ Different goals:
  - Model the domain
  - Detect anomalies/novelty/bursts
  - Detect change (Concept drift)

- Clustering has an on-line and an off-line phase
- Elements:
  - The **data structure** used to **summarize** the data
  - The **window model** used to decide the influence of the current and past data
  - The mechanism for identifying **outliers**
  - The **clustering algorithm** used to obtain the partition of the data

- Involved in the on-line phase
- Data are summarized using sufficient statistics (num of examples, sum of values, sum of squared products of values...)
- Usually a hierarchical datastructure (different levels of granularity)
- Indexing structure that can be updated incrementally
- Stores raw data or prototypes depending on space constraints

- Sliding window model

- Fixed time window
- Only data inside the window updates the structure

- Damped window model

- A weight is associated to examples and clusters
- Influence of data depends on time, old data fades away or are discarded

- Landmark window model

- Defines points of interest in time or amount of data
- Data before the landmark are discarded

- Difficult task because data evolve with time
- Most methods work around the idea of **microclusters**
- A microcluster represents a dense area in the space of examples
- The indexing structure tracks the evolution of the microclusters
- Different thresholds determine if a microcluster is kept or discarded

Aggarwal et al. **On Clustering Massive Data Streams: A Summarization Paradigm**  
Data Streams-Models and Algorithms, Springer, 2007, 31, 9-38

① **On-line phase:**

- Maintains microclusters (more than final number of clusters)
- New data is incorporated to a microcluster or generates new microclusters
- The number of microclusters is fixed, they are merged to maintain the number
- Periodically the microclusters are stored

② **Off-line phase:**

- Given a time window the stored microclusters are used to compute the microclusters inside the time frame
- K-means used to compute the clusters for the time window

Cao, Ester, Qian, Zhou **Density-Based Clustering over an Evolving Data Stream with Noise** Proceedings of the Sixth SIAM International Conference on Data Mining, 2006

① **On-line phase:**

- Core-micro-clusters (a weighted sum of close points)
- The weight of a point fades exponentially with time (damping window model)
- New examples are merged and mc are classified as:
  - core-mc, sets of points with weight over a threshold
  - potential-mc
  - outlier-mc, sets of points with weight below a threshold
- outlier-mc disappear with time

② **Off-line phase:** Modified version of DBSCAN

This Python Notebook has examples of time series clustering

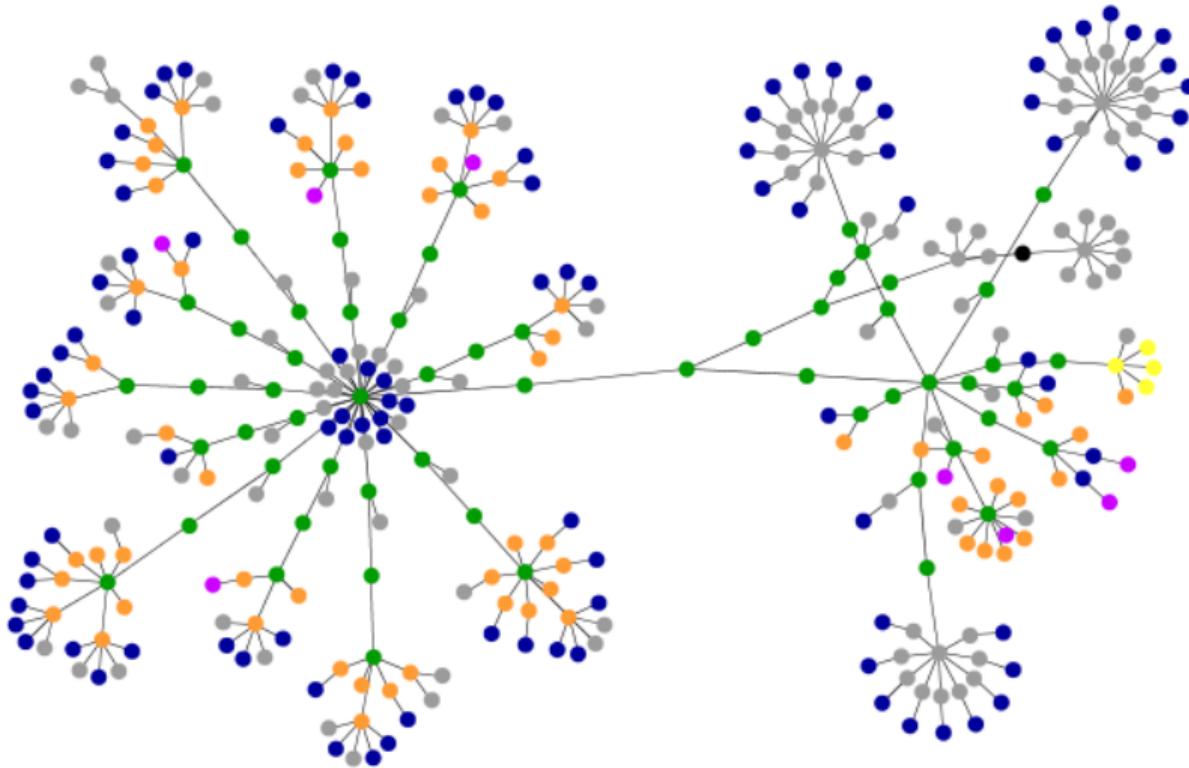
- ⑤ Time Series clustering Notebook ([click here](#) to go to the url)

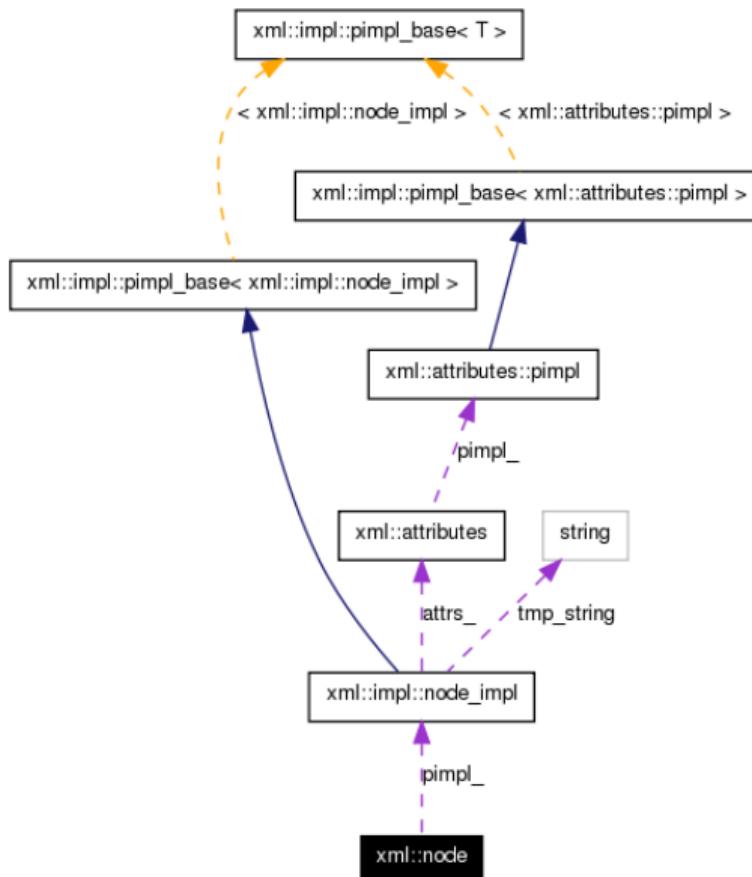
If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

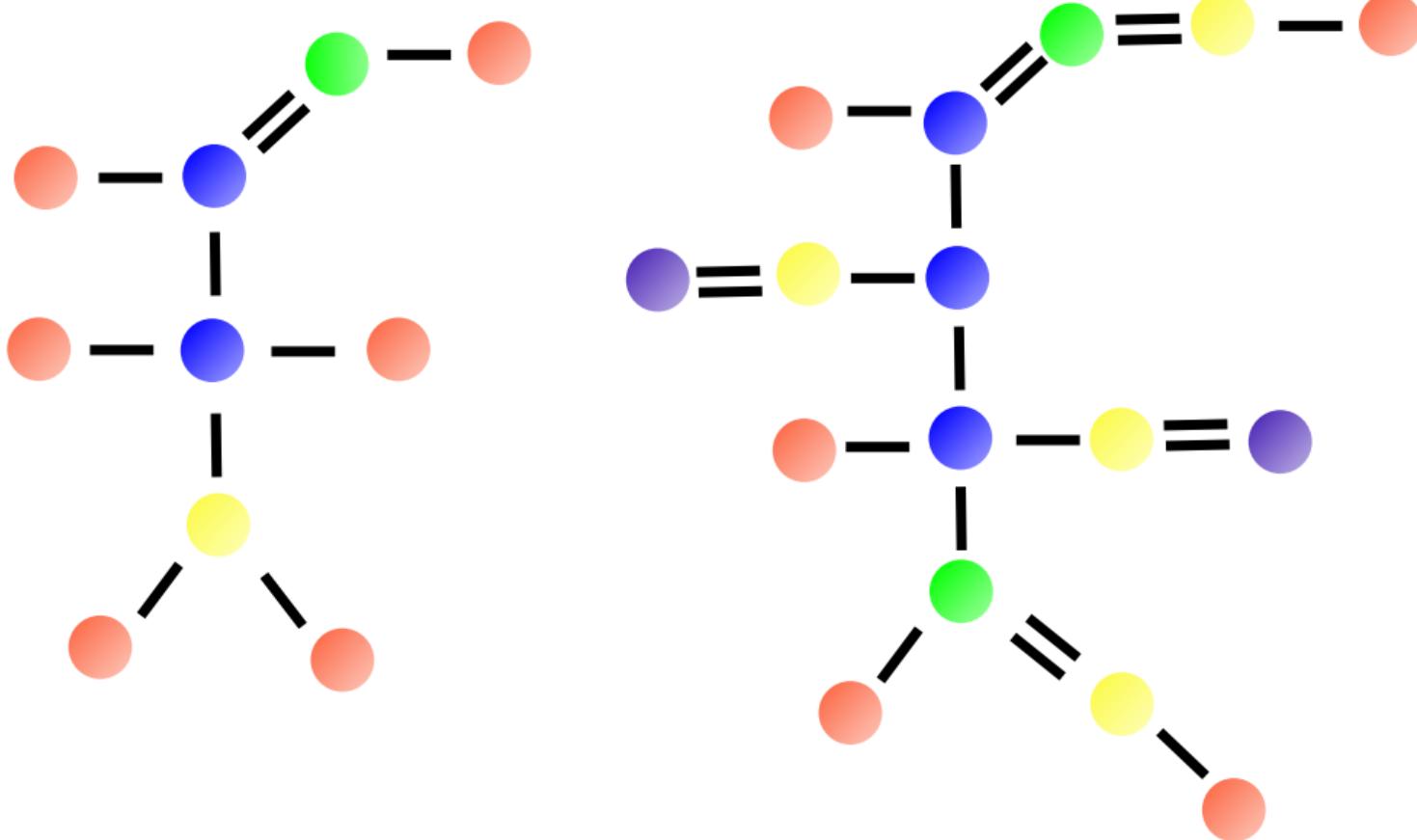
# Graph mining

---

- ⑤ A lot of information has a relational structure
- ⑥ Methods and models used for unstructured data are not expressive enough
- ⑦ Sometimes structure can be flattened, but lots of interesting information is lost
  - Relational database  $\Rightarrow$  unique merged table
  - Attributes representing relations  $\Rightarrow$  inapplicable attributes
  - Graph data  $\Rightarrow$  strings based on graph traversal
  - Documents  $\Rightarrow$  bag of words







- ④ All these types of data have in common that can be represented using graphs and trees
- ④ Historically we can find different approaches to the discovery of patterns in graphs/trees:
  - Inductive logic programming: Structure is represented using logic formulas
  - Graph algorithms
    - Classic algorithms for detecting dense subgraphs (cliques)
    - Graph isomorphism algorithms
    - Graph partitioning algorithms

- ⑤ Most of the problems used to discover structures in graphs are NP-Hard
  - Graph partitioning (Not for bi-partitioning)
  - Graph isomorphism
- ⑥ Two different problems:
  - Clustering large graphs (only one structure)  $\Rightarrow$  Partitioning
  - Clustering sets of graphs  $\Rightarrow$  common substructures

- Some information can be described as a large graph (several instances connected by different relations)
  - For instance: Social networks, Web pages,
- We want to discover interesting substructures by:
  - Dividing the graph in subgraphs (k-way partitioning, node clustering)
  - Extracting dense substructures

- The simplest partitioning of a graph is to divide the graph in two subgraphs
- We assume that edges have values as labels (similarity, distance...)
- This problem is the *minimum cut-problem*:  
“Given a graph, divide the set of nodes in two groups, so the cost of the edges connecting the nodes between the groups is minimum”
- This problem is related to the *maximum flow problem* that can be solved in polynomial time

- ④ Randomized algorithm that approximates the min cut of a graph for undirected graphs
- ④ Computational cost  $O(|E|)$
- ④ Has to be repeated  $O(|V| \log |E|)$  to have high probability of finding the global minimum
- ④ Algorithm:
  1. Pick an edge at random and join its vertices, reconnect the remaining vertices to the new vertex
  2. Repeat until only two vertices remain

- ④ The general problem is NP-hard
- ④ It can be solved approximately by local search algorithms (hill climbing, simulated annealing)
- ④ Kernighan-Lin Algorithm:
  1. Start with a random cut of the graph (k-clusters)
  2. Interchange a pair of nodes from different partitions that reduces the cut
  3. Iterate until no improvement
- ④ Different variations of this algorithm changing the strategy for selecting the pair of nodes to interchange

Classical clustering algorithms can be adapted to obtain a graph partition

⑤ K-means and K-medoids variations

- Nodes of the graphs as prototypes
- Objective functions to define node membership to clusters (geodesic distance)
- Network structure indices

⑥ Spectral Clustering

- Define the Laplacian matrix from the graph
- Perform the eigendecomposition
- The largest Eigenvalues determine the number of clusters

- ④ Graph partitioning is the problem of **Community Discovery** in the area of Social Networks Analysis
- ④ Based on graph measures detecting dense connected areas
- ④ **Edge betweenness centrality:**

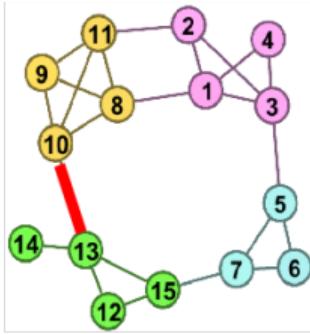
$$B(e) = \frac{\text{NumConstrainedPaths}(e, i, j)}{\text{NumShortPaths}(i, j)}$$

- ④ **Random Walk Betweeness:** Compute how often a random walk starting on node  $i$  passes through node  $j$
- ④ **Modularity:** Percentage of edges within communities compared with the expected number if they are not a community

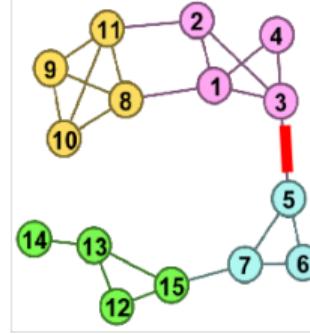
## ⑤ Girvan-Newman Algorithm (Betweenness)

1. Rank edges by  $B(e)$
2. Delete edge with the highest score
3. Iterate until a specific criteria holds (e.g. number of components)

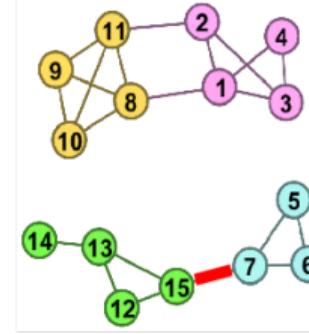
1



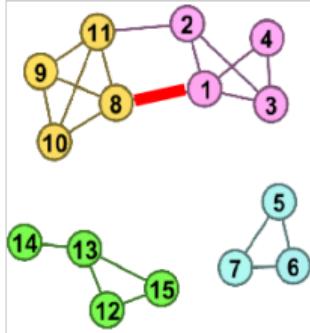
2



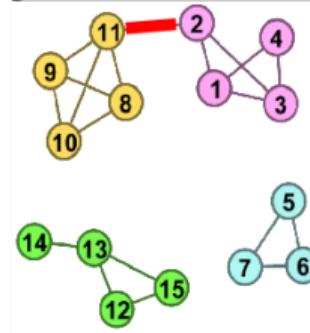
3



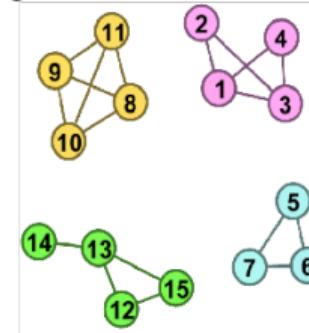
4



5

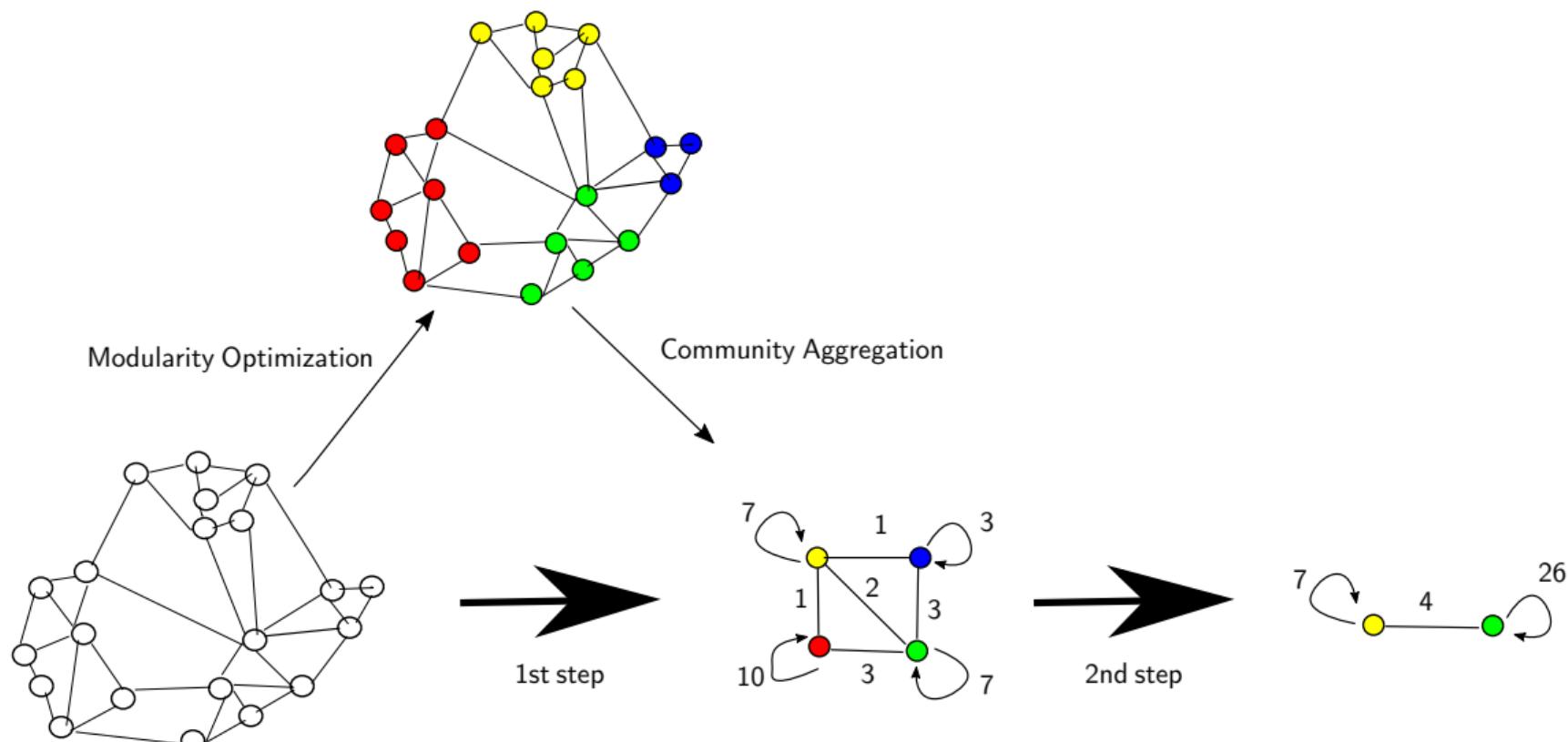


6



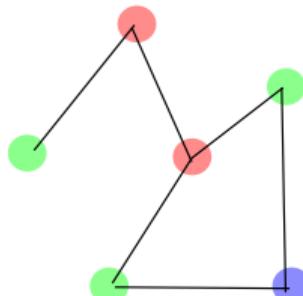
## ⑤ **Louvain Algorithm** (Modularity)

1. Begin with a community for each node of the graph
2. Repeat until no change:
  - For each node  $i$  in the graph and for all its neighbors  $j$  of  $i$ , consider the effect on the modularity of changing the community of  $i$  to the community of  $j$ . Change the node if modularity increases
3. Build a new graph where the new nodes are the communities, and the weights of the edges connecting communities are the sum of the edges among the nodes in the original graph
4. Repeat from 2 until no changes

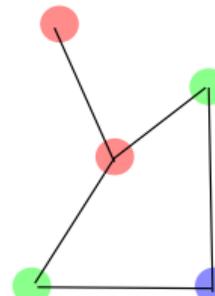


- Some information can be described as a collection of graphs
  - For example: XML documents, chemical molecules
- We look at a graph as a complex object
- We have to adapt the elements of clustering algorithms to these objects:
  - Distance measures to compare graphs
  - Summarization of graphs as prototypes

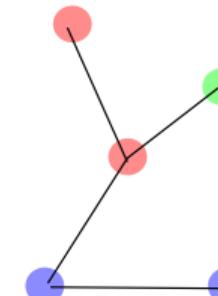
- Edit distance can be adapted to graphs
- Define add/delete/substitute costs for edges and vertices
- Different costs leads to different functions (sometimes do not hold distance properties)
- Distance is the minimum cost path that transforms one graph into another



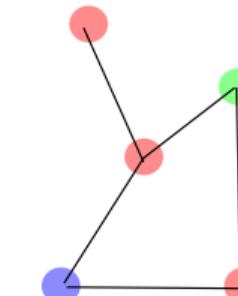
G1



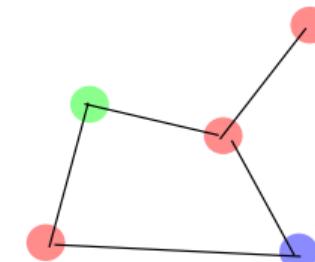
Del V



Subst V



Subst V



G2

- Specific graph kernels can be used to embed the data in a metric space
- A base similarity or distance function can be used (like graph edit distance)
  - Diffusion kernels (extend similarity to closest neighbors)
  - Walk kernels (computing the similarity of traversal paths)
  - RBF kernels

This Python Notebook has examples of community discovery using geolocation information from Twitter for London, Paris and Barcelona

- ⑤ Dense Subgraphs Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# Semi-supervised Clustering

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Semisupervised Clustering

---

- Sometimes we have available some information about the dataset we are analyzing unsupervisedly
- Could be interesting to incorporate this information to the clustering process in order to:
  - Bias the search of the algorithm toward the solutions more consistent with our knowledge
  - Improve the quality of the result reducing the algorithm natural bias (predictivity/stability)

- ⑤ The information that we have available can be of different kinds:
  - Sets of labeled instances
  - Constrains among certain instances: Instances that have to be in the same group/instances that can not belong to the same group
  - General information about the properties that the instances of a group must hold

- ⑤ It will depend on the model we can obtain
  1. Begin with a prior model that changes how the search is performed
  2. Bias the search, pruning the models that are not consistent with the semisupervised knowledge
  3. Modify the similarity among instances to match the constraints imposed by the prior knowledge

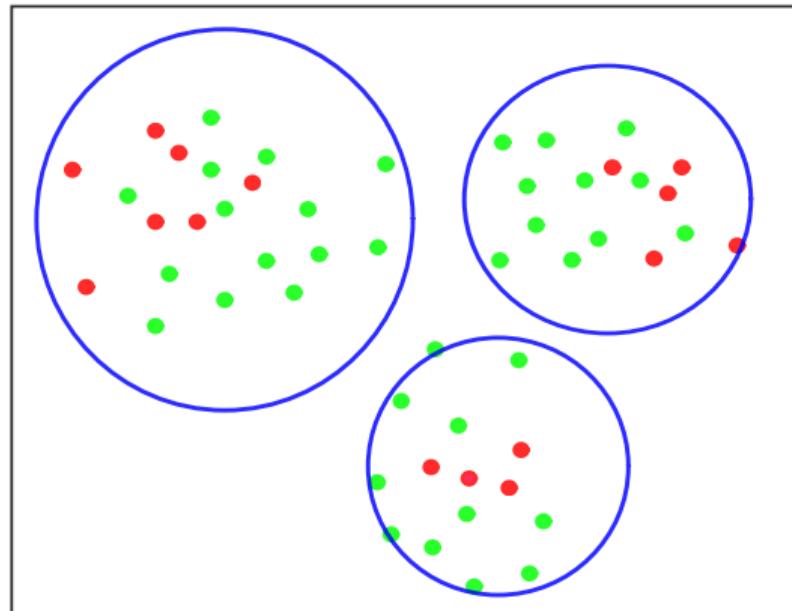
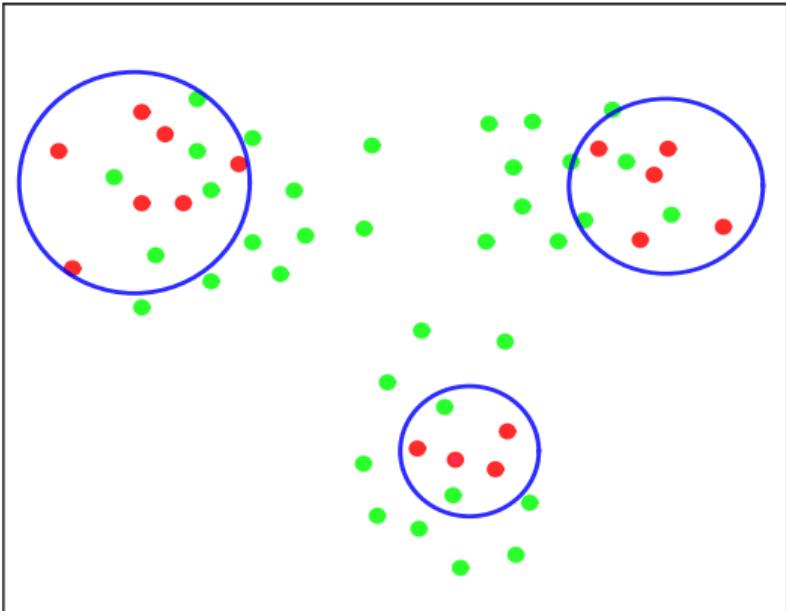
## Semisupervised Cluste- ring/Labeled Examples

---

- ⑤ Assuming that we have some labeled examples, these can be used to obtain an initial model
- ⑤ We only have to know what examples belong to clusters, the actual clusters are not needed
- ⑤ We can begin from this model the clustering process, used as a starting point of the search
- ⑤ This initial model changes the search and biases the final model

[Basu, Banerjee, Mooney \*\*Semi supervised clustering by seeding\*\* ICML 2002](#)

- Ⓐ Algorithm based on K-means
- Ⓐ The usual initialization of K-means is by selecting randomly the initial prototypes
- Ⓐ Two alternatives:
  - Use the labeled examples to build the initial prototypes (seeding)
  - Use the labeled examples to build the initial prototypes and constrain the model, so the labeled examples are always in the initial clusters (seed and constraint)
- Ⓐ The initial prototypes give an initial probability distribution for the clustering



---

**Algorithm:** Seeded-KMeans

**Input:** The dataset  $\mathcal{X}$ , the number of clusters  $K$ , a set  $\mathcal{S}$  of labeled instances (k groups)

**Output:** A partition of  $\mathcal{X}$  in  $K$  groups

**begin**

    Compute  $K$  initial prototypes ( $\mu_i$ ) using the labeled instances

**repeat**

        Assign each example from  $\mathcal{X}$  to their nearest prototype  $\mu_i$

        Recompute the prototype  $\mu_i$  with the examples assigned

**until** Convergence

---

**Algorithm:** Constrained-KMeans

**Input:** The dataset  $\mathcal{X}$ , the number of clusters  $K$ , a set  $\mathcal{S}$  of labeled instances ( $k$  groups)

**Output:** A partition of  $\mathcal{X}$  in  $K$  groups

**begin**

    Compute  $K$  initial prototypes ( $\mu_i$ ) using the labeled instances

**repeat**

        Maintain the examples from  $\mathcal{S}$  in their initial classes

        Assign each example from  $\mathcal{X}$  to their nearest prototype  $\mu_i$

        Recompute the prototype  $\mu_i$  with the examples assigned

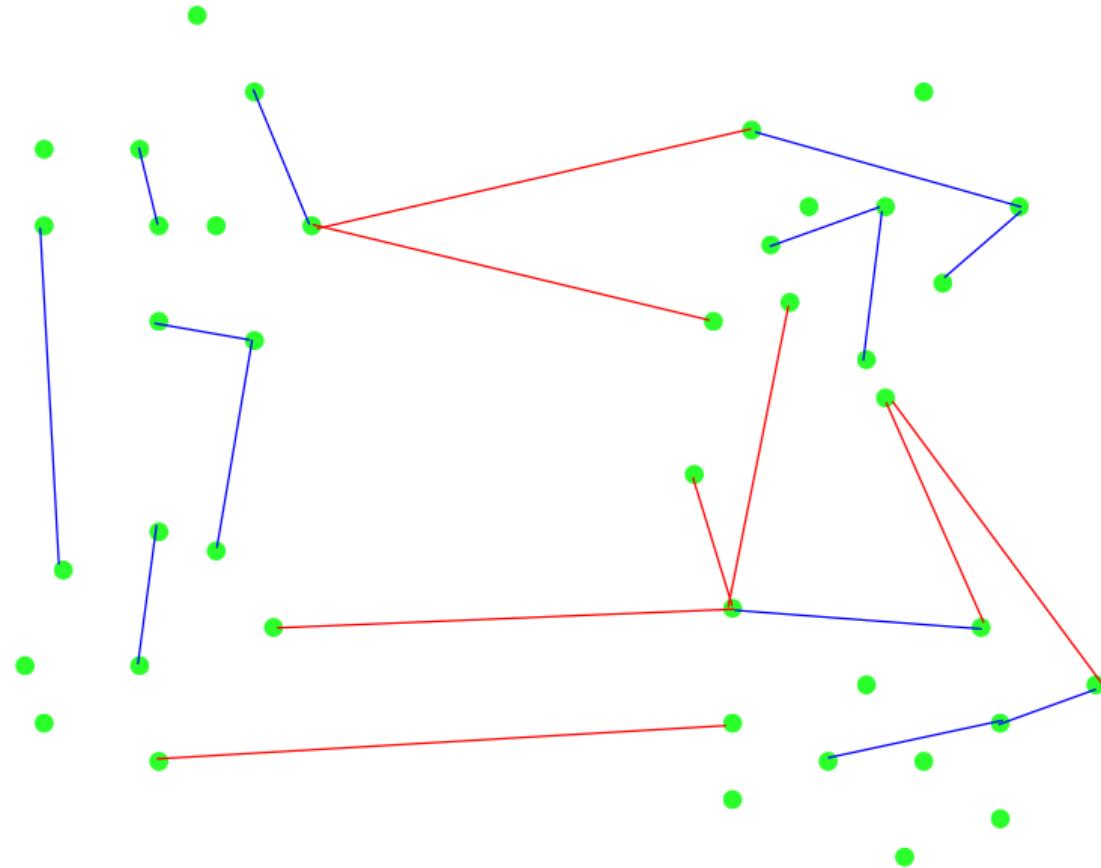
**until** *Convergence*

---

## Semisupervised Cluste- ring/Constraints

---

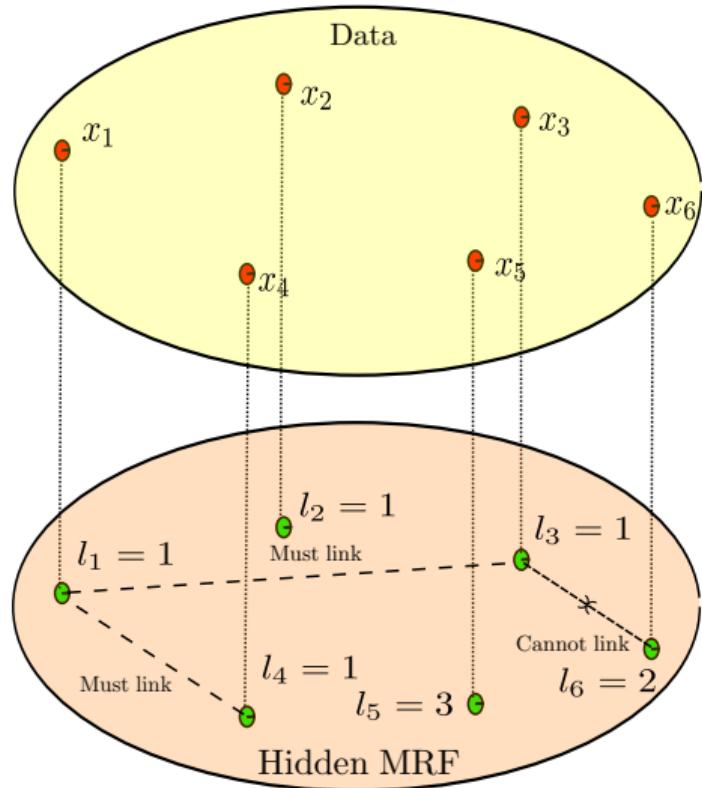
- ⑤ To have labeled examples means that the number of clusters and something about the characteristic of the data are known
- ⑤ Sometimes it is easier to have information about if two examples belong to the same or different clusters
- ⑤ This information can be expressed by means of constraints among examples: **must links** and **cannot links**
- ⑤ This information can be used to bias the search and only look for models that maintain these constraints



Basu, Bilenko, Mooney **A probabilistic framework for semi-supervised clustering** ICML 2002

- Algorithm based on K-means (spherical clusters based on prototypes)
- A set of must-link and cannot-link constraints is defined over a subset of examples
- The quality function of the K-means algorithm is modified to bias the search
- A hidden markov random field is defined using the constraints

- The labels of the examples can be used to define a markov random field
- The must-links and cannot-links define the dependence among the variables
- The clustering of the examples has to maximize the probability of the hidden markov random field



- A new objective function for the K-Means is defined
- The main idea is to introduce a penalty term to the objective function that:
  - Penalizes the clustering that puts examples with must-links in different clusters
  - Penalizes the clustering that puts examples with cannot-links in the same cluster
- This penalty has to be proportional to the distance among the instances

**Algorithm:** HMRF-KMeans

**Input:** The data  $\mathcal{X}$ , the num of clusters  $K$ , must and cannot links, a distance function  $D$  and weights for violating the constraints

**Output:** A partition of  $\mathcal{X}$  in  $K$  groups

**begin**

    Compute  $K$  initial prototypes ( $\mu_i$ ) using constraints

**repeat**

        E-step: Reassign the labels of the examples using the prototypes ( $\mu_i$ ) to minimize  $\mathcal{J}_{obj}$

        M-step: Given the cluster labels recalculate cluster centroids to minimize  $\mathcal{J}_{obj}$

**until** Convergence

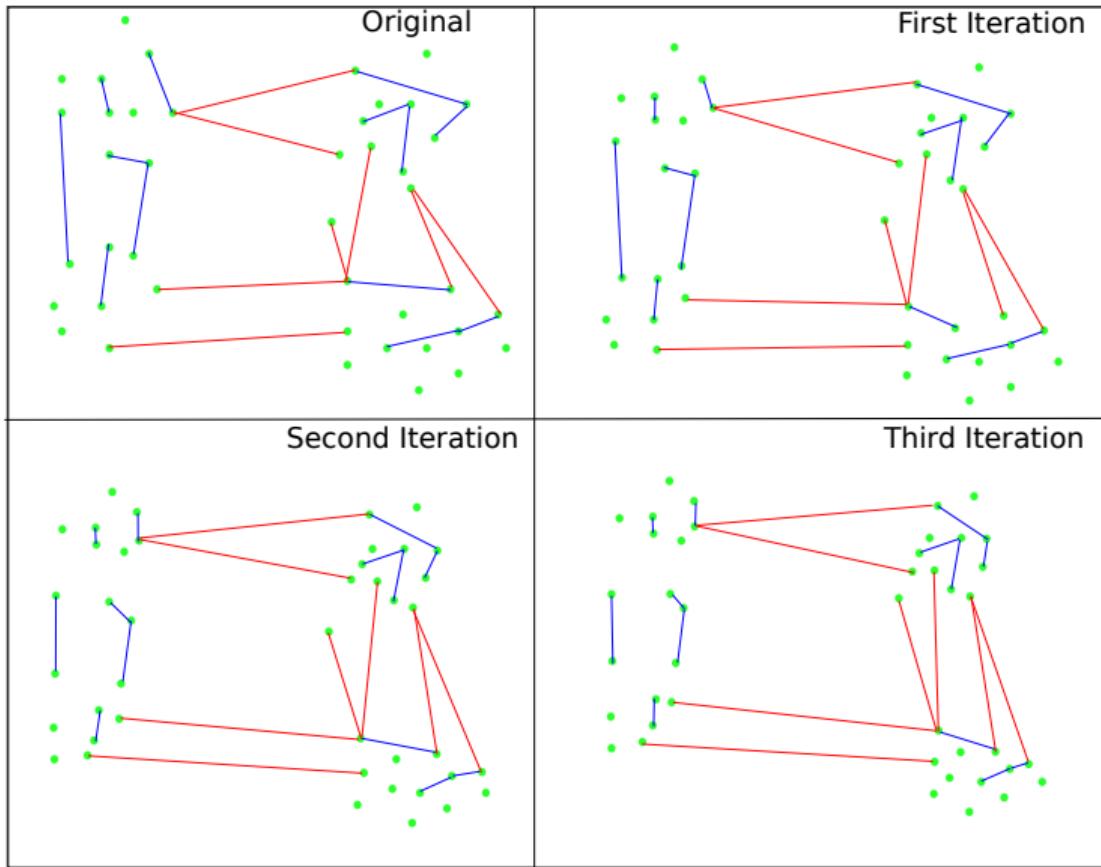
---

# Semisupervised Cluste- ring/Distance Learning

---

- ⑤ Other approach consists on learning a more adequate distance function to fulfill the constraints
- ⑥ The constraints are used as a guide to find a distance matrix that represents the relations among examples
- ⑦ The problem can be defined as an optimization problem that optimizes the distances among examples with respect to the constraints
- ⑧ These methods are related to kernel methods, the goal is to learn a Kernel matrix that represents a new space where the instances have appropriate distances

- ⑤ Relevant Component Analysis [Yeung, Chang (2006)] (optimization of linear combination of distance kernels generated by must and cannot links)
- ⑥ Optimization with the Graph Spectral Matrix, maintaining the constraints in the new space, and the structure of the original space
- ⑦ Learning of Mahalanobis distances: separate/approach the different dimensions to match the constraints
- ⑧ Kernel Clustering (Kernel K-means) with kernel matrix learning via regularization



# Association Rules

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



## Association Rules

---

- ④ A **Binary** database is a database where each row is composed of binary attributes

	A	B	C	D	...
T1	1	0	0	1	...
T2	0	1	1	1	...
T3	1	0	1	0	...
T4	0	0	1	0	...

- ⑤ From this database frequent patterns of occurrence can be discovered

- ④ This kind of databases appear for example on transactional data (market basket analysis)
- ④ An association rule represents coocurrence of events in the database

TID	Items
1	Bread, Chips, Beer, Yogourt, Eggs
2	Flour, Beer, Eggs, Butter,
3	Bread, Ham, Eggs, Butter, Milk
4	Flour, Eggs, Butter, Chocolate
5	Beer, Chips, Bacon, Eggs

$$\{\text{Flour}\} \rightarrow \{\text{Eggs}\}$$
$$\{\text{Beer}\} \rightarrow \{\text{Chips}\}$$
$$\{\text{Bacon}\} \rightarrow \{\text{Eggs}\}$$

- ④ We define  $R$  as the set of attributes of the database
- ④ We define  $X$  as a subset of attributes from  $R$ .
- ④ We say that  $X$  is a **pattern** from DB if there is any row where all the attributes of  $X$  are 1.
- ④ We define the **support** (*frequency*) of a pattern  $X$  as the function:

$$fr(X) = \frac{|M(X, r)|}{|r|}$$

- ④ Where  $|M(X, r)|$  is the number of times that  $X$  appears in the DB and  $|r|$  is the size of the BD.

- Given a minimum support ( $min\_sup \in [0, 1]$ ), we say that the pattern  $X$  is frequent (**frequent itemset**) if:

$$fr(X) \geq min\_sup$$

- $\mathcal{F}(r, min\_sup)$  is the set of patterns that are frequent in :

$$\mathcal{F}(r, min\_sup) = \{X \subseteq R / fr(X) \geq min\_sup\}$$

- Given a BD with  $R$  attributes and  $X$  and  $Y$  attribute subsets, with  $X \cap Y = \emptyset$ , an **association rule** is the expression:

$$X \Rightarrow Y$$

- $conf(X \Rightarrow Y)$  is the **confidence** of an association rule, computed as:

$$conf(X \Rightarrow Y) = \frac{fr(X \cup Y)}{fr(X)}$$

- We consider a minimum value of confidence ( $min\_conf \in [0, 1]$ )

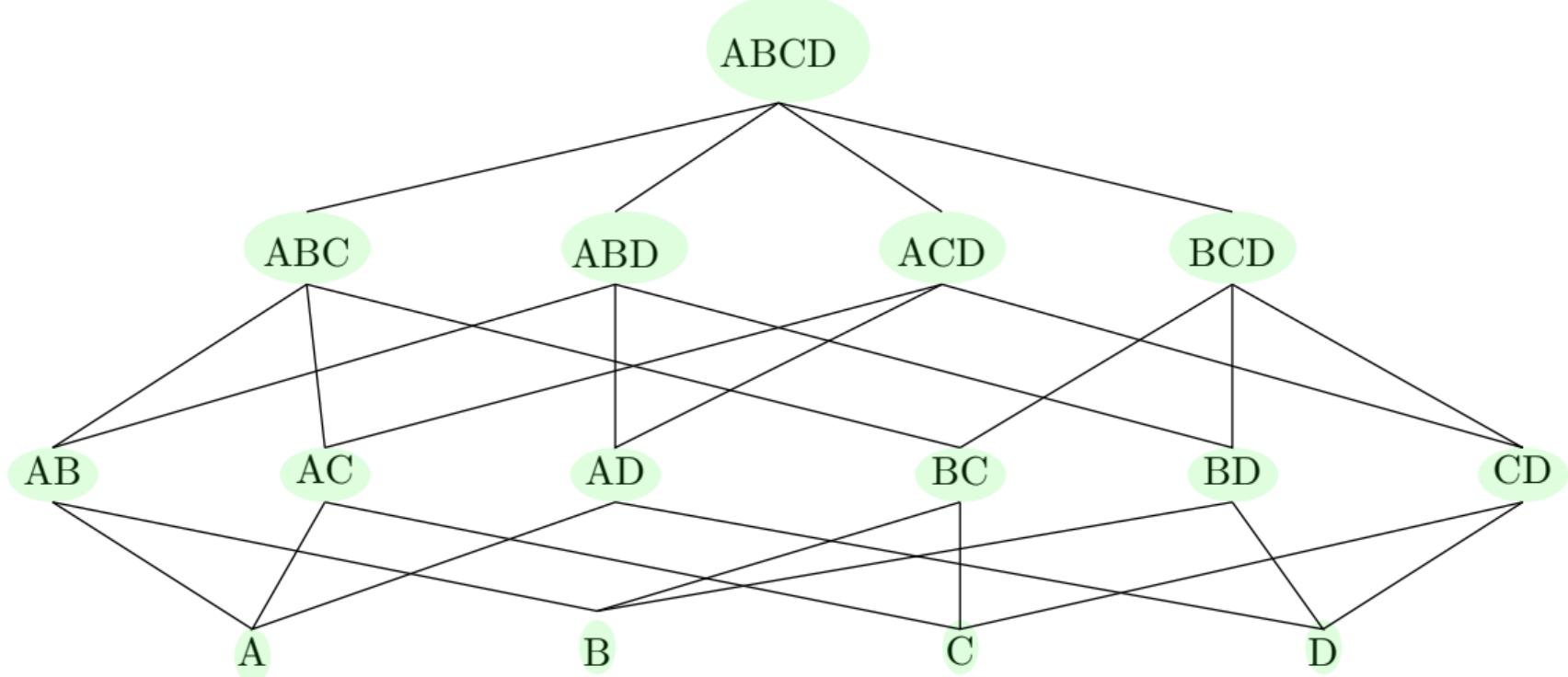
# Apriori algorithm

---

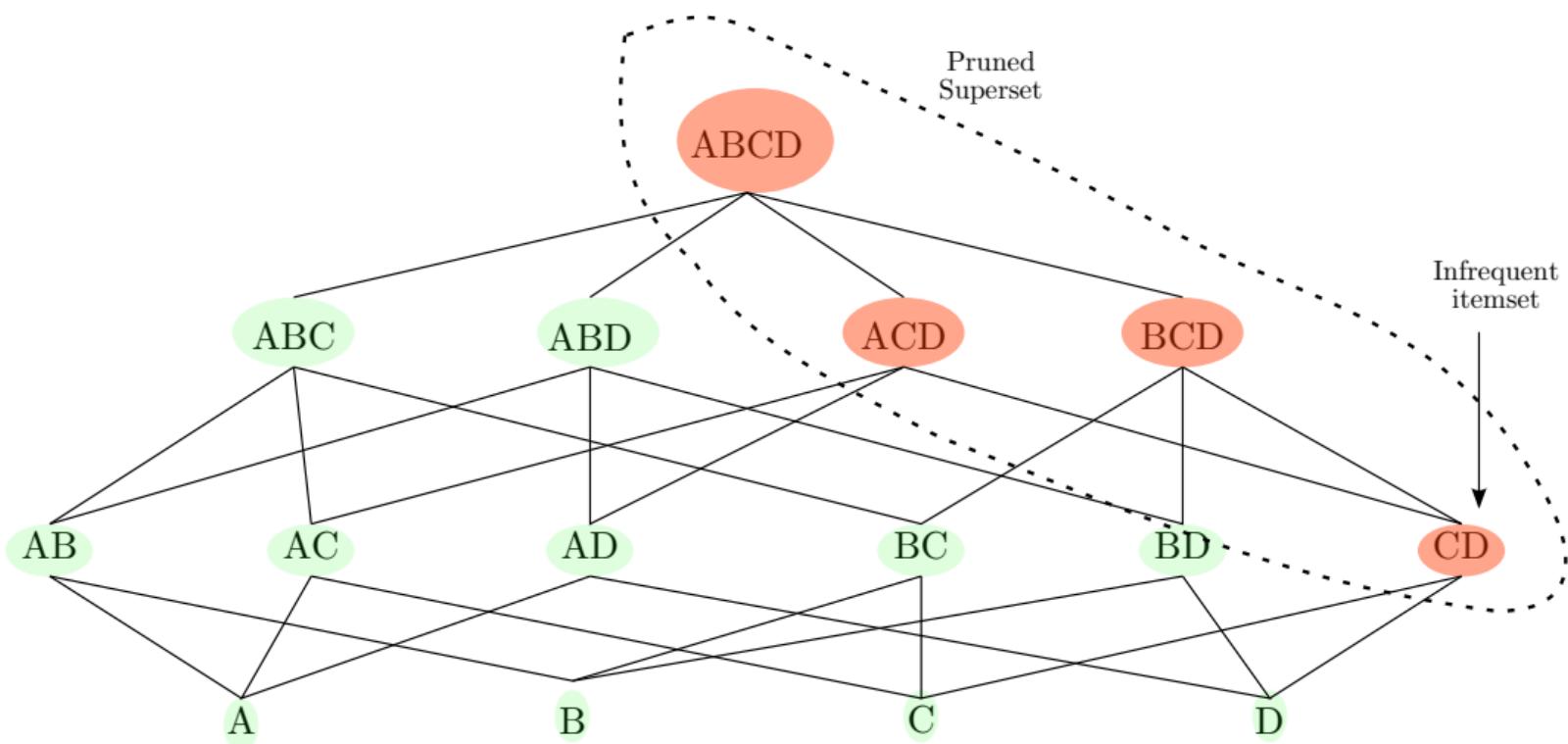
- Given a minimum confidence and a minimum support, a association rule  $(X \Rightarrow Y)$  exists in a DB if:

$$(fr(X \cup Y) \geq min\_sup) \wedge (conf(X \Rightarrow Y) \geq min\_conf)$$

- Trivial approach:
  - It is enough to find all frequent subsets from  $R$
  - We have to explore  $\forall X \forall Y ((X \subseteq R) \wedge (Y \subseteq X))$  the association rule  $(X - Y) \Rightarrow Y$
- There are  $2^{|R|}$  candidates



- We need ways to prune the search space
- Given  $X$  and  $Y$  with  $Y \subseteq X$ :
  - If  $fr(Y) \geq fr(X)$ , if  $X$  is frequent,  $Y$  also is frequent
  - If any subset  $Y$  from  $X$  is not frequent then  $X$  is not frequent
- This is known as the **anti-monotone** property of support
- A feasible exploration approach is
  - Find the frequent sets starting from size 1 and increasing its size
  - Prune the candidates including infrequent itemsets



- ⑤  $\mathcal{F}_l(r, \text{min\_sup})$  is the set of frequent sets from  $R$  of size  $l$ .
- ⑥ Given a set of patterns of length  $l$ , the only frequent set candidates of length  $l + 1$  will be those which all subsets are in the frequent sets of length  $l$ .

$$\begin{aligned} C(\mathcal{F}_{l+1}(r)) = & \{X \subseteq R / |X| = l + 1 \wedge \\ & \forall_Y (Y \subseteq X \wedge |Y| = l \Rightarrow Y \in \mathcal{F}_l(r))\} \end{aligned}$$

- ⑦ The computation of association rules can be done iteratively starting with the smallest frequent subsets until no more candidates are obtained

---

**Algorithm:** Apriori ( $R, \text{min\_sup}, \text{min\_conf}$ )

C,CCan,CTemp:set of frequent subsets

RAs:Set of association rules, L:integer

L=1

CCan=Frequent\_sets\_1( $R, \text{fr\_min}$ )RAs= $\emptyset$ **while**  $CCan \neq \emptyset$  **do**

L=L+1

    CTemp=Candidate\_sets(L,R,CCan) =  $C(\mathcal{F}_{l+1}(r))$ 

C=Frequent\_sets(CTemp,min\_sup)

    RAs=RAs  $\cup$  Confidence\_rules(C,min\_conf)

CCan=C

Initial set of itemsets

L=4

ABCD

L=3

ABC

ABD

ACD

BCD

L=2

AB

AC

AD

BC

BD

CD

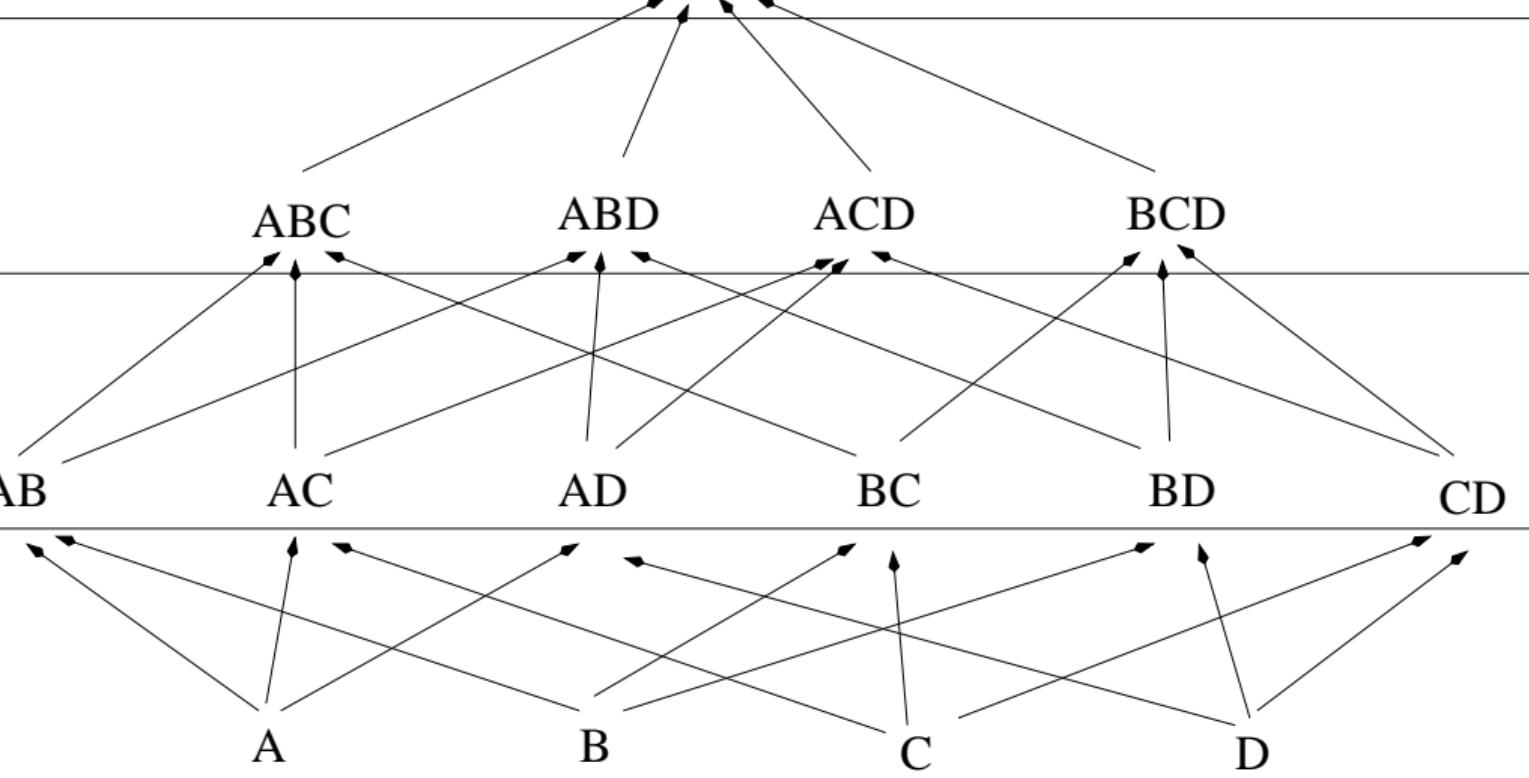
L=1

A

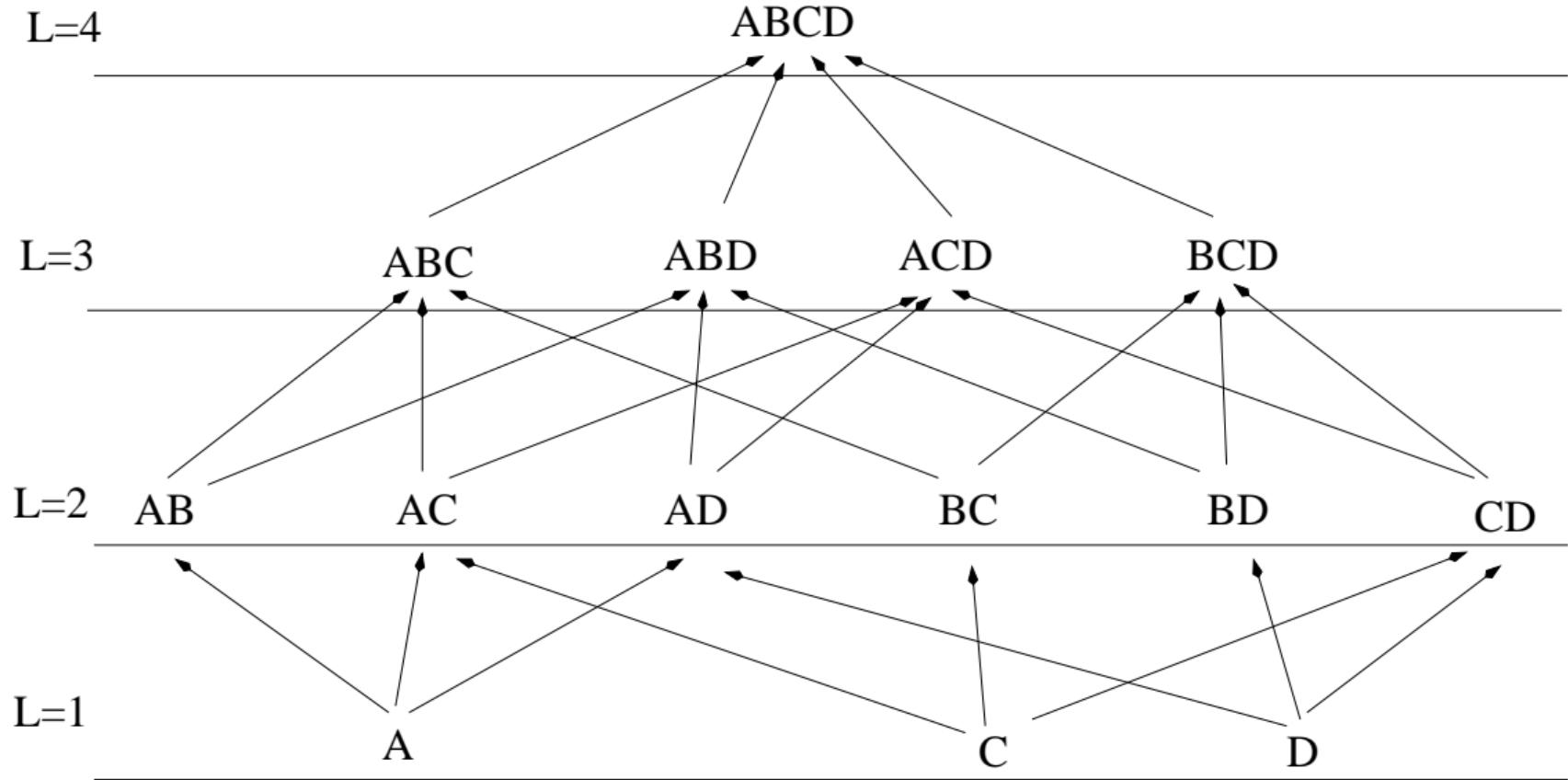
B

C

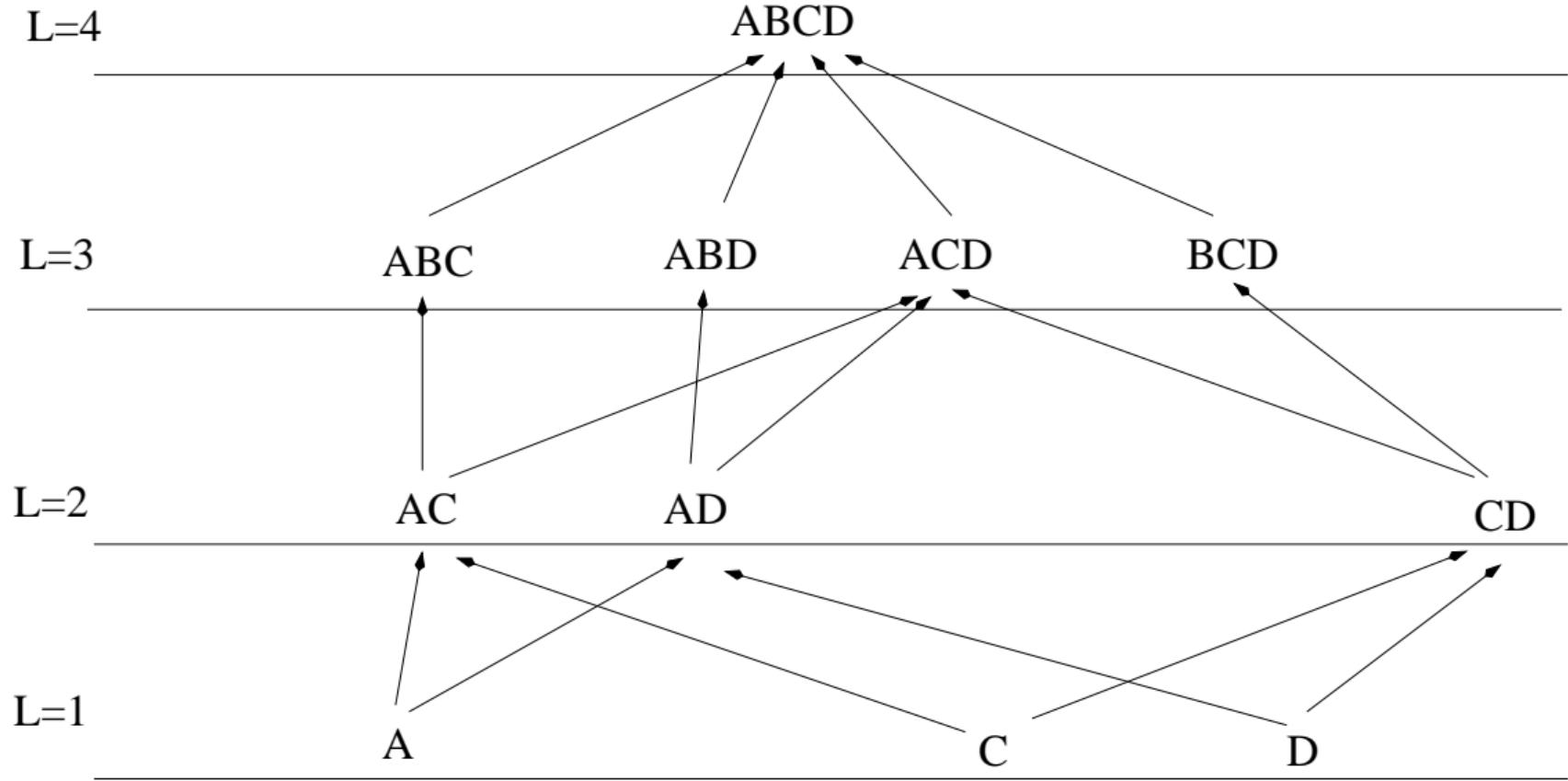
D



Itemsets size=1



Itemsets size=2



Itemsets size=3

L=4

ABCD

L=3

ACD

L=2

AC

AD

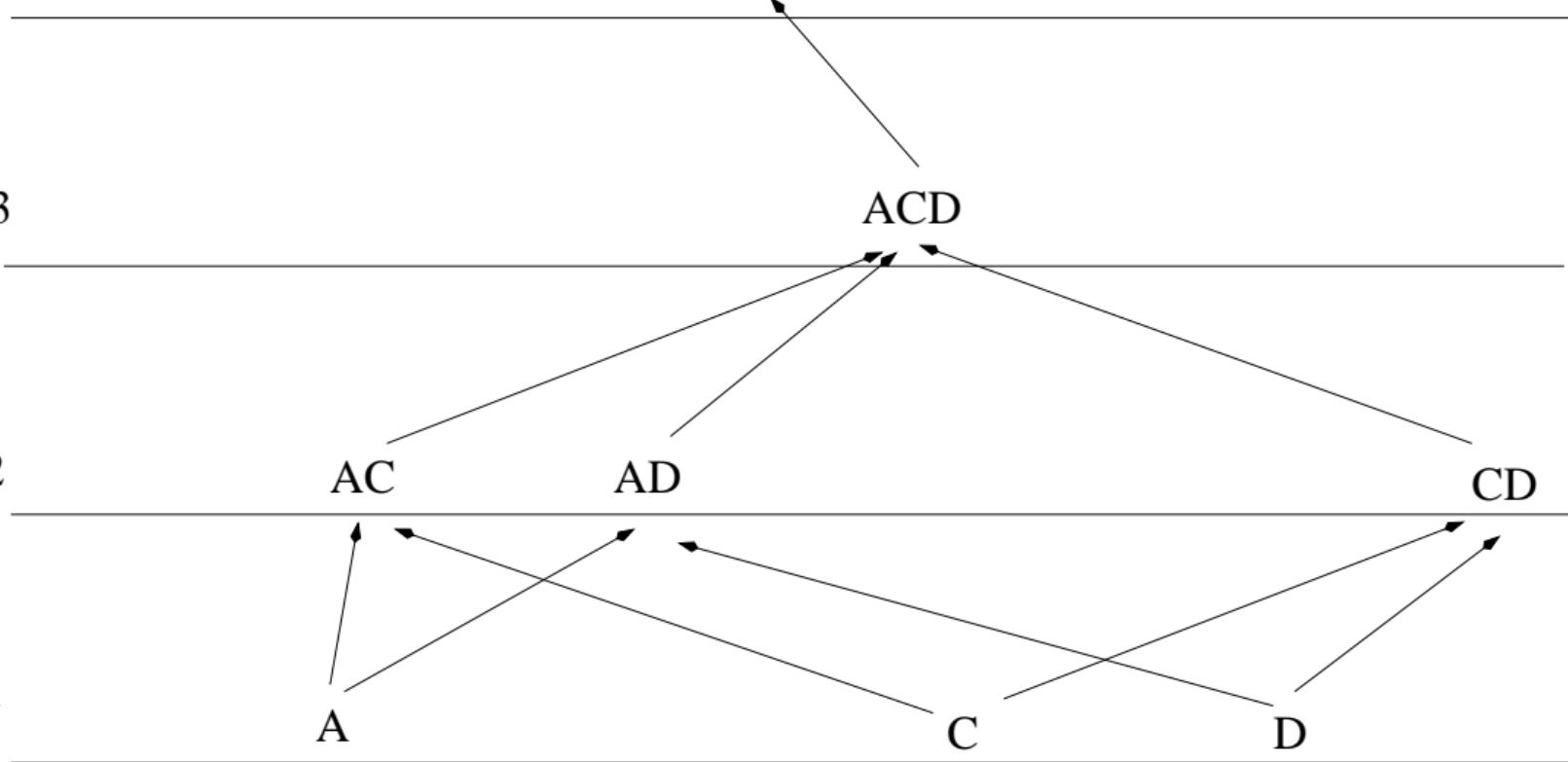
CD

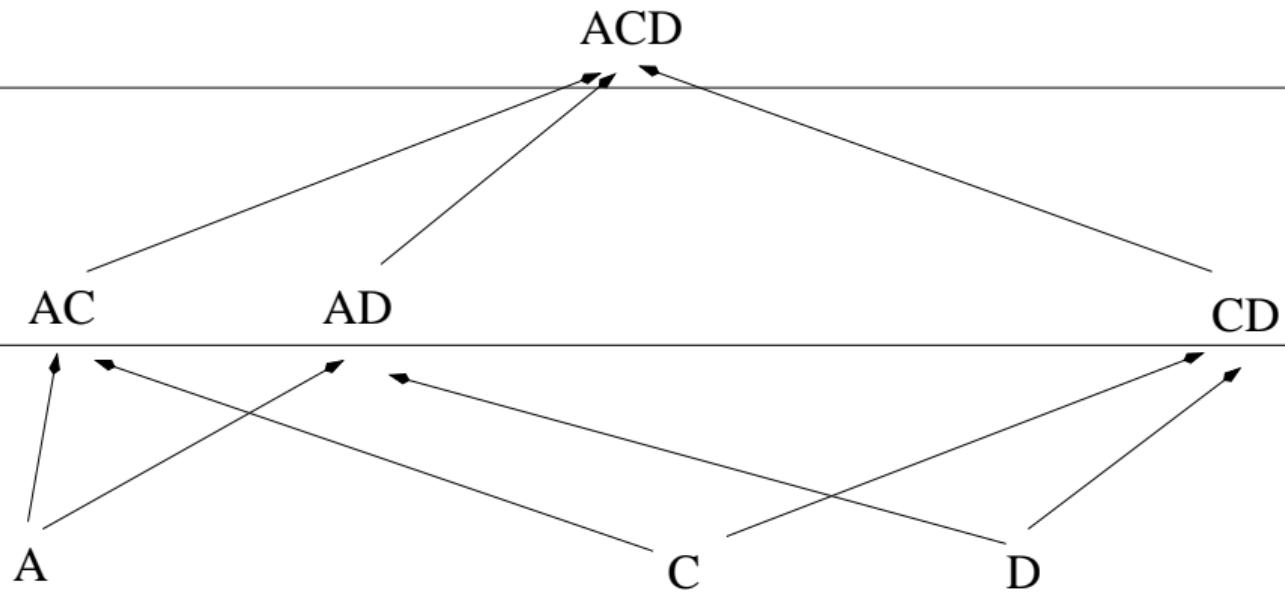
L=1

A

C

D



$L=4$  $L=3$  $L=2$  $L=1$ 

- ④ The specific value used as *min\_sup* has effect on the computational cost of the search
  - If it is too high, only a few patterns will appear (we could miss interesting rare occurrences)
  - If it is too low the computational cost will be too high (too many associations will appear)
- ④ Unfortunately the threshold value can not be known beforehand
- ④ Sometimes multiple minimum supports are needed (different for different groups of items)

- ⑤ Number of different items
  - The more items there are the more space is needed to count its support
- ⑤ Size of the database
  - The algorithm has to perform multiple passes to count the support
- ⑤ Average transaction height
  - Affects the maximum length of frequent itemsets

- Some itemsets are redundant because they have identical support as their supersets
- We could focus only on those itemsets, reducing the number of candidates
  - **Maximal frequent itemsets:** A frequent itemset for which none of its immediate supersets are frequent
  - **Closed frequent itemsets:** A frequent itemset for which none of its immediate supersets have the same support, and its support is larger than *minsupport*

Maximal frequent itemsets  $\subseteq$  Closed frequent itemsets  $\subseteq$  Frequent itemsets

# FP-Growth

---

- ⑤ Despite of pruning and search strategies for Association Rules candidate generation is expensive
- ⑥ Other strategies allow extracting association rules using specialized data structures

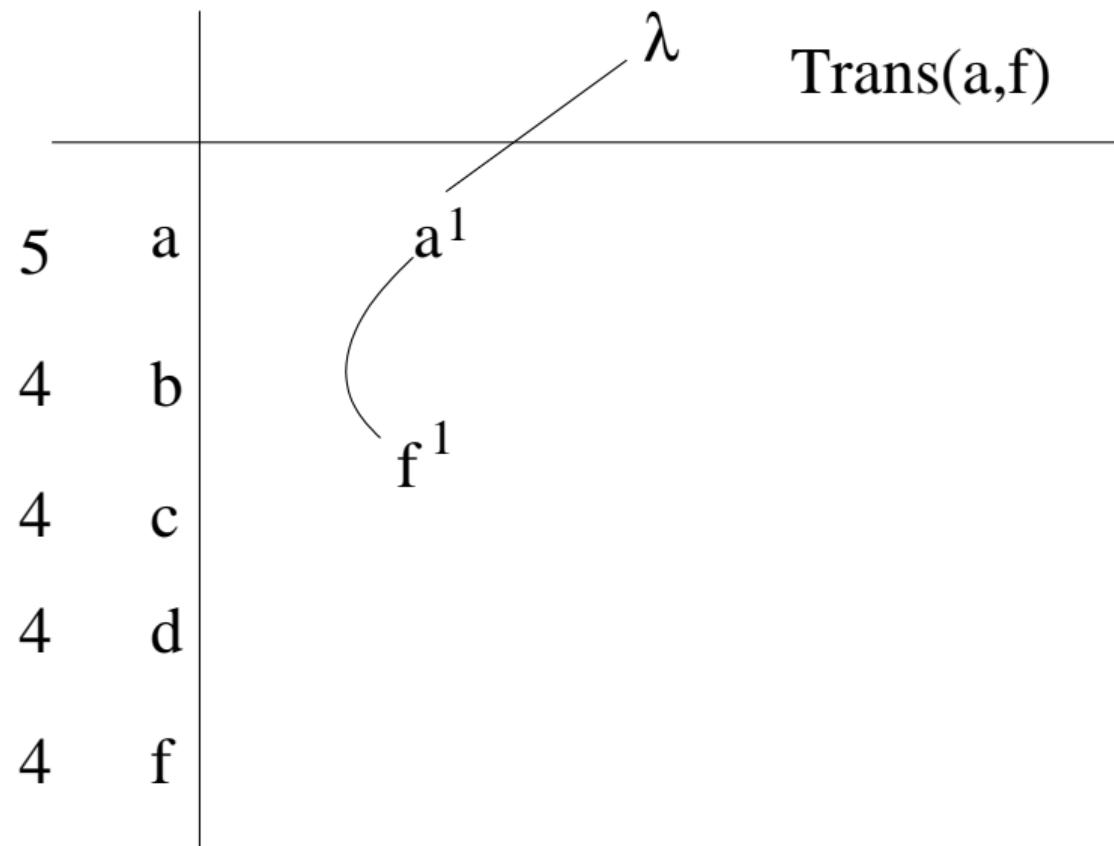
## Han, Pei, Yin, Mao Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach (2004)

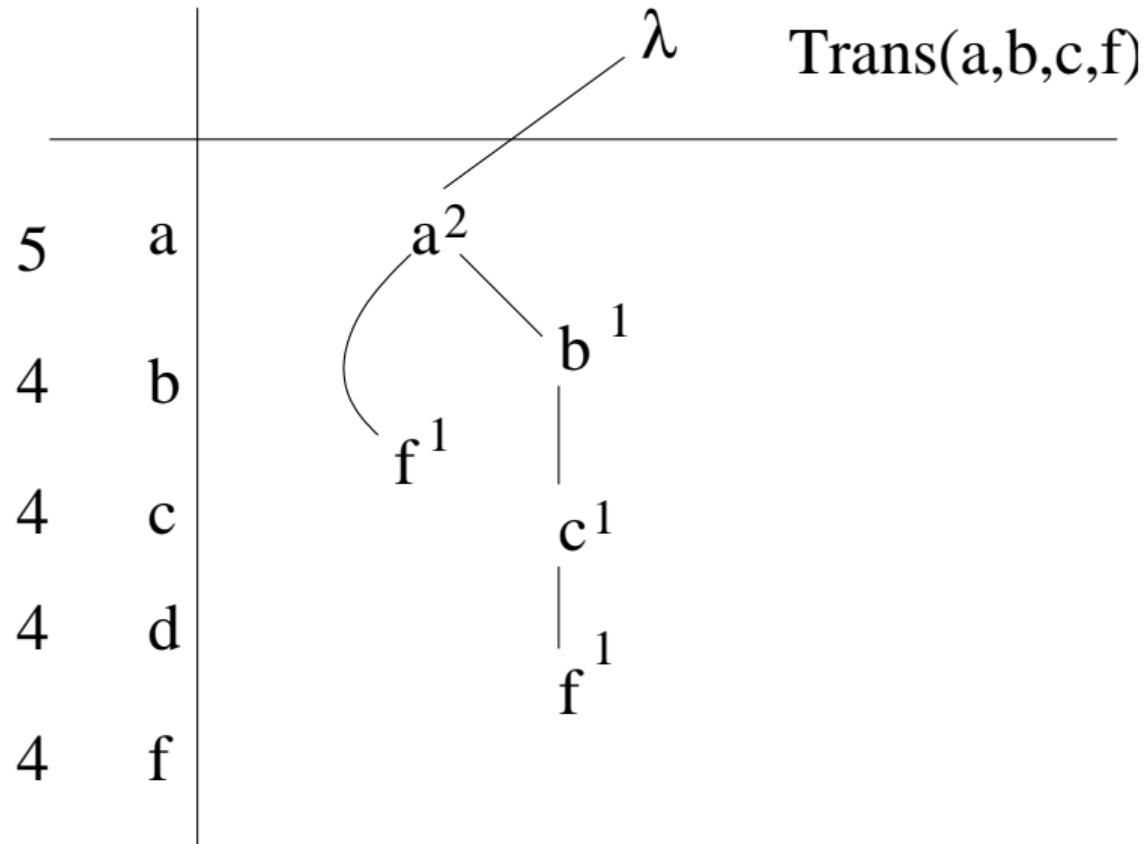
- ④ The problem of the *apriori* approach is that the number of candidates to explore in order to find long patterns can be very large
- ④ This approach tries to obtain patterns from transaction databases without candidate generation
- ④ It is based on a specialized data structure (**FP-Tree**) that summarizes the frequency of the patterns in the DB
- ④ The patterns are explored incrementally adding prefixes without candidate generation

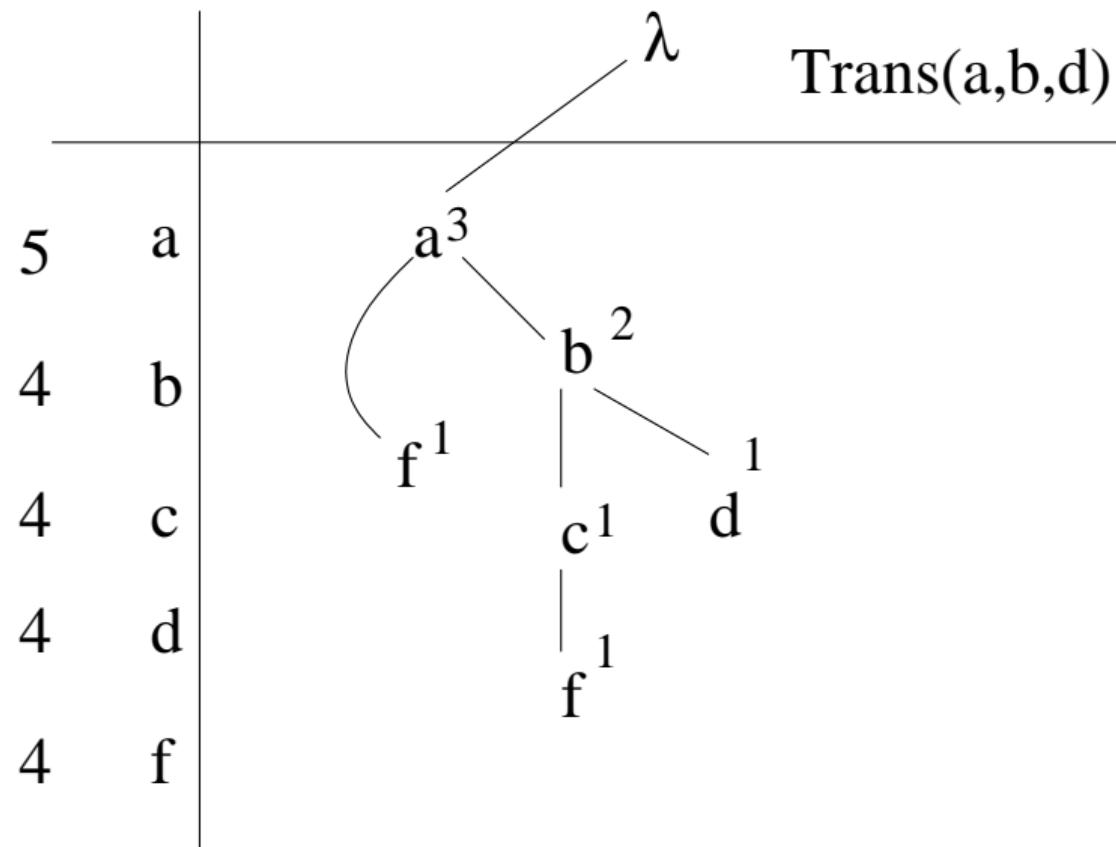
- The goal of this structure is to avoid querying the DB to compute the frequency of patterns
- It assumes that there is an order among the elements of a transaction
- This order allows obtaining common prefixes from the transactions
- The transactions with common prefixes are merged in the structure maintaining the frequency of each subprefix

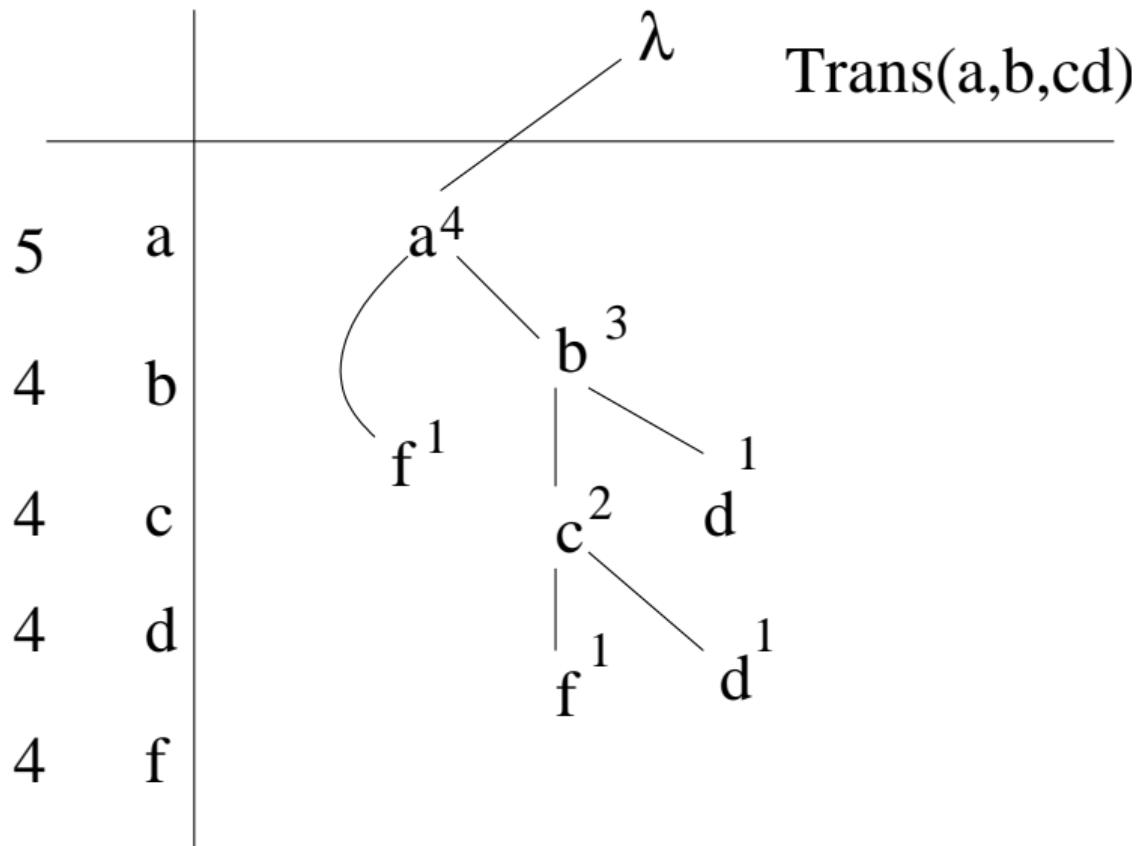
1. Compute the frequency of each individual item in the DB
2. Create the tree root (empty prefix)
3. For each transaction from the DB
  - 3.1 Pick the frequent items
  - 3.2 Order the items by their original frequency
  - 3.3 Iterate for each item, inserting it in the tree
    - If the node has a descendant equal to the actual item increase its frequency
    - Otherwise, a new node is created

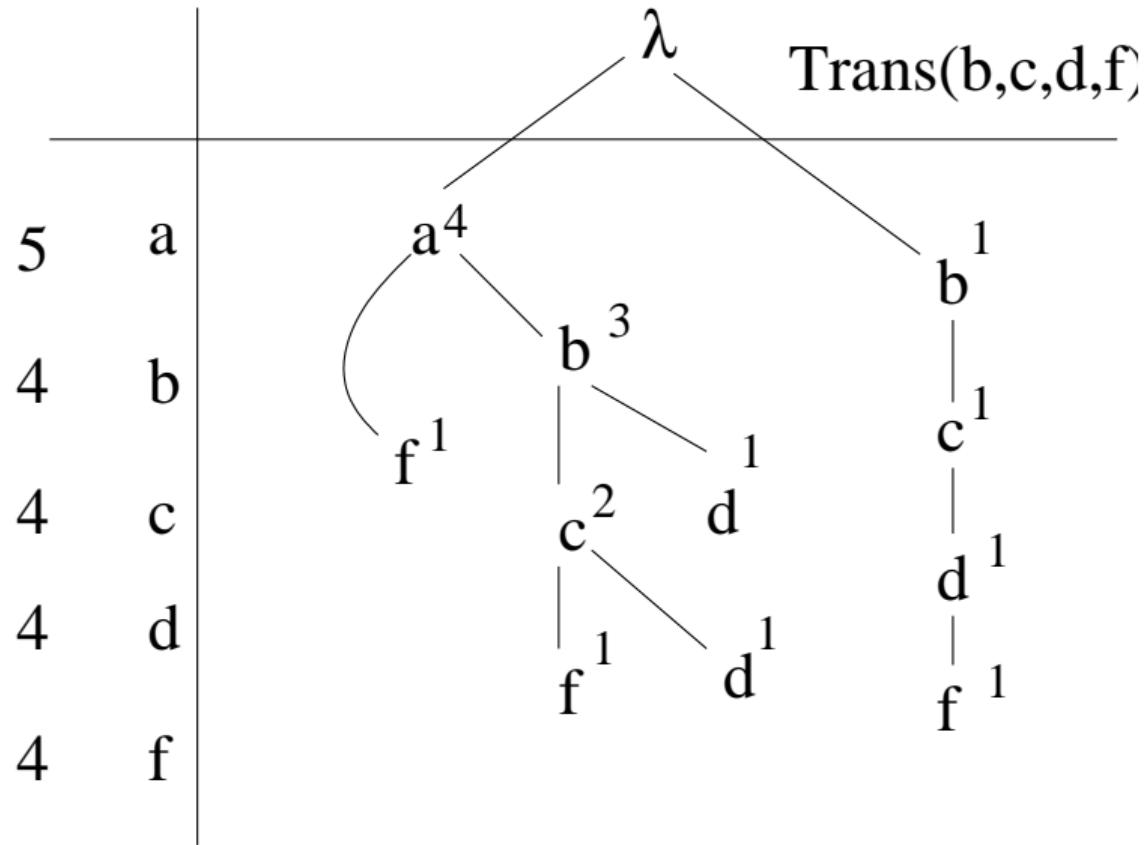
BD	Transactions	Item	frequency	BD	Transactions (fr=4)
1	a, e, f	a	5	1	a, f
2	a, c, b, f, g	b	4	2	a, c, b, f
3	b, a, e, d	c	4	3	a, b, d
4	d, e, a, b, c	d	4	4	a, b, c, d
5	c, d, f, b	e	3	5	b, c, d
6	c, f, a, d	f	4	6	a, c, d, f
		g	1		

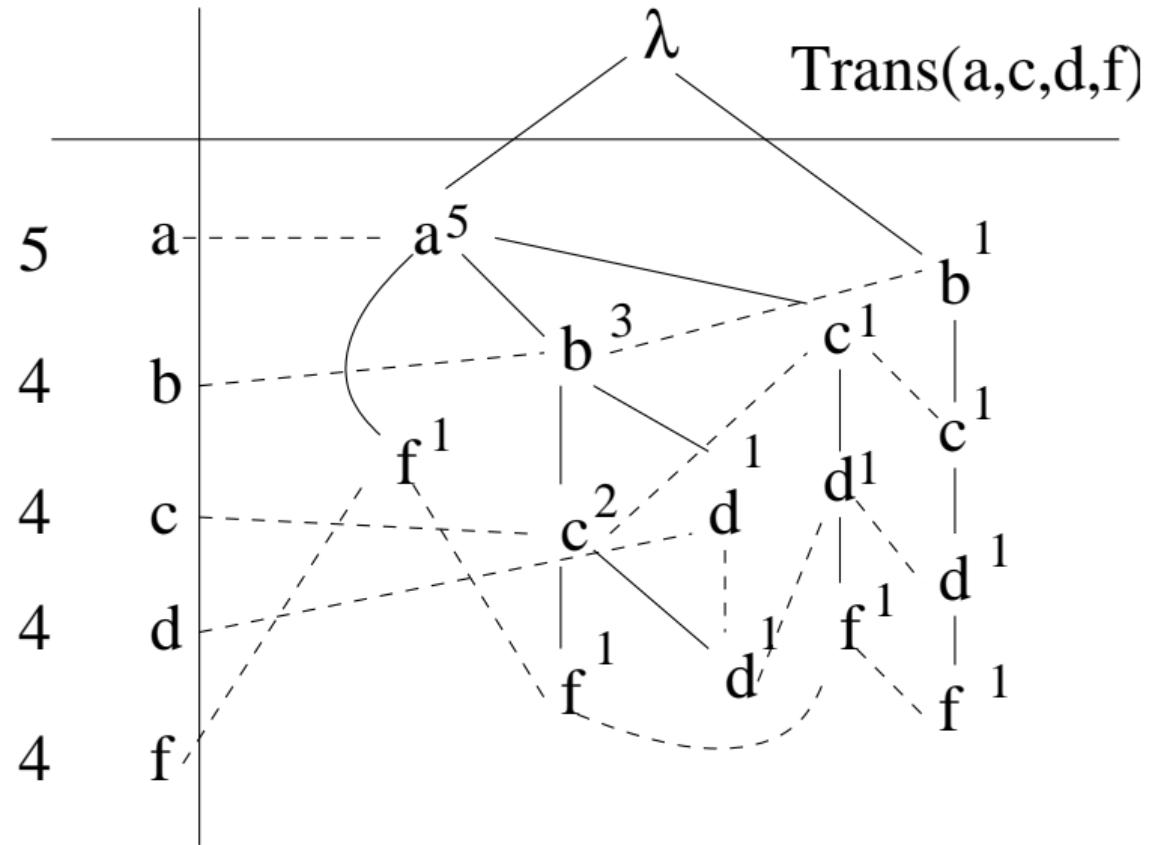






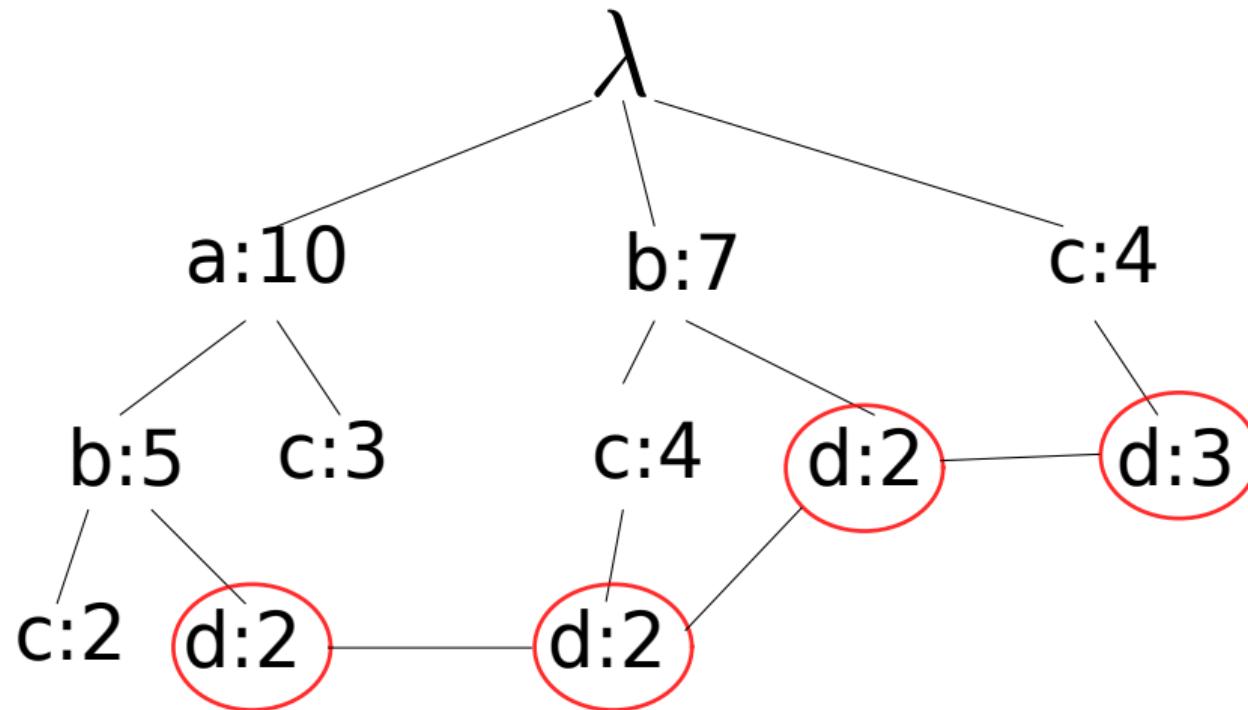




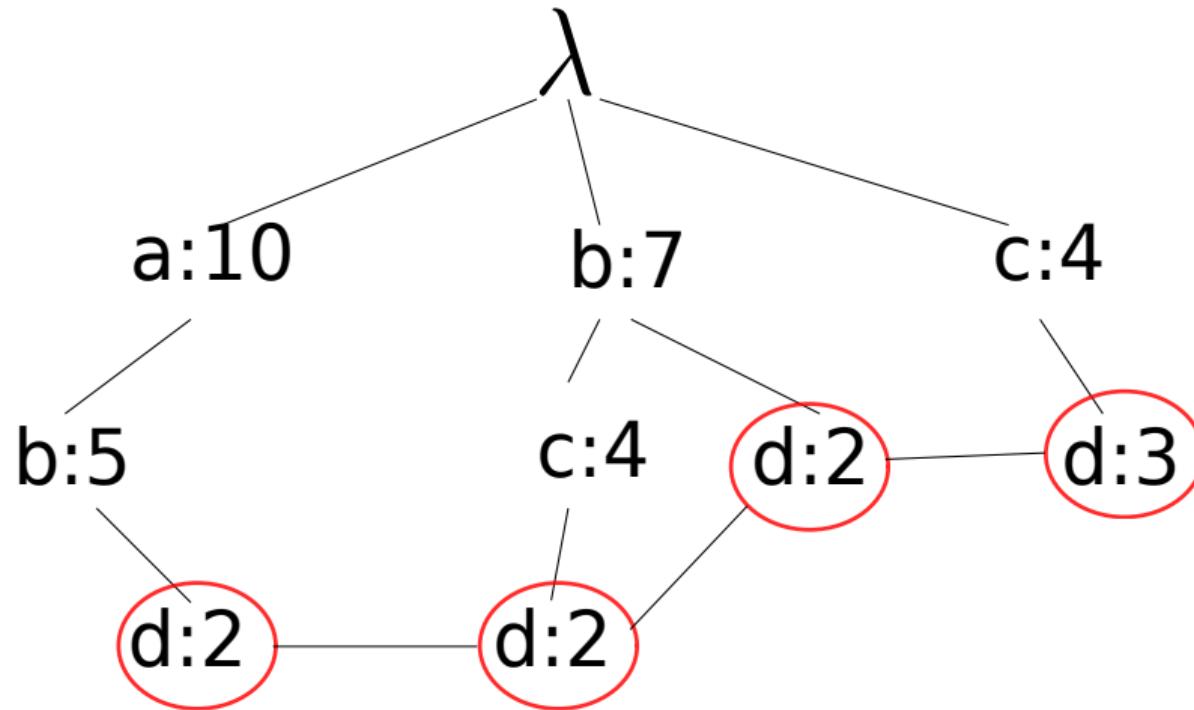


1. Given an item, select all the paths that contain the item (**prefix paths**)
2. Convert all the paths that contain the item into a *conditional* FP-tree:
  - 2.1 Update the counts along the path using the frequency of the selected item
  - 2.2 Truncate the paths removing the nodes for the item
  - 2.3 Eliminate from the paths the items that are no longer frequent (if any)
3. For all the items previous in order that are frequent in the conditional FP-tree
  - 3.1 Count the prefix as frequent
  - 3.2 Recursively find the frequent items for that prefix

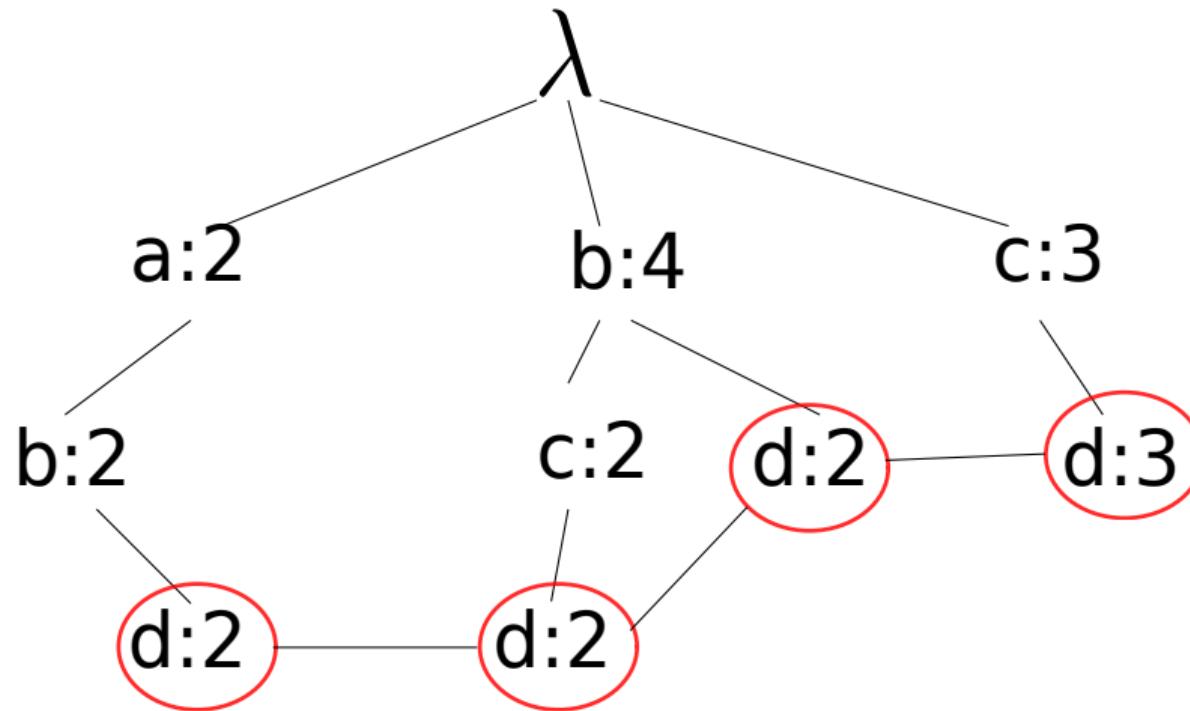
Extract patterns with suffix  $d$



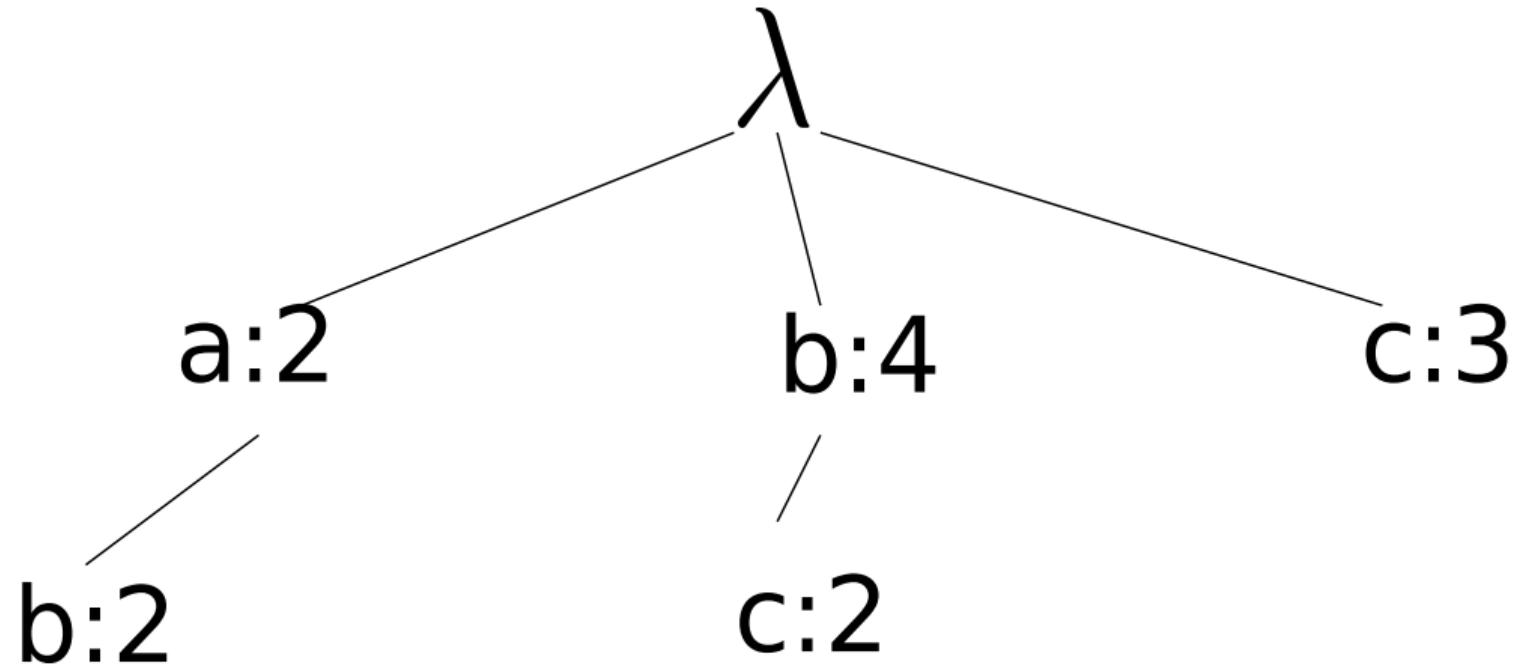
Keep only the paths that contain  $d$  (minsupport = 2)



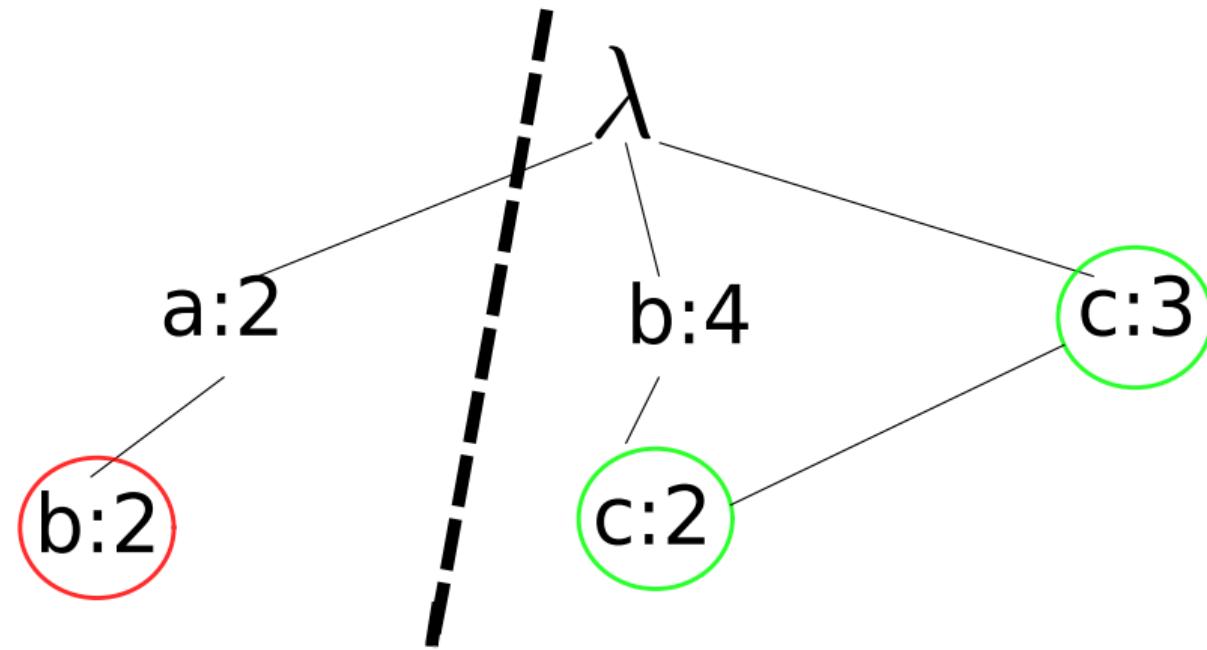
Generate the **conditional FP-tree** for  $d$  (updating the counts)



Prune the path eliminating  $d$  (is no longer needed)



Solve the problem for the predecessors of  $d$ , in this case  $b$  and  $c$  (we are looking if  $bd$  and  $cd$  are frequent)



## Measures of interestingness

---

- Given a rule, its interestingness can be computed from a contingency table:

	$Y$	$\neg Y$	
$X$	$f_{11}$	$f_{10}$	$f_{1+}$
$\neg X$	$f_{01}$	$f_{00}$	$f_{0+}$
	$f_{+1}$	$f_{+0}$	$ r $

- $f_{11}$ ; support of  $X$  and  $Y$
- $f_{10}$ ; support of  $X$  and  $\neg Y$
- $f_{01}$ ; support of  $\neg X$  and  $Y$
- $f_{01}$ ; support of  $\neg X$  and  $\neg Y$

- ⑤ Other measures are based on probabilistic independence and correlation

Lift/Interest factor	$N \times \frac{f_{11}}{f_{1+} \times f_{+1}}$
All Confidence	$\min\left(\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}}\right)$
Max Confidence	$\max\left(\frac{f_{11}}{f_{1+}}, \frac{f_{11}}{f_{+1}}\right)$
Kulczynski	$\frac{1}{2}\left(\frac{f_{11}}{f_{1+}} + \frac{f_{11}}{f_{+1}}\right)$
Cosine Measure	$\frac{f_{11}}{\sqrt{f_{1+}f_{+1}}}$

- Properties that a measure must hold

- If  $A$  and  $B$  are statistically independent then  $M(A, B) = 0$
- $M(A, B)$  increases monotonically with  $P(A, B)$  when  $P(A)$  and  $P(B)$  remain unchanged
- $M(A, B)$  decreases monotonically with  $P(A)$  (or  $P(B)$ ) when  $P(A, B)$  and  $P(B)$  (or  $P(A)$ ) remain unchanged

- Other properties

- Symmetry ( $M(A, B) = M(B, A)$ )
- Invariant to row/column scaling
- Invariant to inversions ( $f_{11} \leftrightarrow f_{00}$ ,  $f_{10} \leftrightarrow f_{01}$ )
- Null addition invariant (not affected if  $f_{00}$  is increased)

This Python Notebook has examples using the Authors dataset for Apriori and FP-Growth

- ⑤ Association Rules Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

## Twitter - City Connections

---

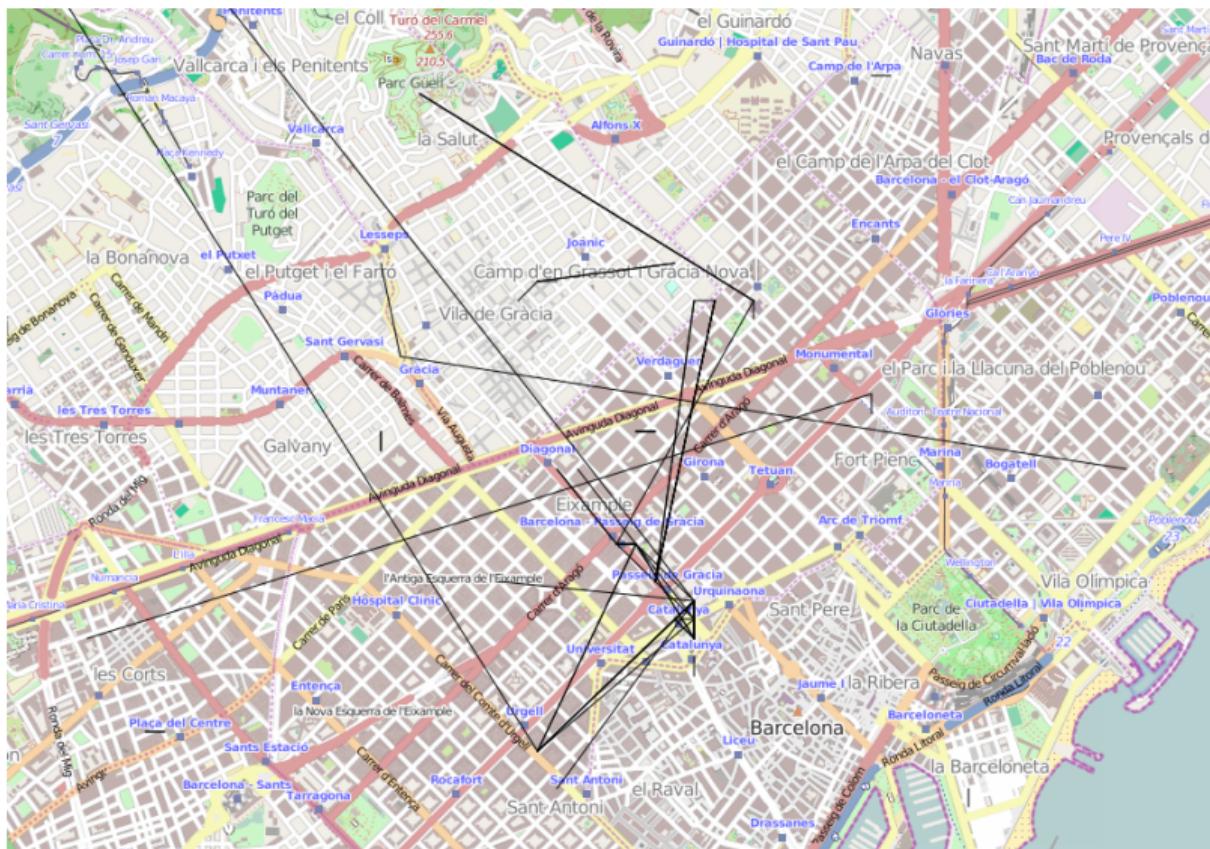
- The dataset consists on tweets with a time stamp and longitude and latitude coordinates
- Data has been collected for several months (2.5 million tweets) in the Barcelona area ( $30Km \times 30Km$ )
- The goal is to perform spatio-temporal analysis of the behavior of people in a geographical area
- We are interested in patterns that show connections among different parts of the city

- ④ One approach is to consider a tweet as an event performed by a user
- ④ The events of a user in a single day can be seen as a transaction (market basket)
- ④ The attributes of a transaction are the time and the position of the user
- ④ To discover city connections can be solved by discovering frequent itemsets

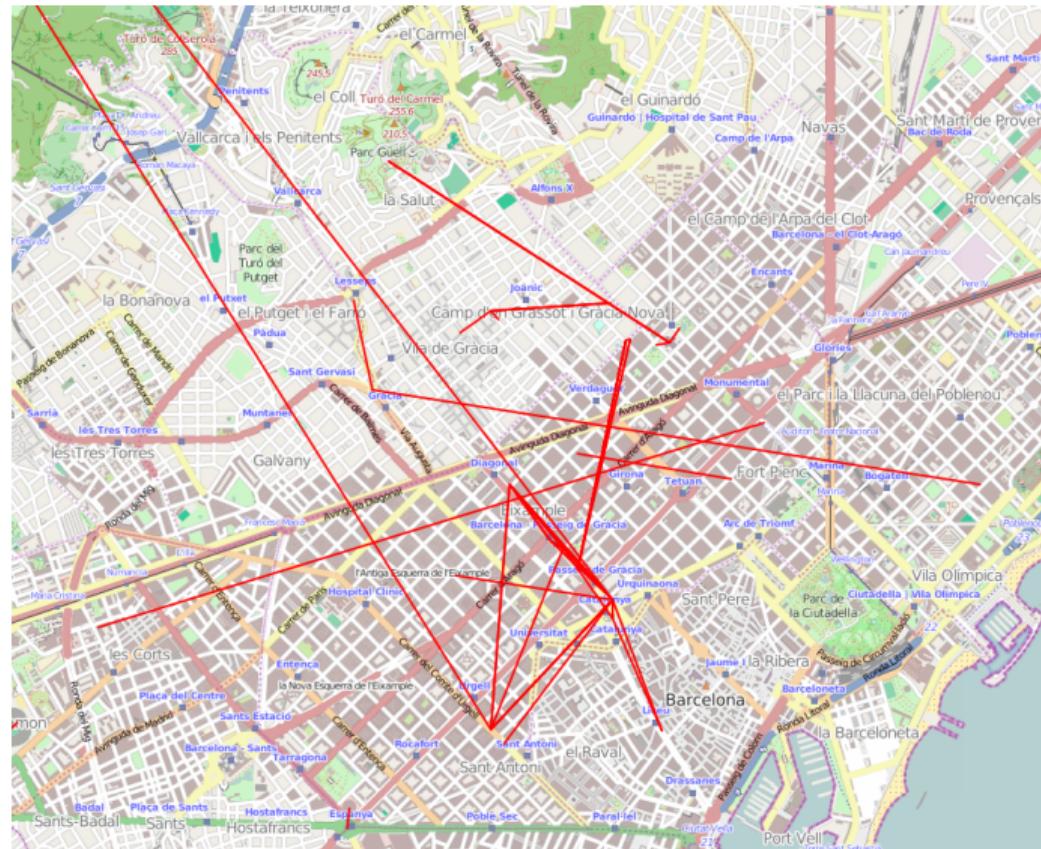
- ④ Considering all possible values for position and time makes impossible to discover frequent patterns
- ④ In order to obtain a suitable representation for frequent itemsets discovery we need to discretize the attributes
- ④ For time:
  - Consider different meaningful hour groups (morning, evening, night, ...)
  - Groups of hours
- ④ For coordinates:
  - Equally spaced grid (granularity?)
  - Clustering (what clustering algorithm?)

- ⑤ We pick to discretize time in intervals of 8 hours (0-7, 8-16, 13-23)
- ⑤ Two possibilities for coordinates:
  - An equally spaced  $300 \times 300$  grid ( $100m \times 100m$ )
  - Clusters obtained by the leader algorithm (radius  $100m$ )
- ⑤ The first option makes that a transaction has a determined number of attributes ( $300 \times 300 \times 3$ ), but not all the possibilities will appear
- ⑤ For second the option the number of attributes will depend on the densities in the coordinates

- Generate the database of transactions
- Discard the transactions that have only values for 1 attribute
- Decide for a value for the support parameters
- Use FP-Grow because of the size of the dataset, and the number of possible attributes
- Only present maximal itemsets to reduce the number of redundant patterns



Twitter - City Connections - results



# Mining of Structures

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Introduction

---

- There are some domains where patterns are more complex
- In these domains instances are related to each other
- Mining these relationships is more interesting than mining instances individually
- For example:
  - Temporal domains
  - Relational databases
  - Structured instances (trees, graphs)
- Usually the methods used in this kind of domains are specific

# Sequences

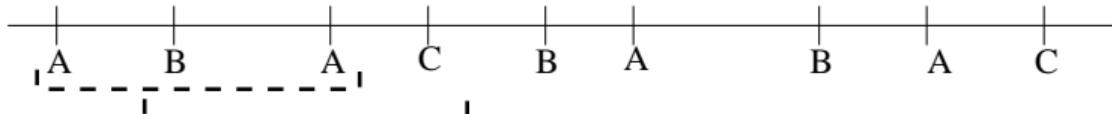
---

## Frequent sequence patterns: Discovery of patterns inside the sequence

- The series does not need to have a temporal relationship (eg.: DNA sequences)
- The values can be continuous, discrete or structured (transaction data)
- Goals:
  - Discovery of subseries that reveal some causality of events
  - Discovery of abnormal/novel patterns (deviation from normal behavior)
  - Discovery of frequent patterns

## Mannila, Toivonen, Verkamo **Discovery of frequent episodes in event sequences** (1997)

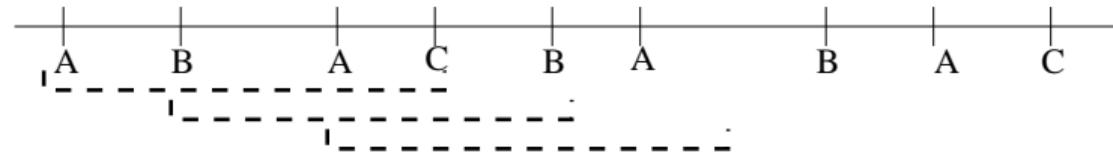
- Qualitative data (events), temporal relationship
- The goal is to discover if close events are causally related
- The result is a set of frequent episodes of different length
- The algorithm discovers the patterns using a sliding window of a fixed size
- The size of the window is increased in successive iterations until no further patterns are discovered
- To reduce time complexity uses the property that frequent patterns have to contain smaller frequent patterns (the same as in association rules)



ABA      BAB  
BAC      ABA  
ACB      BAC  
CBA      ...



ABA (10)  
BAC (4)  
BAB(6)  
...



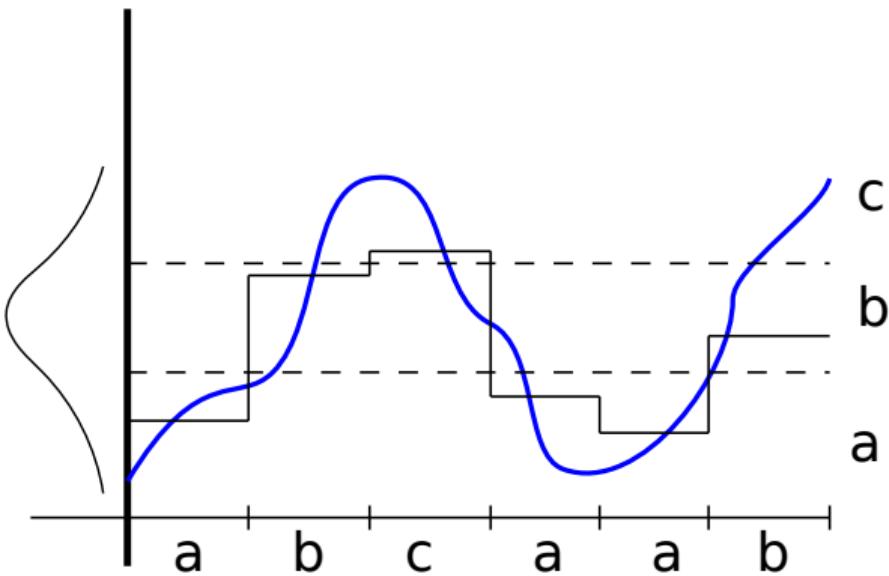
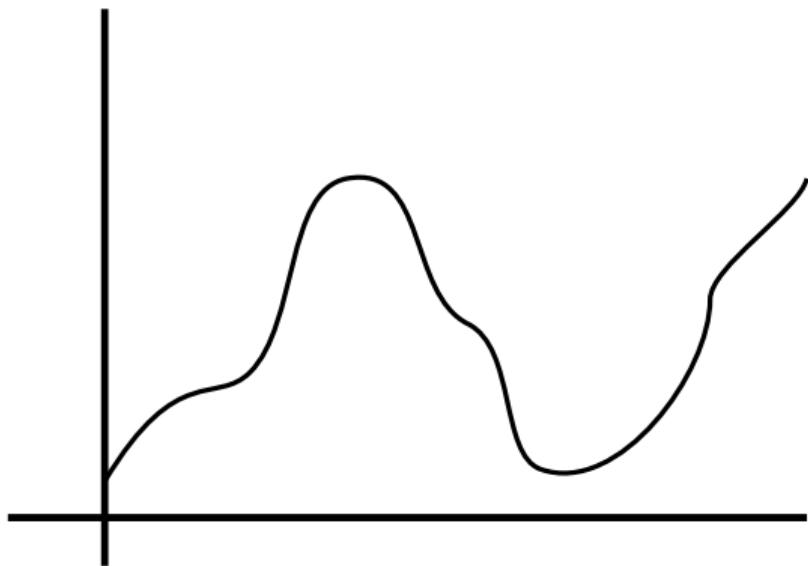
ABAC      BABA  
BACB      ABAC  
ACBA      ...  
CBAB



ABAC (3)  
...

## Keogh, Lonardi, Chiu **Finding Surprising Patterns in a Time Series Database In Linear Time and Space** (2002)

- Quantitative data, continuous time series
- The goal is to discover the most frequent patterns given a window length
- The trivial algorithm is  $O(N^2)$
- To reduce computational complexity the windows are transformed to a simplified representation (Symbolic Aggregate Approximation/SAX)
- Distance over new representation is a lower bound of distance in original space



- ⑤ A window of a specified length is used to segment the sequence (overlapped windows)
- ⑥ The transformed sequence is stored in multiple hash tables using locality sensitive hashing
- ⑦ Search for similar sequences:
  - Sequences appearing in the same or contiguous buckets in different hash tables are compared
  - First, compute the distance in the transformed series (cheap), if it is less than a threshold, then compute the exact distance
  - If sequences are near, store the sequences as candidate frequent motifs
- ⑧ Return the candidates that appear more than a number of times

## Vilo Discovering Frequent Patterns from Strings (1998)

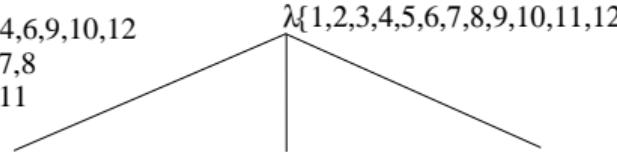
- The sequence is a string of characters (no assumptions about the relationship among the elements of the sequence)
- We look for frequent patterns of any length (complete, with equivalent classes, with wildcards)
- These algorithms use specialized data structures to obtain the patterns within a reasonable computational cost (for example suffix-trees)
- The construction of these structures are sometimes a preprocess (later we extract the patterns) or given a threshold frequency we obtain only the patterns we are looking for

A suffix-trie is obtained from the sequence, a given ( $k$ ) parameter sets the minimum frequency of the patterns. The structure is created with the empty string.

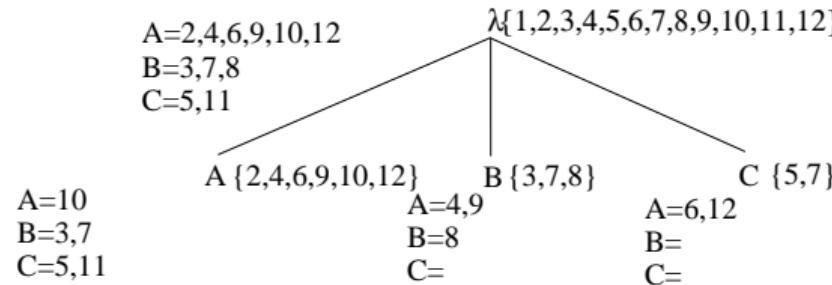
1. For each character that belongs to the alphabet of the sequence
  - 1.1 Compute a list with the next position to the occurrence of the character
  - 1.2 If the character appears more than  $k$  times we create a new descendant in the trie
2. Recursive call for each descendant created

1 2 3 4 5 6 7 8 9 10 11 12  
ABACABBAACAC\$ K=2

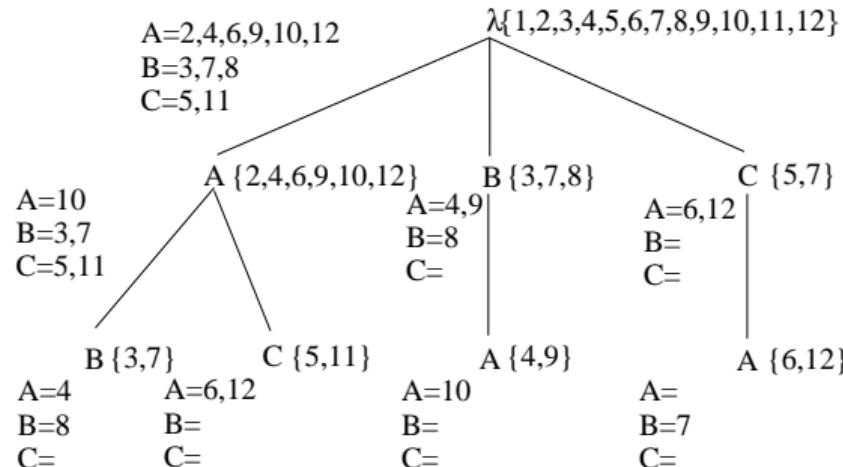
A=2,4,6,9,10,12  
B=3,7,8  
C=5,11



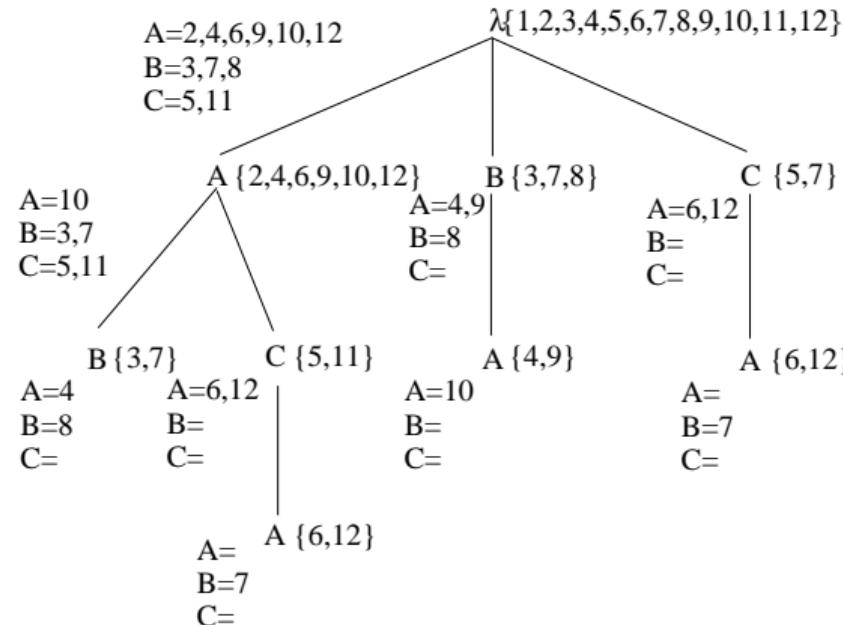
1 2 3 4 5 6 7 8 9 10 11 12  
ABACABBAACAS\$ K=2



1 2 3 4 5 6 7 8 9 10 11 12  
 ABACABBAACAS K=2



1 2 3 4 5 6 7 8 9 10 11 12  
 ABACABBAACAC\$    K=2    → {A,B,C,AB,AC,BA,CA,ACA}



- Sometimes the events in the sequence are sets of elements
- The definition of sequence has to be adapted:
  - Given an ordered list of elements:

$$s = \langle e_1 e_2 e_3 \dots \rangle$$

- Each element contains a collection of items
- $$e_i = \{i_1, i_2, \dots, i_k\}$$
- Also, the definition of subsequence:
    - A sequence  $\langle a_1, a_2, \dots, a_n \rangle$  is contained in another sequence  $\langle b_1, b_2, \dots, b_m \rangle$  ( $m \geq n$ ) if exists integers  $i_1 < i_2 < \dots < i_n$  such that  $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, \dots, a_n \subseteq b_{in}$

Agrawal, Srikant, **Mining Sequential Patterns** (1995)

Srikant, Agrawal, **Mining Sequential Patterns: Generalizations and Improvements** (1996)

- ④ An apriori approach to the mining of sequential patterns
- ④ A similar monotonic property exists among a sequence and its subsequences:
  - Given a sequential pattern  $s$  all its subsequences are also sequential patterns
  - Given a sequential pattern  $s$  if  $s'$  is a subsequence then  $\text{support}(s) \leq \text{support}(s')$
- ④ This means that generating all the frequent sequences for a database can be done using a sequential approach

- Each iteration the sequences of length  $k + 1$  are generated by merging the frequent sequences of length  $k$
- For the generation of candidates all items in each transaction are listed alphabetically
- Each iteration the database is scanned to test for the candidates and compute their support
- Sequences not frequent are pruned

## ⑤ Case 1:

$$(ab)(ac)ab + (ab)(ac)ac \Rightarrow (ab)(ac)a(bc), (ab)(ac)abc, (ab)(ac)acb$$

## ⑥ Case 2:

$$(ab)(ac)(ab) + (ab)(ac)(ac) \Rightarrow (ab)(ac)(abc)$$

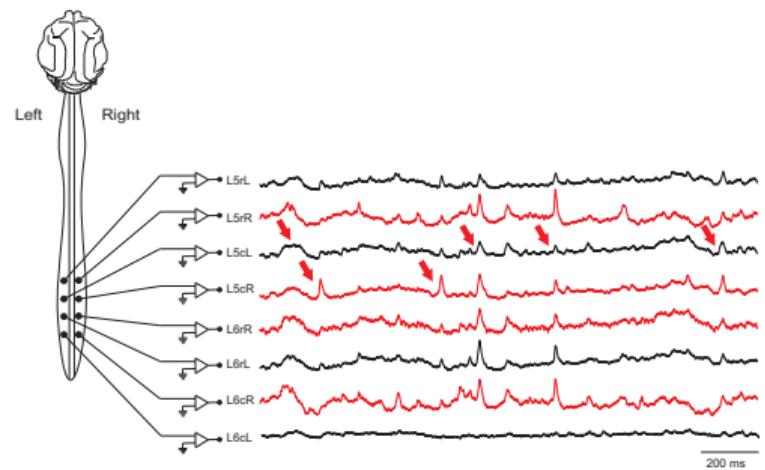
## ⑦ Case 3:

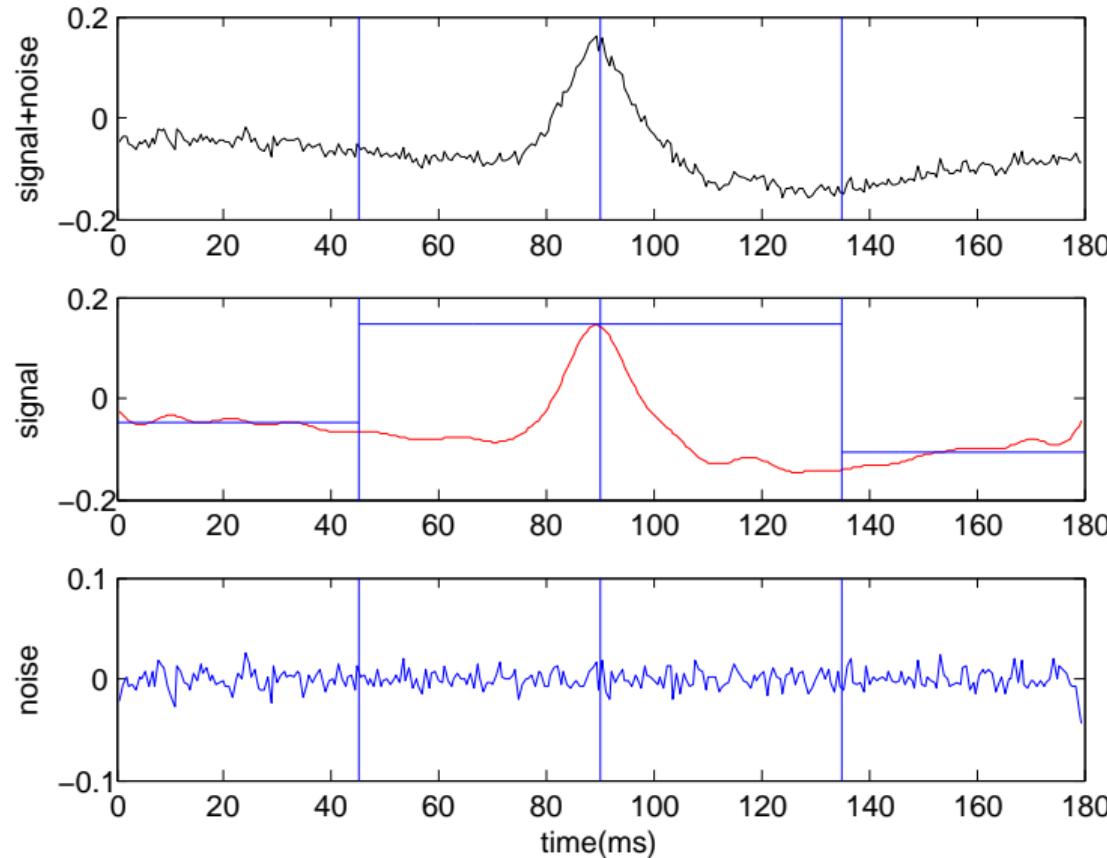
$$(ab)(abc) + (ab)(ab)a \Rightarrow (ab)(abc)a$$

# Application

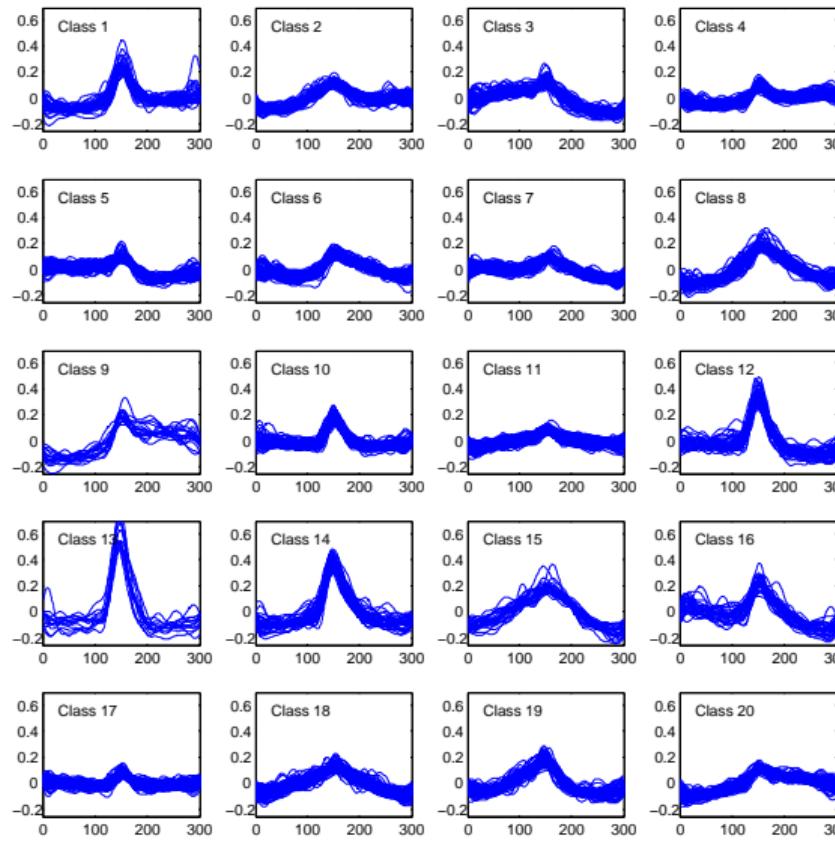
---

- Recorded signals over the lumbar vertebrae of an anesthetized cat (from L4 to L7)
- Different experimental conditions
- Study of the peaks generated spontaneously by groups of neurons
- Study of the synchronizations of the peaks in the different recording points
- Study of the sequential pattern behavior

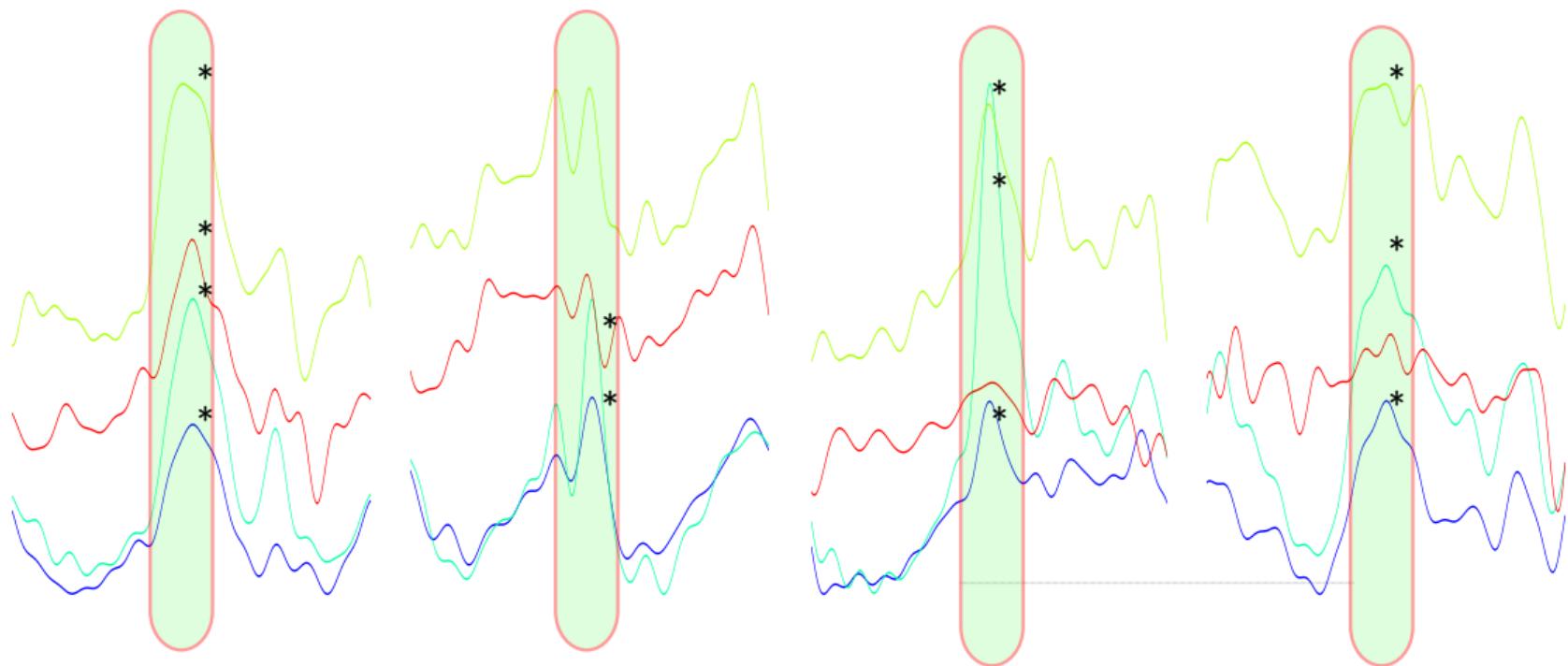




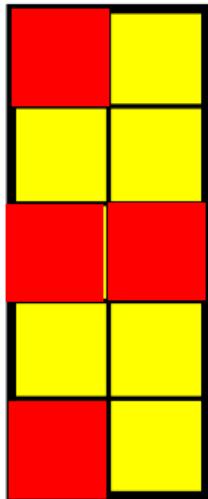
- ① Goal: Determine a structure for the peaks
- ② Process
  - Identify the peaks in the signals (peak finding algorithm)
  - The peaks extracted from each measuring point will be the first dataset
  - Clean/Transform the data to visualize the structure
  - Cluster the datasets to obtain patterns of peaks (k-means  $\Rightarrow$  determine the number of clusters)
- ③ Results: A tentative dictionary of peaks shapes



- ⑤ Goal: Find patterns in the synchronizations of the peaks
- ⑤ Process
  - Determine the synchronizations in the signal (synchronization algorithm)
  - Generate a database of transactions from the synchronizations
  - Apply a frequent transactions algorithm to the transactions database
- ⑤ Results: A set of frequent sequences of synchronization

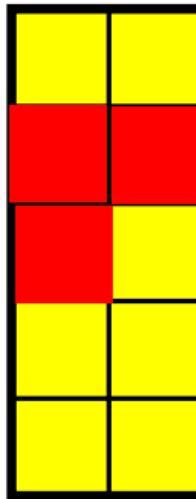


L R



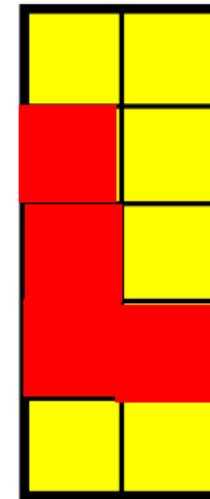
t

L R



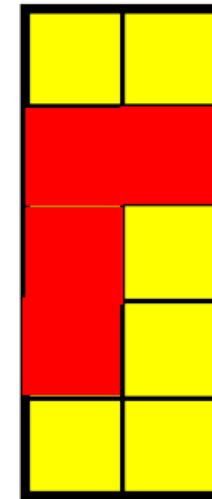
t+1

L R



t+2

L R



t+3

# Graph mining

---

- ⑤ There is a lot of information that has a relational structure
- ⑥ Methods and models used for unstructured data are not expressive enough
- ⑦ Sometimes structure can be flattened, but lots of interesting information is lost
  - Relational database ⇒ unique merged table
  - Attributes representing relations ⇒ inapplicable attributes
  - Graph data ⇒ strings based on graph traversal algorithms
  - Documents ⇒ bag of words

- ① All these types of data have in common that can be represented using graphs and trees
- ② Algorithms that use these data structures as input are needed
- ③ Historically we can find different approaches to the discovery of patterns in graphs/trees:
  - Inductive logic programming: Structure is represented using logic formulas
  - Graph algorithms
    - Classic algorithms for detecting dense subgraphs (cliques)
    - Graph isomorphism algorithms

- Most of the problems used to discover structures in graphs are NP-Hard
  - Graph partitioning (Not for bi partitioning)
  - Graph isomorphism
- Other approaches for mining graphs
- Two different problems:
  - Mining large graphs (only one structure) ⇒ Partitioning, dense subgraphs
  - Mining sets of graphs ⇒ common substructures

- Some information can be described as a large graph (several instances connected by different types of relations)
  - For example: Social networks, Web pages,
- We are interested in discovering interesting substructures by:
  - Dividing the graph in subgraphs (k-way partitioning, node clustering)
  - Extracting dense substructures

- Not always a complete partitioning of the graph is necessary
- Dense subgraphs can represent interesting behaviors
- Different types of (pseudo)dense subgraphs:
  - Clique: All nodes are connected
  - Quasi-clique: Almost all nodes are connected (minimum density or minimum degree)
  - K-core: Every node connects to at least k nodes
  - K-plex: Each node is missing no more than k-1 edges
  - ...
- Different goals: Minimum size, all or the best ranked, overlapping or not

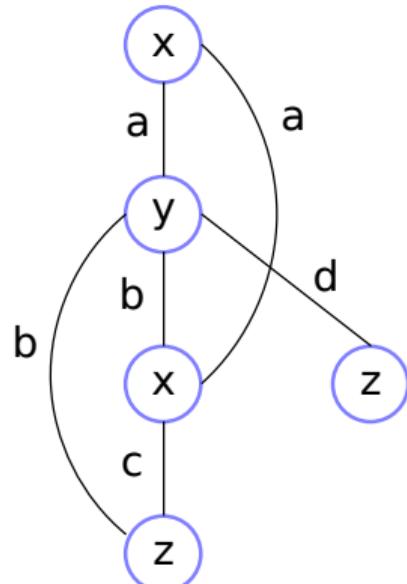
- Some information can be described as a collection graph
  - For example: XML documents, chemical molecules
- We can use two approaches:
  - Cluster the graphs for common patterns and summarization
  - Find frequent substructures

- ④ We look for frequent graphs/trees in a database
- ④ Usually the structures are transformed to some kind of canonical representation (adjacency matrix, tree traversal)
- ④ This representation gives a unique code for each different graph
- ④ We can discover patterns in this code related to the patterns that appear in the original graphs
- ④ Some approaches use the same properties used in association rules increasing the size of the patterns until no pattern is found

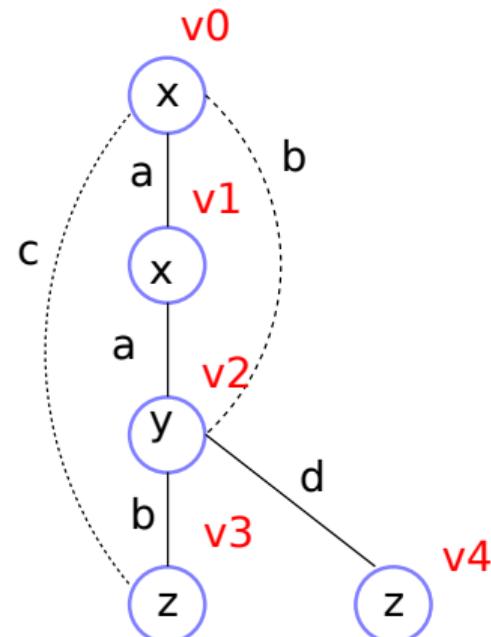
Yan, Han **gSpan: Graph-Based Substructure Pattern Mining** Proceedings of the IEEE International Conference on Data Mining 2002

- Ⓐ The graphs have labels in their edges and their vertices
- Ⓐ A canonical representation is used to reduce the cost to compute graph isomorphism
- Ⓐ This representation is based on the tree obtained by the depth first search of the graph, and a lexicographical order among labels
- Ⓐ This representation transforms a graph into a string that contains the labels of the graph
- Ⓐ With this representation we can have an ordering over all the graphs that can be used to explore all possible subgraphs

- ⑤ We assume an initial vertex ( $v_0$ ), and an order among vertices ( $i = 0 \dots n$ ), many DFS traversals of a graph can be obtained
- ⑤ For a DFS traversal of a graph we define the forward edges (those with  $i < j$ , the DFS tree of the graph), and the backward edges (those with  $i > j$ )
- ⑤ We define a linear order among edges, given  $e_1 = (i_1, j_1)$  and  $e_2 = (i_2, j_2)$ 
  - if  $i_1 = i_2$  and  $j_1 < j_2$  then  $e_1 \prec_T e_2$ ; if  $i_1 < j_1$  and  $j_1 = i_2$  then  $e_1 \prec_T e_2$ ; if  $e_1 \prec_T e_2$  and  $e_2 \prec_T e_3$  then  $e_1 \prec_T e_3$
- ⑤ Given a DFS tree for a graph and edge sequence can be defined based on  $\prec_T$ , this is the DFS code of the graph
- ⑤ Given the DFS codes for a graph the linear order, and a linear order for its labels  $\prec_L$ , a linear order among codes can be defined
- ⑤ The *Minimum DFS code* is the **canonical label** for G



DFS



DFS Code

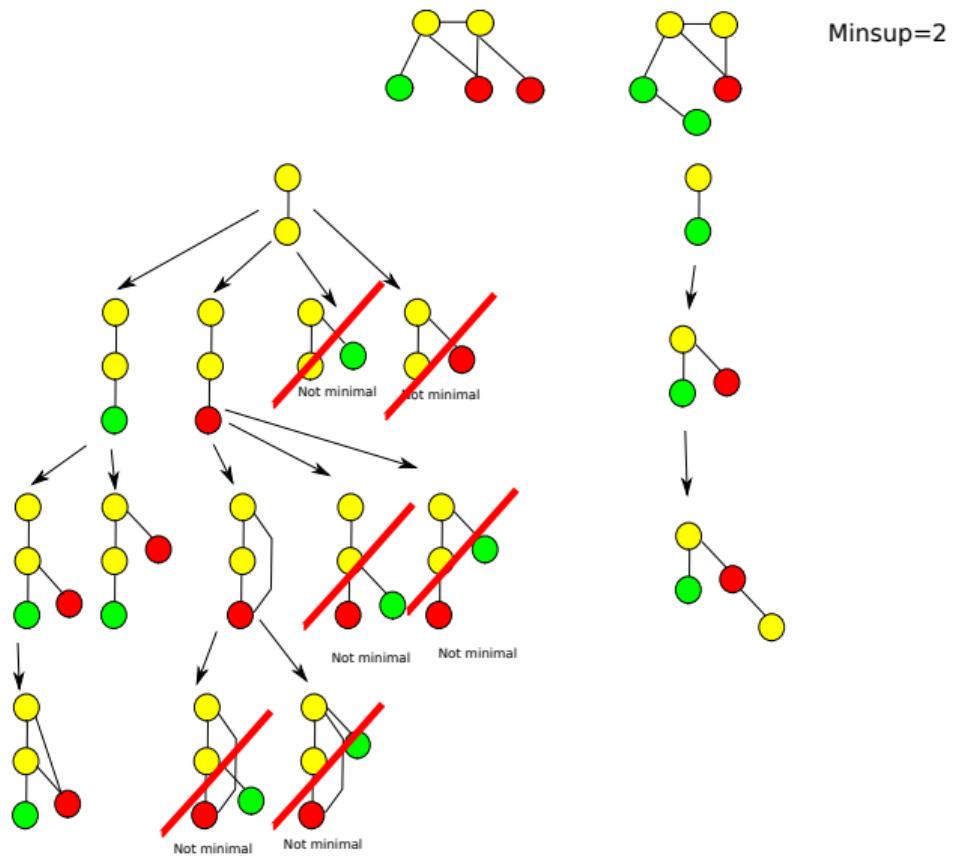
0: (0,1,X,a,X)  
 1: (1,2,X,a,Y)  
 2: (2,0,Y,b,X)  
 3: (2,3,Y,b,Z)  
 4: (3,0,Z,c,X)  
 5: (2,4,Y,d,Z)

**Initialization ( $D, MinSup$ )**

1. sort labels of the vertices and edges in  $D$  by frequency
2. remove infrequent vertices and edges
3.  $S_0$ =code of all frequent graphs with single edge
4. sort  $S_0$  in DFS lexicographic order
5.  $S = S_0$
6. for each code  $s$  in  $S_0$ 
  - 6.1  $gSpan(s,D,MinSup,S)$
  - 6.2  $D = D - s$
  - 6.3 if  $|D| < MinSup$  then return

 **$gSpan(s,D,MinSup,S)$** 

1. if  $s \neq mincode(s)$  then return
2. insert  $s$  into  $S$
3.  $C = \emptyset$
4. scan  $D$ 
  - o find every edge  $e$  such that  $s$  can be right-most extended to frequent  $s * e$
  - o insert  $s * e$  into  $C$ ;
5. sort  $C$  in DFS lexicographic order
6. for each  $s * e$  in  $C$  do
  - o  $gSpan(s * e,D,MinSup,S)$



This Python Notebook has examples of dense subgraphs discovery and community discovery using geolocation information from Twitter for London, Paris and Barcelona

- ⑤ Dense Subgraphs Notebook ([click here](#) to go to the url)

If you have downloaded the code from the repository you will be able to play with the notebooks (run jupyter notebook to open the notebooks)

# UDL: Introduction to Unsupervised DL

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Motivation

---

- Deep learning as the other methods of machine learning has also a bias for supervised methods
- Deep learning relies on huge datasets, but there is a limit to how many labels we can obtain efficiently
- Unsupervised methods can benefit of all the data that we can not label and make an impact on supervised tasks
  - We could learn the inherent probability distribution of the data (generative model) as a representation for supervised tasks
  - We can solve unsupervised tasks with data that need semantic understanding of the problem for building representations

## Geoffrey Hinton

(in his 2014 AMA on Reddit)

*The brain has about  $10^{14}$  synapses and we only live for about  $10^9$  seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get  $10^5$  dimensions of constraint per second.*

# How Much Information is the Machine Given during Learning?

- ▶ “Pure” Reinforcement Learning (**cherry**)
  - ▶ The machine predicts a scalar reward given once in a while.
  - ▶ **A few bits for some samples**
- ▶ Supervised Learning (**icing**)
  - ▶ The machine predicts a category or a few numbers for each input
  - ▶ Predicting human-supplied data
  - ▶ **10→10,000 bits per sample**
- ▶ Self-Supervised Learning (**cake génoise**)
  - ▶ The machine predicts any part of its input for any observed part.
  - ▶ Predicts future frames in videos
  - ▶ **Millions of bits per sample**



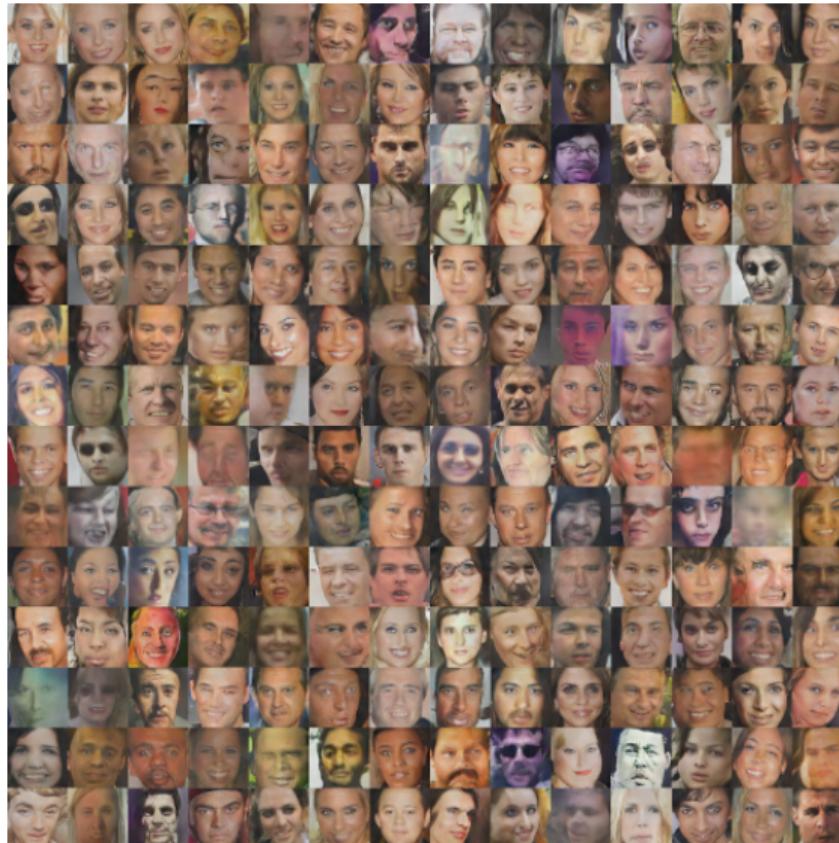
- ⑤ Generation of new data from generative models (probability distribution)
- ⑤ Generation using conditioning (bias the generation with a prior or transfer from one domain to another)
  - Generating speech with different characteristics, generating images from text, image to image translation...
- ⑤ Information compression from latent representations
- ⑤ Improve supervised tasks with data embeddings generated from un/self-supervised pre-training

# Evolution of UDL

---

8 7 3 3 7 0 0  
9 9 1 7 1 4 4  
2 0 8 3 8 2 9  
6 3 8 7 0 6 1  
8 8 9 8 9 1 0  
4 3 4 8 2 8 1









- ⑤ Open AI's Generative Pre-Trained Transformers (GPT-1, 2, 3)

<https://app.inferkit.com/demo>

*While I was teaching my morning class a bird entered the classroom and saw me.*

*He started flying around in the front.*

*I asked one of my students to open the door, thinking the bird would fly out and this would be the last thing they'd see me do that day.*

*I opened the door and the bird flew up into the air and sat on a table.*

*I opened the blinds and the bird went into the classroom and sat under a table.*

*I walked back to my room and I went up to my students, sitting in a circle, writing what we did today on the board.*

*I stopped at the bird.*

- ④ Word2vec (2013), GloVe (2014), fastText (1026)
- ④ ElMO, Em-beddings from Language Models, (2018) 5 NLP tasks
- ④ BERT, Bidirectional Encoder Representations from Transformers (2018) 11 NLP tasks
- ④ BERT based models are now king for text classification, machine translation, question answering, NER, morphological tagging, syntactical analysis. . .

# UDL: Autoregressive Models

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Motivation

---

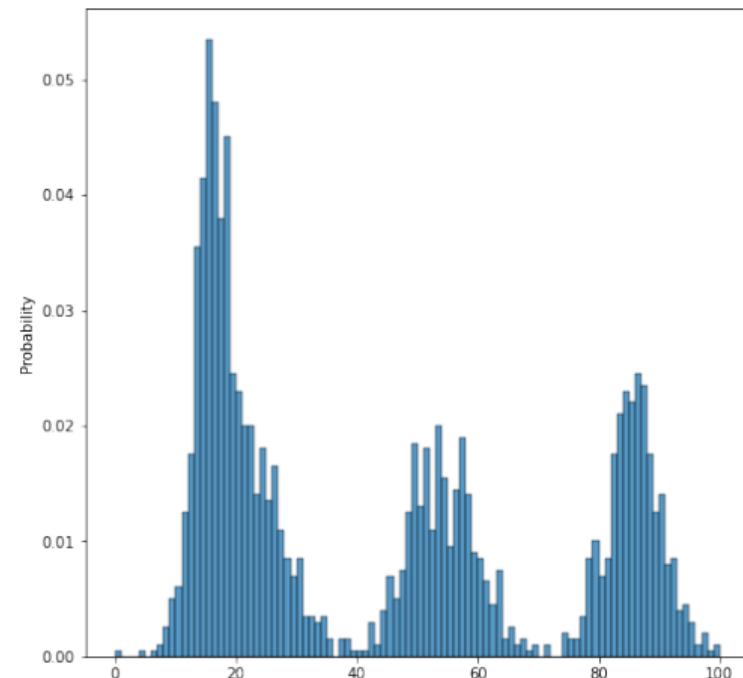
- We are interested in models able to learn the probability distribution  $p(x)$  of a problem from samples
- From this probability distribution we can
  - Generate new samples: Images, video, speech, text...
  - Obtain representations able to compress the data (codes)
  - Compute the probability of new data for applications (eg: anomaly detection)
- **Likelihood-based models** allow estimating  $p(x)$  from samples  $x_1, \dots, x_n$  given a parametrized distribution function
- From these models we can compute the probability of new data and generate samples

- ⑤ For real world applications we want
  - To be able to model complex and high dimensional data (eg. images, video, text)
  - To be efficient in terms of model training and representation
  - To be able to express a large variety of problems (expressive), and to be able to generalize from examples
  - To be able to generate good samples efficiently
  - To have a good compression ratio of the data (representation size/amount of data)
- ⑥ We will start with discrete probability distributions

# Histograms

---

- A simple approach to learn the pdf of a dataset is to compute a histogram of the data
- Given a sample  $x_1, \dots, x_n$  assuming a finite set of values for the variables
  1. Count how many times each value appears in the sample
  2. Divide the counts by the number of samples
- Use the histogram as  $p(x)$



- ④ Inference can be computed as simply looking up the frequency corresponding to the new sample
- ④ Sampling requires:
  1. compute the cumulative distribution function (CDF)
  2. sample from a uniform distribution  $u \sim U(0, 1)$
  3. find the smallest value  $i$  in the CDF that  $u \leq CDF_i$

- ① **Curse of dimensionality:** The size of the histogram (number of parameters) grows exponentially with the number of dimensions
- ① Samples are limited, so only a tiny fraction of the probability space is estimated (a lot of zeros!)
- ① There is no generalization, each sample only affects one parameter
- ① Even in low dimensions we can overfit easily
- ① **Solution:** Compute the probability distribution function using function approximation  $p_\theta(x)$

# Parameterized Distributions

---

- ④ Our goal is to approximate  $p_{data}(x)$  using a parameterized function  $p_\theta(x)$
- ④ We need to learn the parameters  $\theta$  so  $p_\theta(x) \approx p_{data}(x)$
- ④ Basically we need to solve a problem over the parameters that:

$$\arg \min_{\theta} loss(\theta, x_1, \dots, x_n)$$

- ④ There are different approaches that we can use, but we have to take in account
  - We need to work with large datasets
  - We need to obtain a function that gets as close as possible to the data distribution
  - We need the function to generalize (we have a limited sample, high dimensionality)

- ④ Maximum likelihood parameter estimation guarantees that for a family of functions expressive enough and given enough data, it results in the parameters that generate the data
- ④ We are optimizing the loss

$$\arg \min_{\theta} \text{loss}(\theta, x_1, \dots, x_n) = -\frac{1}{n} \sum_{i=1}^n \log p_{\theta}(x_i)$$

- ④ This is equivalent to minimizing the Kulback-Leibler divergence (KL) between the empirical data distribution and the model

$$KL(\hat{p}_{data} || p_{\theta}) = \mathbb{E}_{x \sim \hat{p}_{data}} [-\log p_{\theta}(x)] - H(\hat{p}_{data})$$

- Maximum likelihood estimation can be solved using stochastic gradient descent (SGD)
- SDG minimizes the expectations, for a  $f$  a differentiable function with parameters  $\theta$  it solves

$$\arg \min_{\theta} \mathbb{E}[f(\theta)]$$

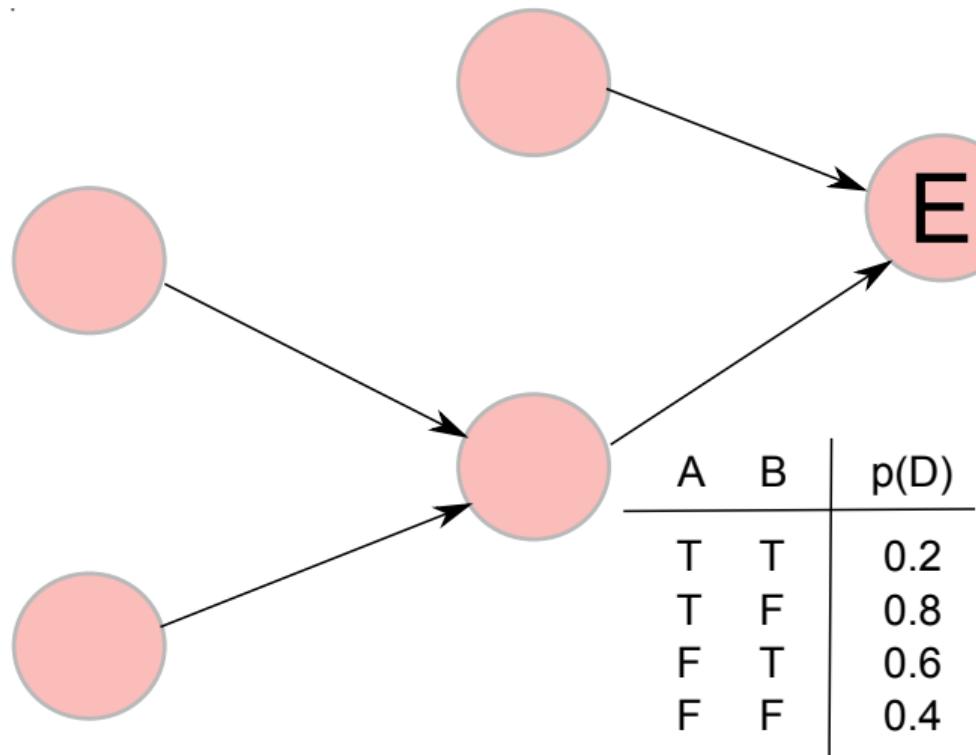
- For Maximum Likelihood we have:

$$\arg \min_{\theta} \mathbb{E}_{x \sim \hat{p}_{data}} [-\log p_{\theta}(x)]$$

- The advantage of SDG is that it works with large datasets, and we can use neural networks as estimation functions

- ④ We are going to use neural networks as estimators, but not any architecture would do
- ④ We need the network to be a proper probability distribution, basically
  - The sum of the probabilities for all the data sums to 1
  - The probability for any example must be  $\geq 0$
- ④ We also need that  $\log p_\theta(x)$  is easy to compute and differentiable respect to  $\theta$

- Bayesian networks are probabilistic graphical models that factorize a joint probability distribution function using a Directed Acyclic Graph over the variables
- Each node of the graph models the joint distribution of a subset of the variables of the problem, the variable conditioned to its parents in the graph  $p(A|B, C, \dots)$
- We can model this conditioned distribution as a neural network,
  - The input of the network are the parents of the variable
  - The output is the variable that is conditioned
  - The network outputs the probability distribution of the variable



~



# Autoregressive Models

---

- Given a Bayesian network structure to transform the conditional distribution to neural network will obtain a tractable log likelihood and gradients

$$\log p_{\theta}(x) = \sum_{i=1}^d \log p_{\theta}(x_i | \text{parents}(x))$$

- Given a joint probability distribution the product rule allows expressing the probability function as the product of conditional probabilities

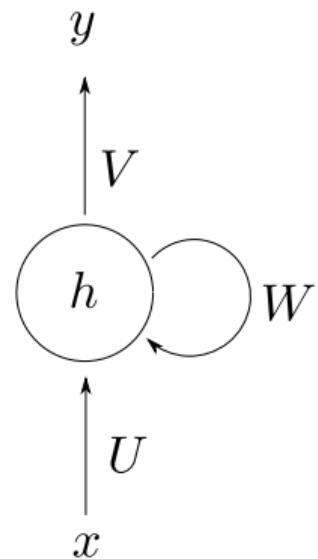
$$\log p(x) = \sum_{i=1}^d \log p(x_i | x_{i-1}, \dots, x_1)$$

- This is called an **autoregressive model**
- We can represent this bayesian network using neural networks for the conditional probabilities obtaining a tractable maximum likelihood problem

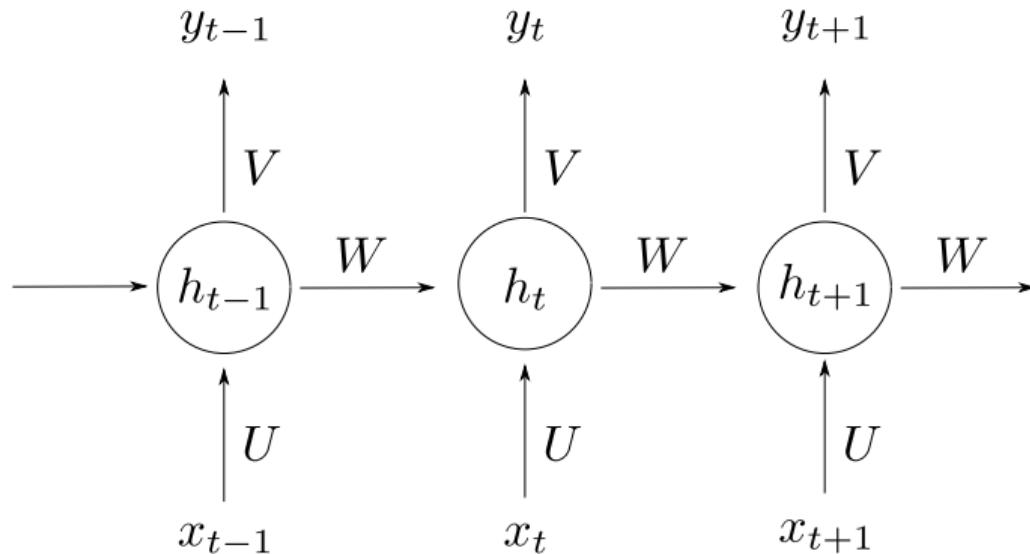
- The network that we obtain reduces the number of parameters from  $o(2^d)$  to  $o(d)$
- Having separate networks for each variable limits the generalization power
- A better solution is to share parameters among the different networks
- Two approaches:
  - Recurrent Neural Networks
  - Masking

- Recurrent Neural Networks are feed forward networks for sequential data
- The network has recurrent connections that connect each element in the sequence
- Each step the computation uses the previous step

$$h^{(t+1)} = f(h^{(t)}, x_{t+1}; \theta) = f(f(h^{(t-1)}, x_t; \theta), x_{t+1}; \theta) = \dots$$



- Parameters are shared among the steps reducing the number of parameters and helping to generalization



- Recurrent network for generating text from characters
- Trained from sequences to predict a character conditioned to the elements on the sequence
- Able to generate different kinds of text from literature to code
- The main handicap is that RNN training can not be parallelized each step depend on the previous one
- Recurrent networks have problem with vanishing gradients because of the length of the sequences and the sharing of parameters
- Better RNN units: LSTMs and GRUs

⑤ Sampling sequences consists on:

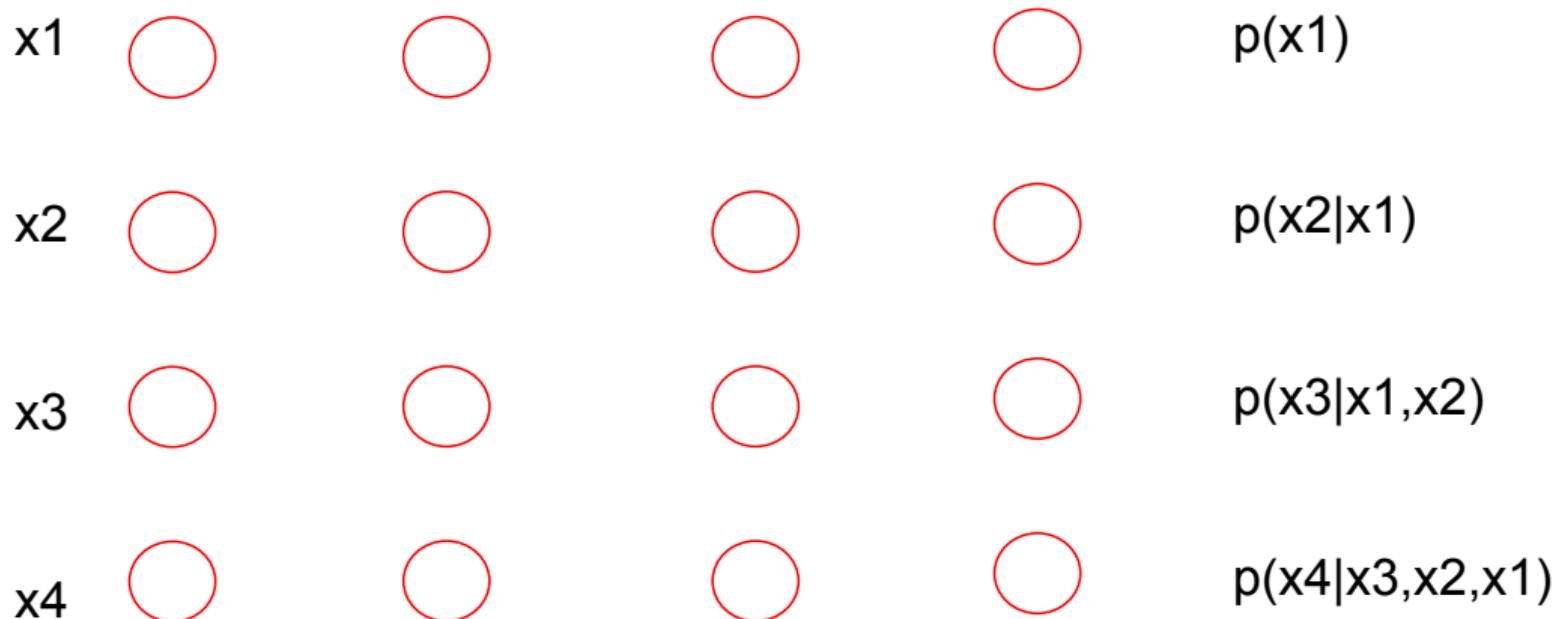
1. Condition the first input to some sequence (even random) to obtain a new character
2. Discard the first element of the sequence and put the generated character as last element
3. Feed the new sequence to the network for obtaining the next one
4. Repeat from 2 until you reach a specific length

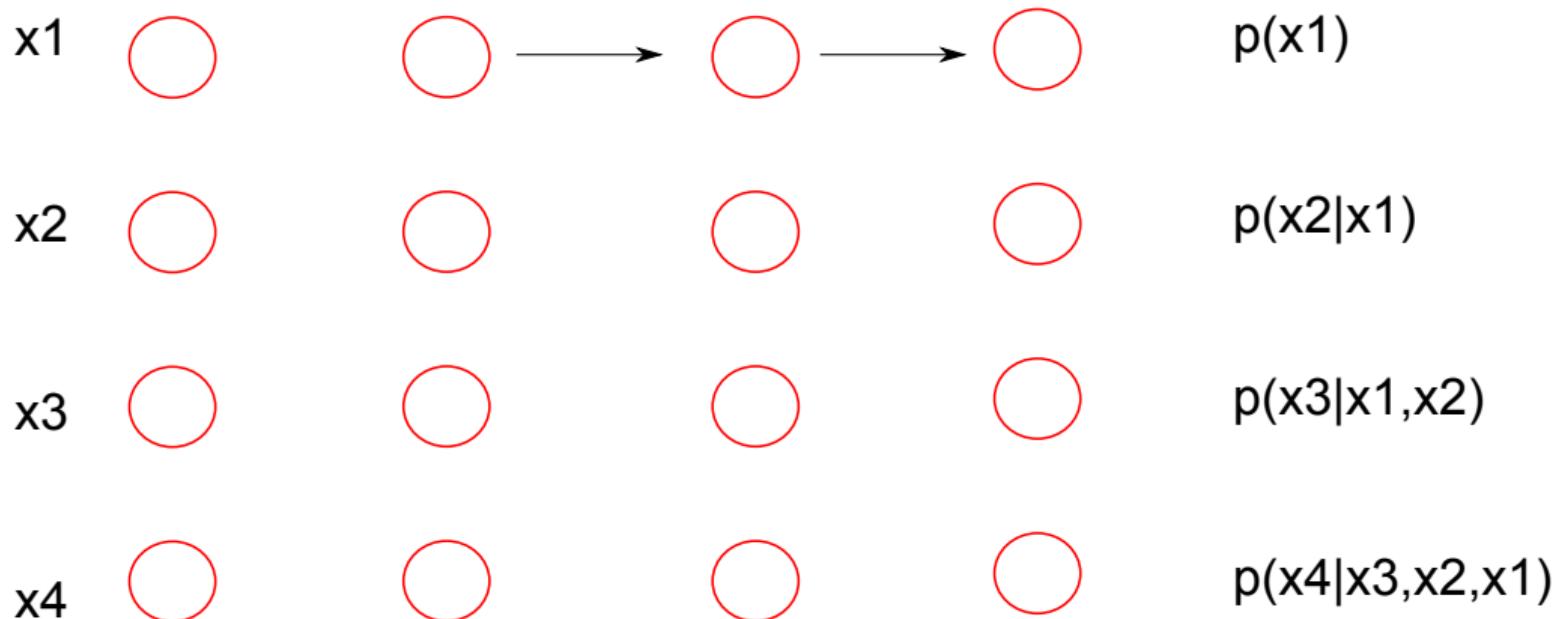
the sun roared and begun,  
but line of war, among the sky,  
and a sound is in his eyes;  
and the true heard the sets so fair  
and water thrickes hill,  
o'er dark adore with his hull dream,  
with a sheet for earth's luck and run,  
the roof with the border hair;

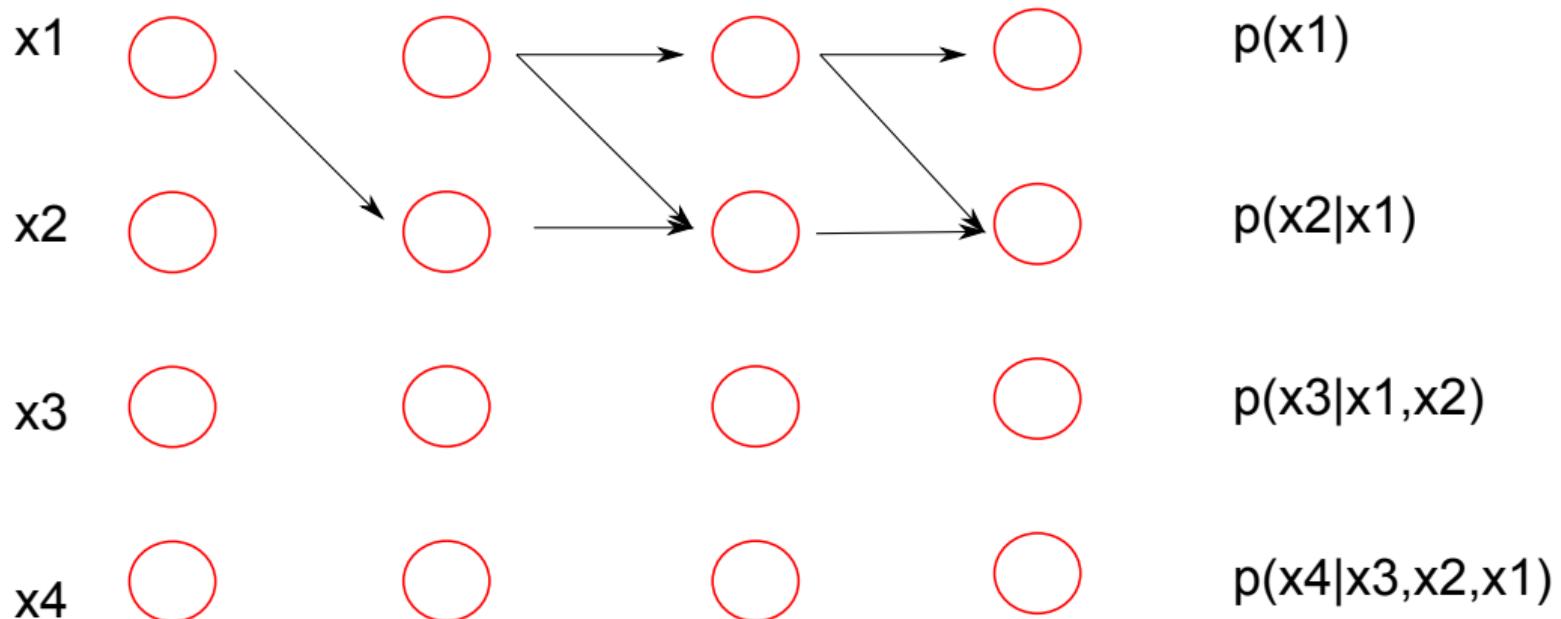
and the grey house put fled in a bird  
the moon and out and sold;  
there is no sight nor pray  
the book with a silent weat;

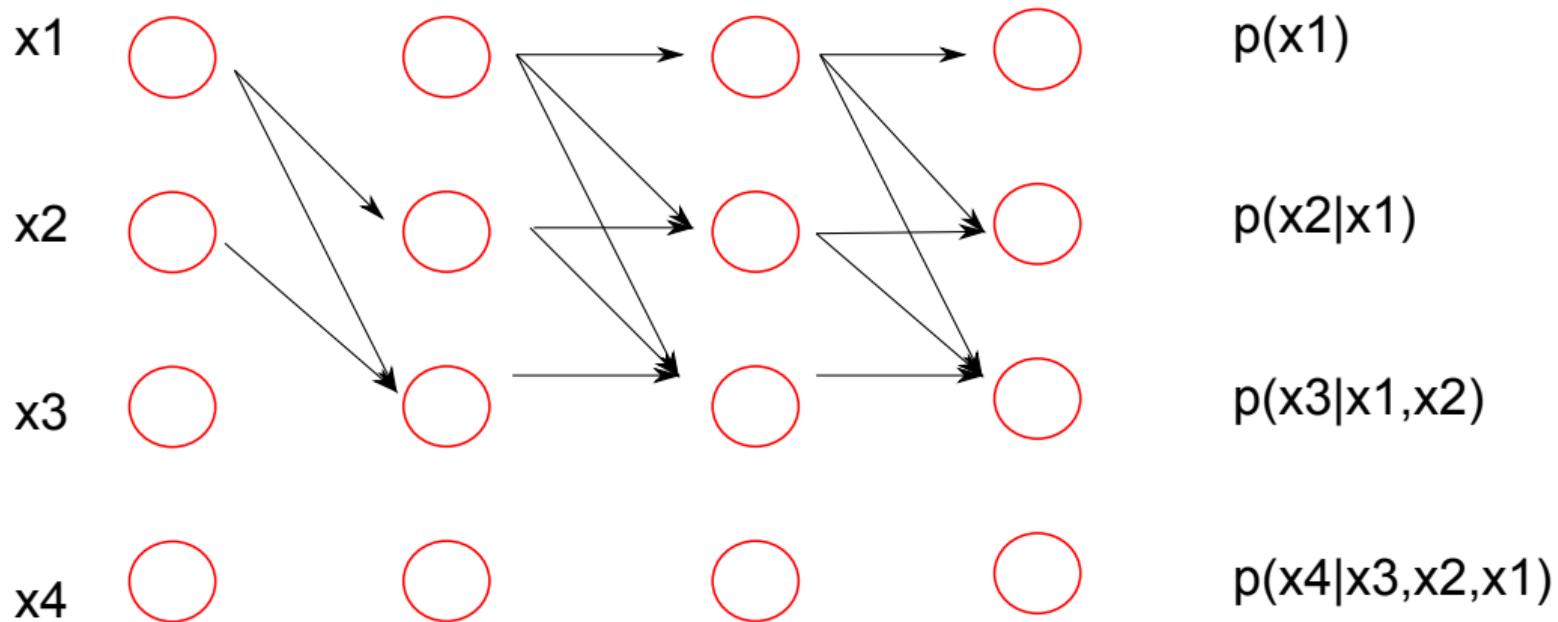
- ⑤ In order to reduce computational cost we should be able to parallelize the training adjusting all the conditionals at the same time
- ⑥ Two solutions
  - Masked MLPs: MADE
  - Masked Convolutions and attention

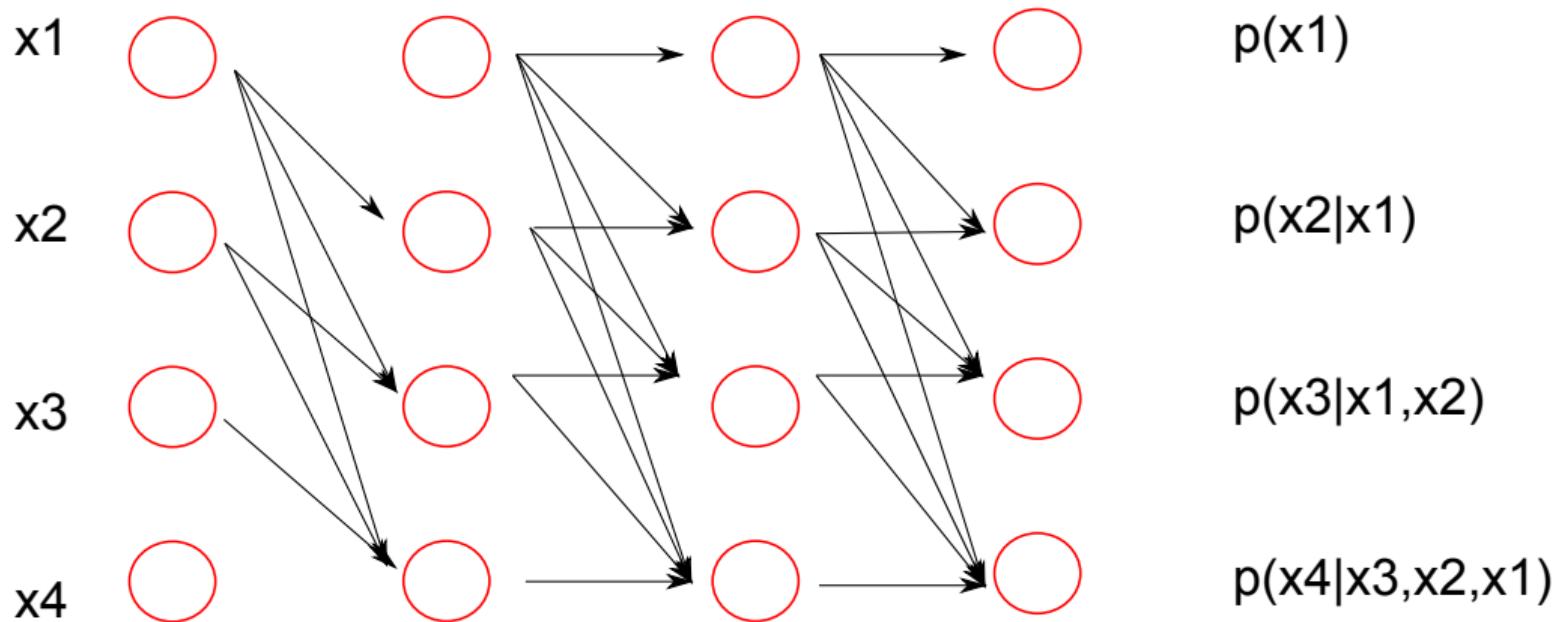
- The goal is to define an architecture based on autoencoders that satisfies the autoregressive property
- We can use masks to eliminate the connections in the autoencoder so for each output  $x_d$  there are only paths that connect the  $x_{<d}$  inputs
- These masks correspond to matrices that have zeros on the positions of the connections that violate that constraint
- The output layer for each variable has a sigmoid activation function for binary variables (so a  $[0, 1]$  value is obtained) or a softmax for multivalued variables











- ④ For any architecture with an arbitrary number of layers and units per layer, we can define the autoregressive connectivity with a set of rules
- ④ We assign to each hidden unit of each layer a value  $m^l(k)$  that is a number between 1 and  $D - 1$  meaning the maximum number of connections it can have to the next layer
- ④ For the input layer (the  $d$  variables)  $m^0(d) = d$
- ④ The mask matrix  $M^{W^l}$  that connects two consecutive layers has a 1 in position  $(i, j)$  if  $m^l(i) \geq m^{l-1}(j)$ , and 0 otherwise
- ④ For the output layer ( $d$  outputs) the mask matrix  $M^V$  has a 1 in position  $(i, j)$  if  $i \geq m^L(j)$

- ⑤ For hidden layers the computation performed is:

$$h^l(x) = g(b + (W^l \odot M^{w^l})x)$$

- ⑥ For the output layer

$$\hat{x} = \sigma(c + (V \odot M^V)h^L(x))$$

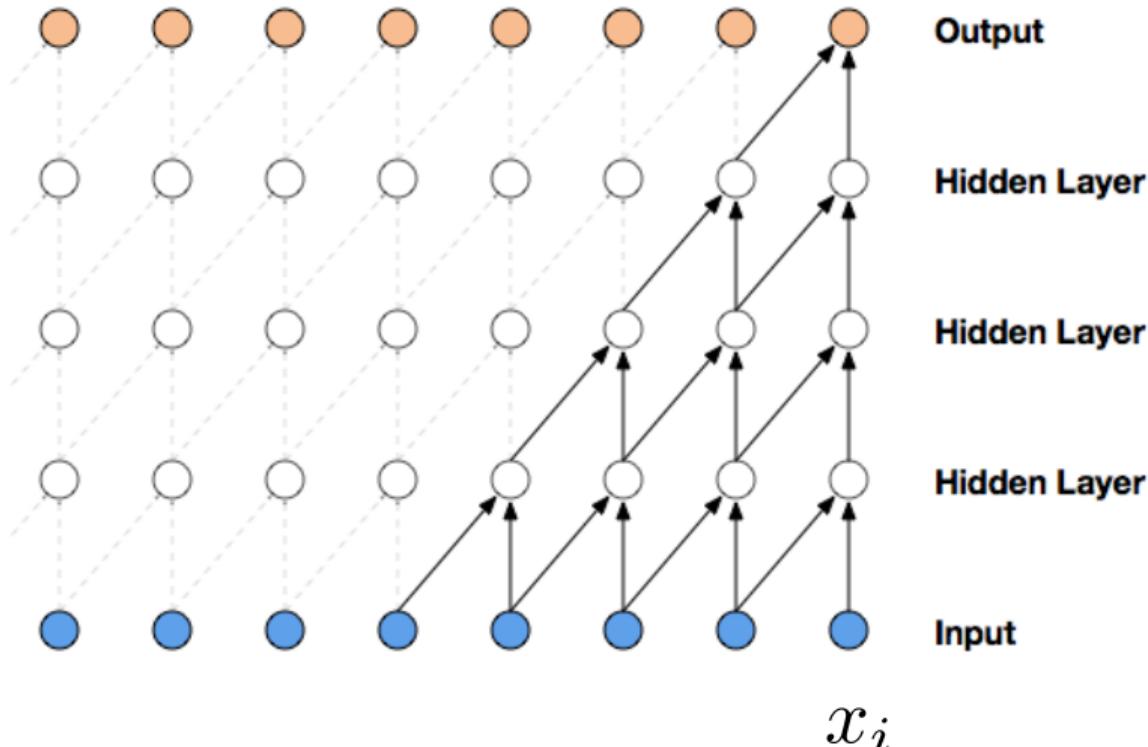
- ⑦ Sampling is done autoregressively

1. Sample  $x_1$  from  $p(x_1)$
2. input sampled variables 1 to  $i - 1$  to obtain  $p(x_i|x_1, \dots, x_{i-1})$
3. Sample variable  $x_i$
4. Repeat from 2 until  $d$

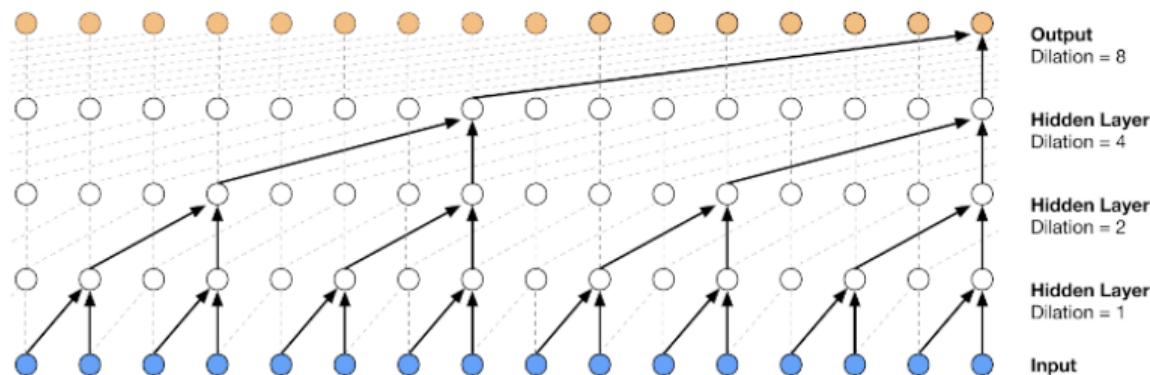
- ⑧ **Problem:** Not very efficient

- One dimensional convolutions allow focusing on a few elements of a sequence
- Each convolutional filter is applied to a subsequence:  $[1 \times k]$  filter
- We can have several convolutional filters and have many layers that combine the results of the filters of previous layers
- Temporal convolutions mask the input of the filters so no elements that correspond to the future are used to compute the next step (unlike normal convolutions), this is called **causal convolutions**
- Convolutions are fast to compute
- Inference is also autoregressive, but is faster given that the number of connections is reduced

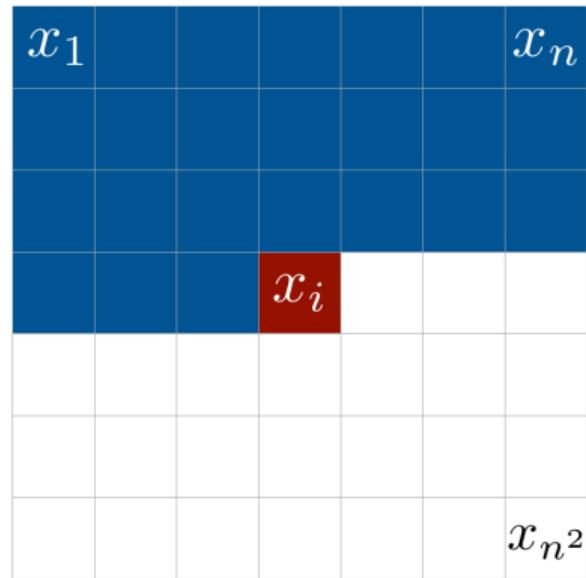
$$p(x_{i+1} | x_{\leq i})$$



- ⑤ Convolutions have a limited receptive field determined by the size of the filters, and the depth of the network
- ⑥ Wavenet used exponential dilated convolutions to increase the receptive field
- ⑦ Also include other improvements as residual blocks and skip connections

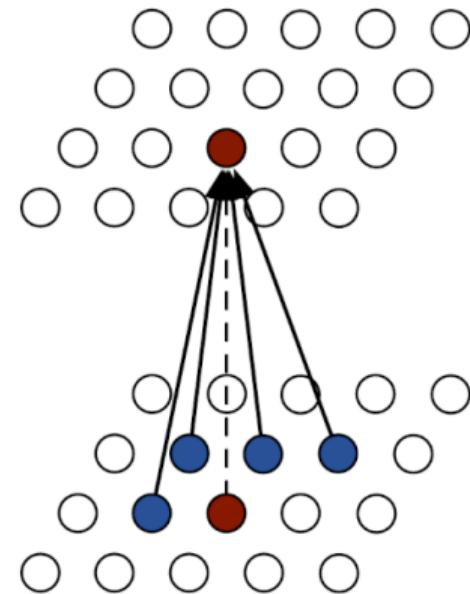


- If we linearize the pixels if a 2D image we lose a lot of information
- We can use 2D convolutions with adequate masking to capture the spatial relationships among pixels maintaining an autoregressive order
- We can use for instance the raster order

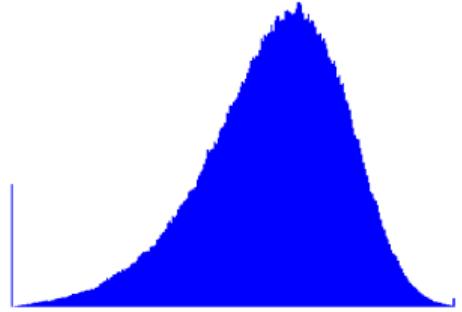


- PixelCNN uses masked 2D convolutional filters
- The convolutional filters are multiplied by a mask that respects the raster order
- We generate a pixel only using the pixels precedent in that order

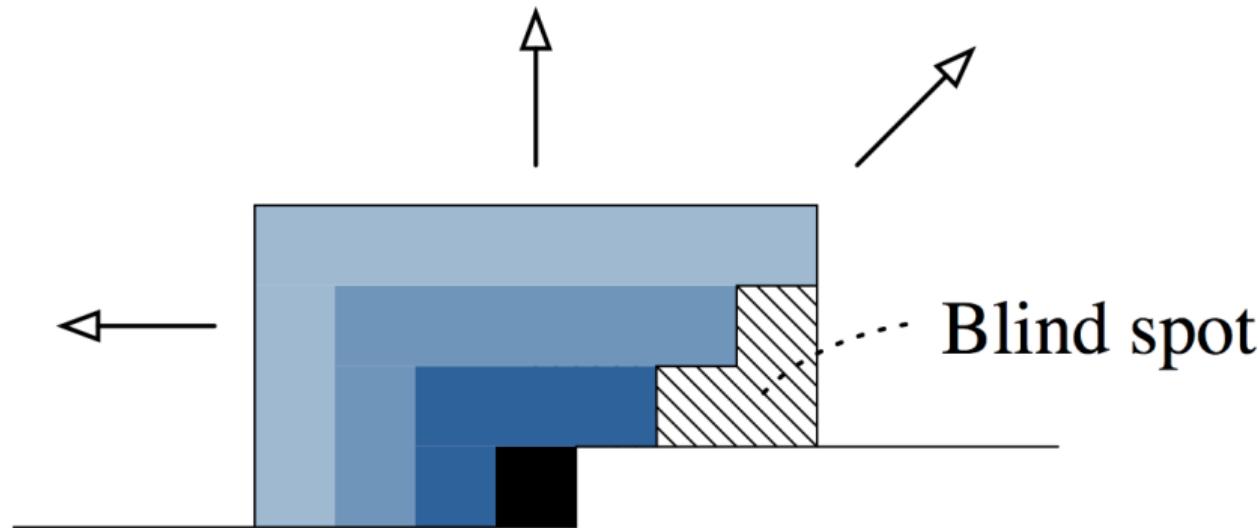
1	1	1
1	0	0
0	0	0



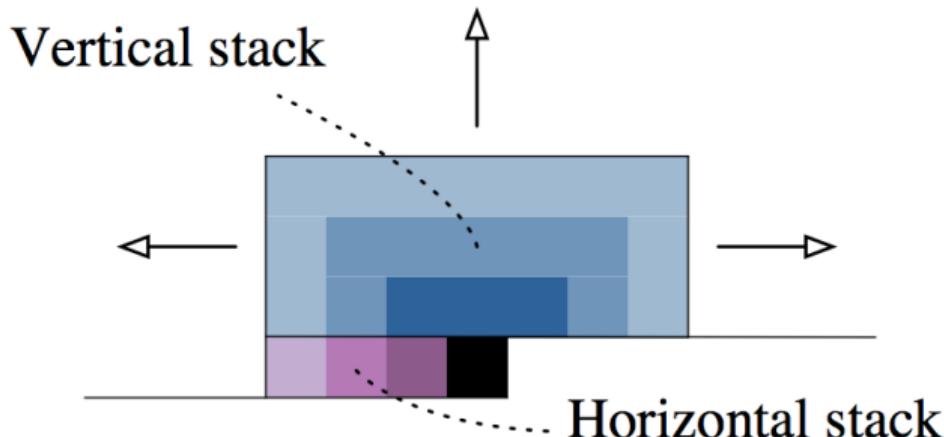
- Sampling is performed in autoregressive way
- Now we have the  $N \times N$  pixels for three channels (RGB)
- Each time we obtain a probability distribution that corresponds to a softmax over 255 values
- We sample a value from this distribution and add the value to the image and to the input to predict the next value



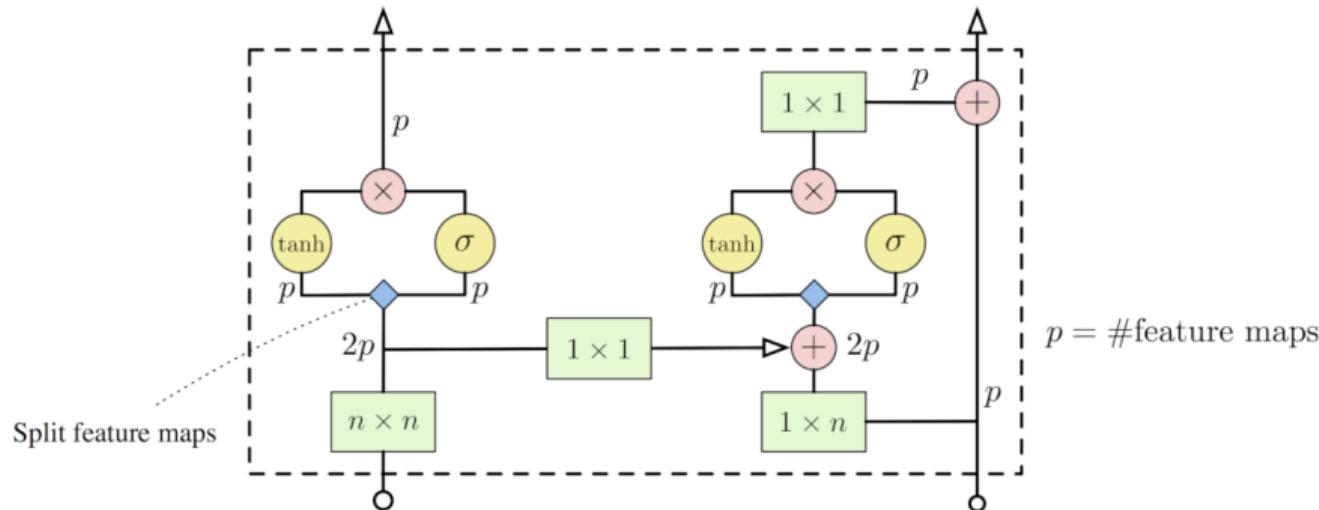
- PixelCNN has a problem with the receptive field of the convolutions
- As we add more layers the upper right side of the image is not used by the convolutions



- ⑤ We can eliminate the blindspot by adding two processing stacks
- ⑥ A horizontal stack that performs 1-D causal convolutions
- ⑦ A vertical stack that performs 2-D convolutions that conditions on the rows above the pixel to predict (only part of the receptive field is used)



- ⑤ The final architecture is composed of many layers (to expand the effective receptive field)
- ⑥ The results of the vertical stack is combined and added to the results of the horizontal stack

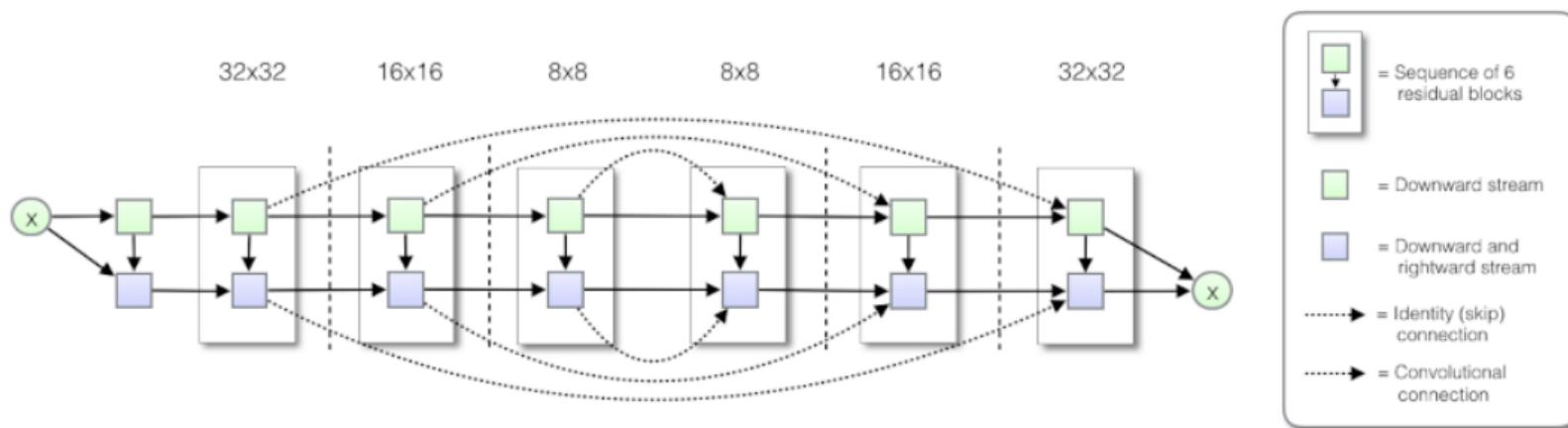


- Ⓐ To model the conditional probability using softmax is memory consuming
- Ⓐ The softmax is not aware of the order of the pixel values
- Ⓐ The conditional probabilities can be represented using a continuous distribution that can be discretized for predictions
- Ⓐ **Solution:** To use a mixture of logistic distributions

$$p(x|\pi, \mu, s) = \sum_{i=1}^K \pi_i [\sigma((x + 0.5 - \mu_i)/s_i) - \sigma((x - 0.5 - \mu_i)/s_i)]$$

- ⑤ The mixture of logistic distributions approximates the histogram corresponding to the pixel values probabilities
- ⑥ The continuous distribution is discretized to the pixel values by dividing the range of the values of the probabilities
- ⑦ In practice the number of elements in the mixture is small, so the memory consumption is reduced

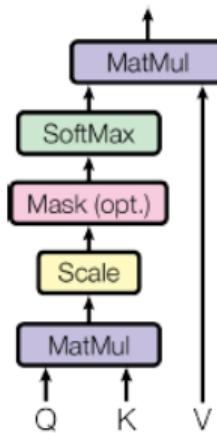
- ⑤ Different improvements are added to the architecture apart from the two stream of vertical and horizontal convolutions
  - U-net style skip connections
  - Downsampling and upsampling



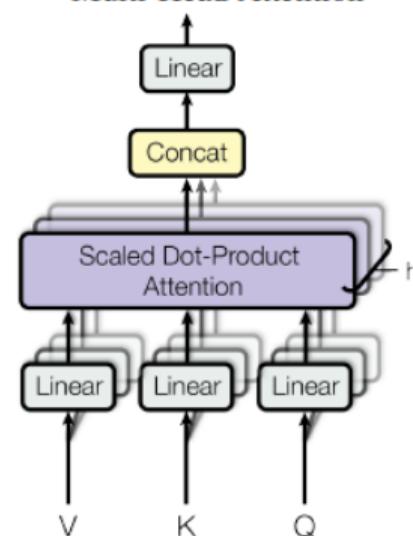
- ⑤ Convolutions have the problem of a limited receptive field, that reduces how long are the dependencies that we can capture
- ⑤ Self Attention provides a better solution
  - It has an unlimited receptive field
  - The number of parameters does not increase with the number of dimensions
  - Computations can be done in parallel

- Attention is a specialized layer developed in NLP
- It allows working with sequences
- Computes for an output what is the influence of the elements from the input sequence for predicting it

Scaled Dot-Product Attention



Multi-Head Attention



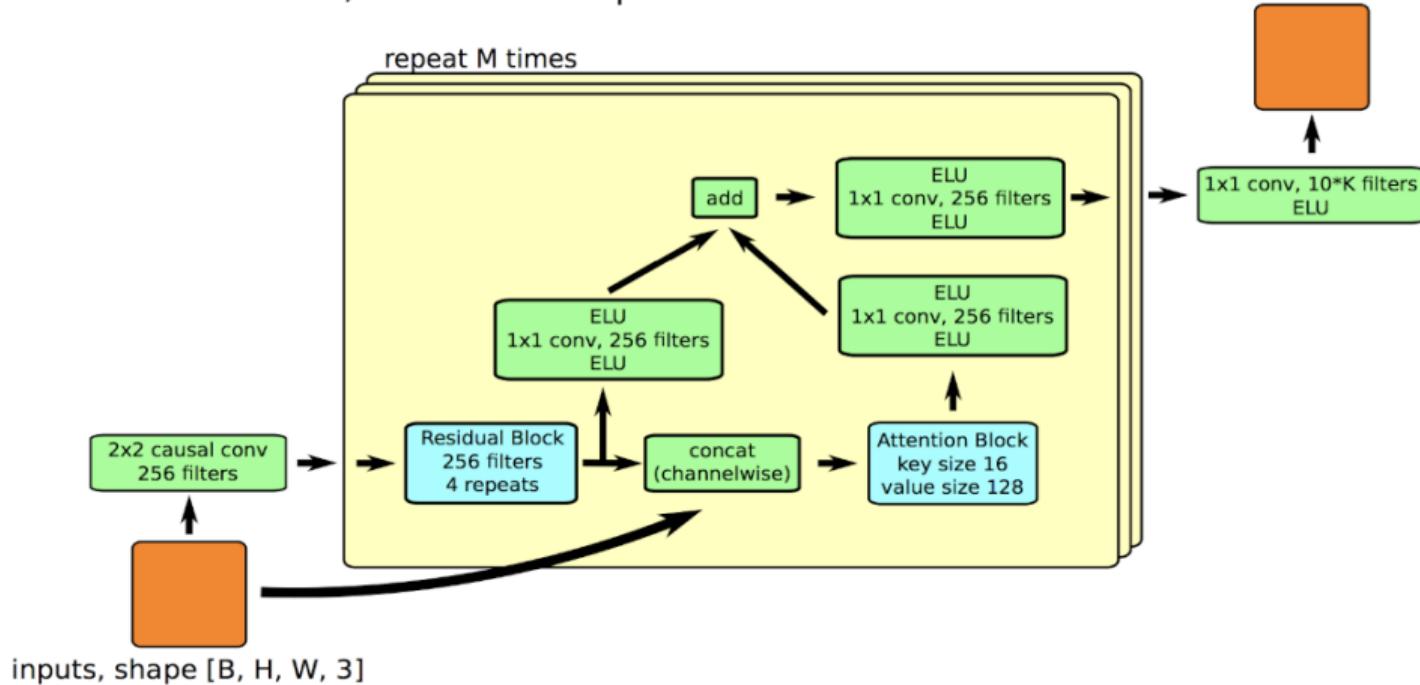
- An attention layer receives three elements: Query, Key and Value
- These three elements correspond from outputs from some part of the network and are learnable parameters
- If they all come from the input it is called **self attention**
- These three elements are combined through a softmax

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

- We can apply a mask in order to decide what elements are used in the combination, for instance to impose an autoregressive order (just make zero the components to mask)
- The mask can be in any order (unlike for convolutions)

PixelSNAIL: M blocks, K mixture components

outputs, shape [B, H, W, 10\*K]



① Pros:

- They are expressive models given that the probability factorization is general
- Parameter sharing allows generalization
- There are many state-of-the-art models for different datasets

② Cons:

- Prediction must be done autoregressively, this means an inference pass for each pixel (channel) to predict
- Different works for speeding up prediction: Multi scale, breaking up autoregressive pattern, caching of results . . .

# UDL: Flow Models

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



- Autoregressive models are limited to discrete variables
- We want to be able to fit probability density functions for **continuous variables**
- We want to obtain a **latent representation** that is meaningful for our problem  
(captures characteristics that are semantically relevant)
- We want also the usual:
  - A good fit to the training data (generalization)
  - Be able to compute  $p_\theta(x)$  for new examples  $x$  (efficiently better)
  - Able to sample from  $p_\theta(x)$

# 1-D Flow models

- ① Fitting a density model means to obtain the parameters from a sample for a function that complies with the properties of a PDF

$$P(x \in [a, b]) = \int_a^b p(x)dx$$

$$\int_{-\infty}^{\infty} p(x)dx = 1 \quad p_{\theta}(x) \geq 0 \quad \forall x$$

- ② This can be done using maximum likelihood

$$\max_{\theta} \sum_i \log(p_{\theta}(x_i))$$

or

$$\min_{\theta} \mathbb{E}_x[-\log(p_{\theta}(x))]$$

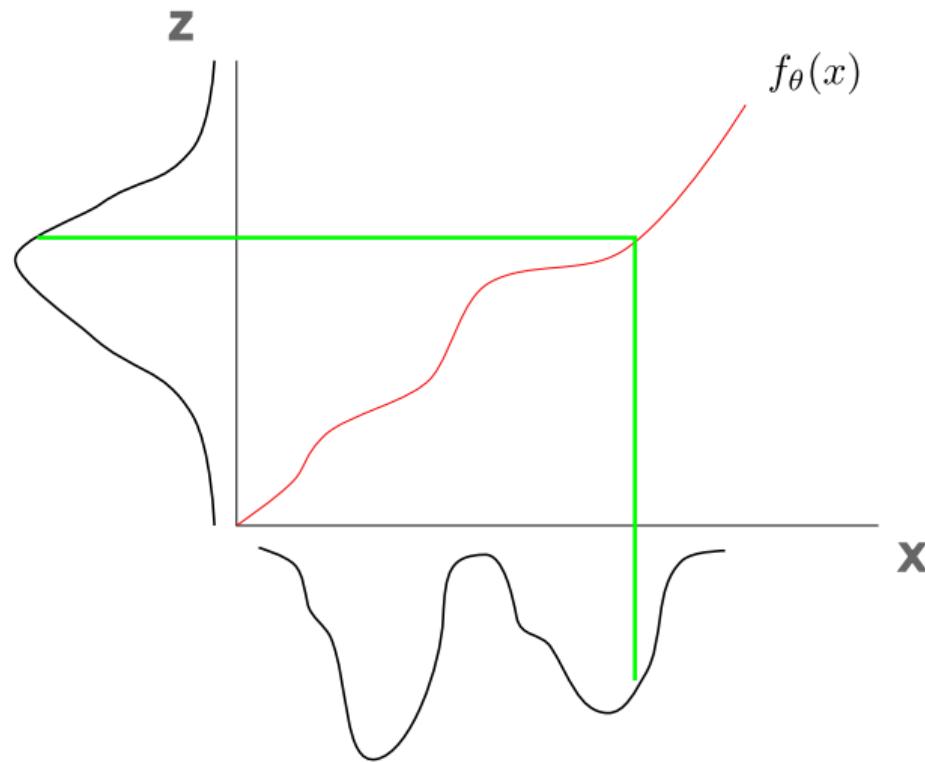
- We already have seen how to fit a density model using Gaussian Mixture Models

$$p_{\theta}(x) = \sum_{i=1}^k \pi_i \mathcal{N}(x|\mu_i, \sigma_i^2)$$

- Unfortunately this model does not work well in natural applications with large dimensionality
- Sampling is:
  - Pick a centroid
  - Add Gaussian noise to the centroid
- For instance for image generation this will obtain blurry images

- Flows are a general method for fitting density probability models
- They are based on transforming a density model into another through a series of transformations
- These sets of transformations maintain the properties of being a density probability model
- The original density adapts to the target distribution

- A flow is composed by a chain of transformations  $f_\theta = f_1, \dots, f_n$
- Each one preserving the density properties
- The results will be  $z$ , a value that is a transformation of  $x$  through the chain of functions  $z = f_\theta(x)$
- $z$  will belong to a probability distribution  $p_Z(z)$  of our choice
- When  $z$  belongs to a Normal distribution  $\mathcal{N}(0, 1)$  this is called a **Normalizing Flow**
- To require a specific target distribution for  $z$  will force the function  $f_\theta$  to be a distribution
- Remember Quantile and Power transformation for normalizing data?



- ④ The requirements for being a flow function are that it has to be monotone, invertible and differentiable
- ④ The goal is:
  - to map uniquely any element from one density to another (monotone)
  - to be able from samples from  $z$  to obtain samples from  $x$ , so we can perform sampling (invertible), so  $x = f_\theta^{-1}(z)$
  - to be able to train the function using gradient descent (differentiable) using maximum likelihood

$$\max_{\theta} \sum_i \log p_{\theta}(x_i)$$

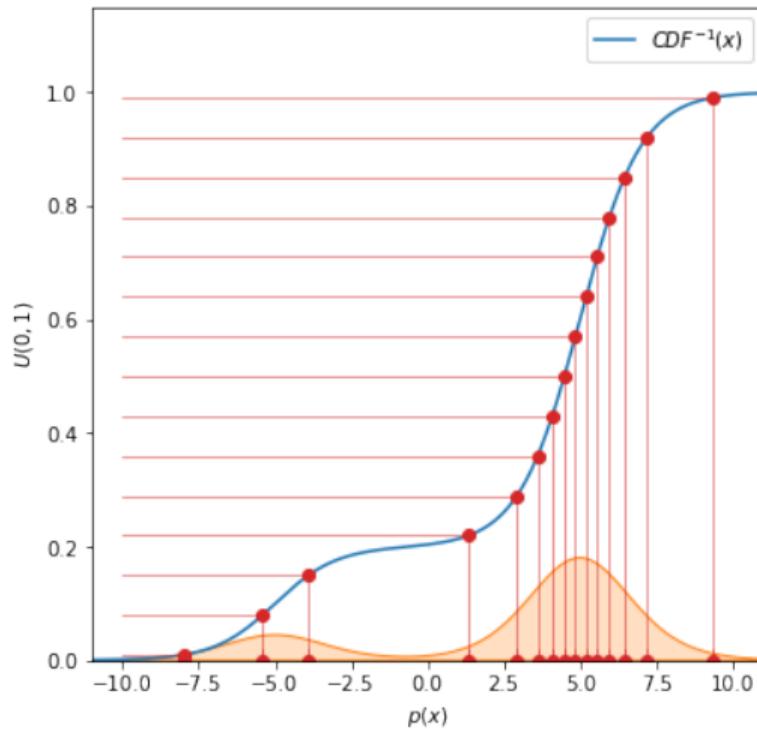
- ④ We do not have access directly to  $p_\theta(x)$ , so we make use of the transformation with a variable change

$$\begin{aligned} z &= f_\theta(x) \\ p_\theta(x)dx &= p_Z(z)dz \\ p_\theta(x) &= p_Z(f_\theta(x)) \left| \frac{\partial f_\theta(x)}{\partial x} \right| \end{aligned}$$

- ⑤ Given that we have an expression for  $p_Z$

$$\max_{\theta} \sum_i \log p_\theta(x_i) = \max_{\theta} \sum_i \log p_Z(f_\theta(x_i)) + \log \left| \frac{\partial f_\theta(x_i)}{\partial x} \right|$$

- We can transform any smooth pdf function to a uniform distribution using its cumulative distribution function (CDF)
- The CDF is a monotone, differentiable and invertible function
- The composition of flows is also a flow
- We can transform any pair of distributions through these transformations



- We have some constraints that must be followed:
  - Activation function must be differentiable and invertible (eg. sigmoid, tanh, Leaky ReLUs, **but not ReLUs**)
  - The input and output must have the same dimensionality
- The compositability of flows allows having different processing layers, so we can use deep models

## N-D Flow models

---

- Autoregressive flows implement a transformation of the form

$$z_i = f_\theta(x_i | x_{<i}) \quad x_i = f_\theta^{-1}(z_i | x_{<i})$$

- Fitting is the same as before but each variable is going to add a term to the likelihood

$$p_x(x) = \prod_{i=1}^D p_x(x_i | x_{<i})$$

So

$$\max_{\theta} \sum_i \sum_j \log p_z(f_\theta(x_{ij} | x_{<j})) + \log \left| \frac{\partial f_\theta(x_{ij} | x_{<j})}{\partial x_j} \right|$$

- ④ Training can be done by gradient descent, given that we have all the  $x$  we can fit the different functions in parallel
- ④ Sampling is going to be slow given that we have to compute each  $z_{<i}$  for computing  $z_i$
- ④ This could be impractical for domains with high dimensionality

- Given that flows are invertible functions we can reformulate the problem as

$$z_i = f_\theta^{-1}(x_i | z_{<i}) \quad x_i = f_\theta(z_i | z_{<i})$$

- This makes the model slower to train, but with a fast sampling
- There are several models that use this formulation like Parallel Wavenet and IAF-VAE
- To reduce the number of parameters of the models, the same strategies as autoregressive models can be applied using masking and recurrent neural networks

- ⑤ Autoregressive flows only change one variable at a time, we could work with all variables
- ⑥ The transformation  $f$  from  $x$  to  $z$  is rescaling the probability mass from one volume to another, we can write the relation among the two distributions as:

$$p(x)vol(dx) = p(z)vol(dz)$$

In other terms, the transformation must distribute the probability mass from  $x$  to fill the probability mass of  $z$

- ⑦ We can rewrite  $p(x)$  as:

$$p(x) = p(z) \frac{vol(dz)}{vol(dx)} = p(z) \left| \det \frac{dz}{dx} \right|$$

- ④ This reformulation allows to write the change of variable formula as:

$$p_{\theta}(x) = p(f_{\theta}(x)) \left| \det \frac{\partial f_{\theta}(x)}{\partial x} \right|$$

- ④ That can be optimized using maximum likelihood as:

$$\min_{\theta} \mathbb{E}_x[-\log(p_{\theta}(x))] = \mathbb{E}_x \left[ -\log p_z(f_{\theta}(x)) - \log \left| \det \frac{\partial f_{\theta}(x)}{\partial x} \right| \right]$$

- ④ This is only scalable if the determinant of the jacobian is easy to compute (basically the jacobian needs to be a triangular matrix, so the determinant is the product of the diagonal elements)
- ④ Flows can be composed to increase expressive power

- Affine transformations can be used to compute flows
- The parameters are an invertible matrix  $A$  and a vector  $b$  (basically a linear one layer perceptron)

$$f(x) = A^{-1}(x - b)$$

- Sampling can be performed using a gaussian distribution  $\mathcal{N}(0, 1)$

$$x = Az + b$$

- Computing the log likelihood is expensive for high dimensionality, it involves computing  $\det(A)$ , that is not usually a triangular matrix

- ⑤ We could build a flow using each variable independently with different choices for the transformation function

$$f_\theta(x_1, \dots, x_n) = (f_\theta(x_1), \dots, f_\theta(x_n))$$

- ⑥ The jacobian is diagonal, so the determinant is easy to compute

$$\begin{aligned}\frac{\partial z}{\partial x} &= \text{diag}(f'_\theta(x_1), \dots, f'_\theta(x_n)) \\ \det \frac{\partial z}{\partial x} &= \prod_{i=1}^d f'_\theta(x_i)\end{aligned}$$

- ⑦ We gain efficiency, but we lose expressive power

- ④ We can combine these two ideas to obtain expressiveness and efficiency
- ④ Split the variables in two  $x_1, \dots, x_{\frac{d}{2}}, x_{\frac{d}{2}+1}, \dots, x_d$
- ④ Separate the transformation in two

$$\begin{aligned} z_{1:\frac{d}{2}} &= x_{1:\frac{d}{2}} \\ z_{\frac{d}{2}+1:d} &= x_{\frac{d}{2}+1:d} \cdot s_\theta(x_{1:\frac{d}{2}}) + t_\theta(x_{1:\frac{d}{2}}) \end{aligned}$$

- ④ This is an invertible transformation
- ④  $s_\theta$  and  $t_\theta$  can be arbitrary neural networks

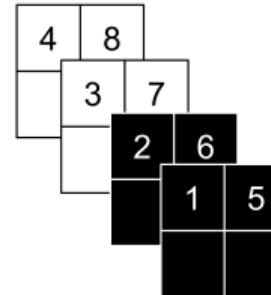
- ④ The jacobian has the form

$$\frac{\partial z}{\partial d} = \begin{bmatrix} I & 0 \\ \frac{\partial z_{\frac{d}{2}+1:d}}{\partial x_{1:\frac{d}{2}}} & \text{diag}(s_\theta(x_{1:\frac{d}{2}})) \end{bmatrix}$$

- ⑤ Given that is a triangular matrix the determinant can be computed as:

$$\det \frac{\partial z}{\partial d} = \prod_{i=1}^d s_\theta(x_{1:\frac{d}{2}})_k$$

- RealNVP: flow based architecture for generating images
- Uses a checkerboard pattern to partition the variables (the pattern matters)
- Implements a multi scale architecture that squeezes the input changing spatial resolution for number of channels  $(d, d, c) \rightarrow (\frac{d}{2}, \frac{d}{2}, 2 \times c)$
- Each layer uses alternates spatial and channel wise checkerboard patterns



- Affine based methods (IAF-VAE, NICE, RealNVP) have a limited expressiveness
- Using non-linear functions allow obtaining better results
  - Mixture of logistic functions + self attention (Flow++)
  - Piecewise linear and quadratic functions (splines)
  - Invertible  $1 \times 1$  convolutions + affine flow (Glow)  
(<https://openai.com/blog/glow/>)
  - Continuous time flows based on Ordinary Differential Equations (FFJORD)

# UDL: Latent Variable Models

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI

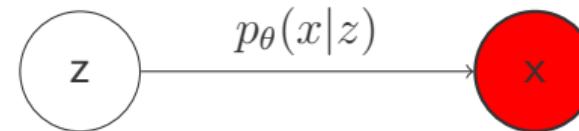


# Introduction

---

- For autoregressive and flow models we observe directly all the variables
- Latent Variable Models assume that there are a set of hidden variables that generate the observed variables
- These hidden variables possibly are less than the observable variables, are more interpretable and hold some semantic information (hidden representation)
- LVMs try to identify that representation from the observed data

- ⑤ LVMs define the relationship among latent and observed variables as probabilistic models



- ⑥ Sampling  $z \sim p_Z(z)$   $x \sim p_\theta(x|z)$
- ⑦ Evaluate Likelihood for samples, given that  $p(x, z) = p(x|z)p(z)$ , marginalizing by  $z$

$$p_\theta(x) = \sum_z p_Z(z)p_\theta(x|z)$$

- ⑧ Train

$$\max_{\theta} \sum_i \log p_\theta(x_i) = \max_{\theta} \sum_i \log \sum_z p_Z(z)p_\theta(x_i|z)$$

# Variational Inference

---

- ⑤ In order to compute this model we need to obtain the probabilities that allows finding the latent variables, basically:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

- ⑥ We need to compute  $p(x)$ , if  $z$  is a vector of continuous values this means to integrate

$$\int_z p(x|z)p(z)dz$$

- ⑦  $p(z)$  is a multidimensional distribution that needs multiple integration that is usually intractable
- ⑧ An approach to approximate  $p(z|x)$  is **Variational Inference**

- ⑤ Variational inference substitutes an intractable probability distribution  $p$  by another well-behaved distribution  $q$
- ⑤ The goal is to transform the distribution  $q$  so it is as close as possible to  $p$  at least locally for a sample, tuning its parameters  $\phi$
- ⑤ Basically we need a distribution  $q_\phi(z|x)$  that is close to  $p_\theta(z|x)$
- ⑤ We can measure the closeness between two probability distributions using the Kullback-Leibler Divergence (KL)

$$KL(q||p) = - \sum_x q(x) \log \frac{p(x)}{q(x)} = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

- ⑤ This is not a symmetric measure, so it is not a proper distance

- ④ We are going to choose a distribution  $q_\phi(z|x)$  as a substitute for  $p_\theta(z|x)$
- ④ In order to choose the parameters of  $p_\phi$  we need to minimize the KL Divergence

$$KL(q_\phi(z|x) || p_\theta(z|x)) = \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

- ④ With this we will have an optimization objective

$$\begin{aligned}
\sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} &= \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x, z)/p_\theta(x)} \\
&= \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \left[ \log \frac{q_\phi(z|x)}{p_\theta(x, z)} p_\theta(x) \right] \\
&= \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \left[ \log \frac{q_\phi(z|x)}{p_\theta(x, z)} + \log p_\theta(x) \right] \\
&= \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x, z)} + \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log p_\theta(x) \\
&= \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x, z)} + \log p_\theta(x) \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \\
&= \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x, z)} + \log p_\theta(x)
\end{aligned}$$

- ④ So we have that

$$KL(q_\phi(z|x) || p_\theta(z|x)) = \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x, z)} + \log p_\theta(x)$$

- ④ That can be rewritten as:

$$\begin{aligned} \log p_\theta(x) &= KL(q_\phi(z|x) || p_\theta(z|x)) - \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(x, z)} \\ &= KL(q_\phi(z|x) || p_\theta(z|x)) + \sum_{z \sim q_\phi(z|x)} q_\phi(z|x) \log \frac{p_\theta(x, z)}{q_\phi(z|x)} \end{aligned}$$

- ④  $\log p_\theta(x)$  is the likelihood of the distribution of the samples

- ④ The second term is called the Variational Lower Bound (VLB) or Evidence Lower Bound (ELBO)
- ④ The KL divergence is always larger or equal to 0, so the ELBO is a lower bound of the probability distribution  $p(x)$
- ④ if  $KL(q_\phi(z|x)||p_\theta(z|x)) = 0$  then the second term is exactly the likelihood of  $p(x)$
- ④ The KL divergence is defining actually two distances
  - How close are the two distributions
  - What is the tightness between the ELBO and the likekihood of the marginal distribution

- ④ We can rewrite the ELBO to obtain an interesting interpretation

$$\begin{aligned}
 \sum_z q_\phi(z|x) \log \frac{p_\theta(x, z)}{q_\phi(z|x)} &= \sum_z q_\phi(z|x) \log \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)} \\
 &= \sum_z q_\phi(z|x) \log p_\theta(x|z) \frac{p_\theta(z)}{q_\phi(z|x)} \\
 &= \sum_z q_\phi(z|x) \left[ \log p_\theta(x|z) + \log \frac{p_\theta(z)}{q_\phi(z|x)} \right] \\
 &= \sum_z q_\phi(z|x) \log p_\theta(x|z) + \sum_z q_\phi(z|x) \log \frac{p_\theta(z)}{q_\phi(z|x)} \\
 &= \sum_z q_\phi(z|x) \log p_\theta(x|z) - KL(q_\phi(z|x)||p_\theta(z))
 \end{aligned}$$

- ④ The first term is a reconstruction error, and the second a regularization term

- ④ This function can be defined as expectations, and given that we are going to work with samples from the distributions we can use gradient descent for optimization

$$\mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \right] + \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p_\theta(x,z)}{q_\phi(z|x)} \right]$$

- ⑤ Maximizing the ELBO we maximize the likelihood of the marginal  $p(x)$  and minimize the divergence among the two conditional distributions

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p_\theta(x,z)}{q_\phi(z|x)} \right]$$

- ④ To train the parameters  $\phi$  and  $\theta$  of the distributions from data we will have

$$\mathcal{L}_{\theta,\phi}(\mathcal{D}) = \sum_{x \in \mathcal{D}} \mathcal{L}_{\theta,\phi}(x)$$

- ④ For a datapoint  $x$  we have:

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)} \left[ \log \frac{p_\theta(x, z)}{q_\phi(z|x)} \right] \\ &= \mathbb{E}_{q_\phi(z|x)} [\log p_\theta(x, z) - \log q_\phi(z|x)]\end{aligned}$$

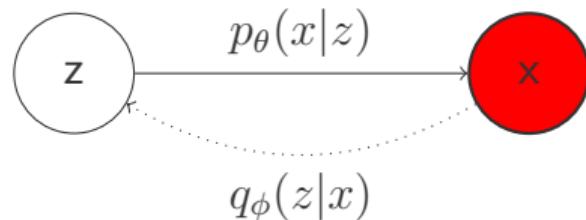
- ④ We can perform alternate gradient descent for this functions computing its derivative respect each group of parameters fixing one at a time

- ⑤ Given that we are working with expectations we can compute estimates for the parameters using samples and the derivatives of the ELBO performing SGD
- ⑥ The values of  $z$  will be generated from a known  $p(z)$  distribution
- ⑦  $\mathcal{L}_{\theta,\phi}(x)$  is not usually a tractable function, but estimates of the parameters can be computed
  - The derivative respect to  $\theta$  can be approximated using the derivative of the function that computes  $p$
  - The derivative respect to  $\phi$  needs to define a  $q$  that is tractable and needs a reparameterization from a fixed distribution (we will see an example)

# Variational AutoEncoders

---

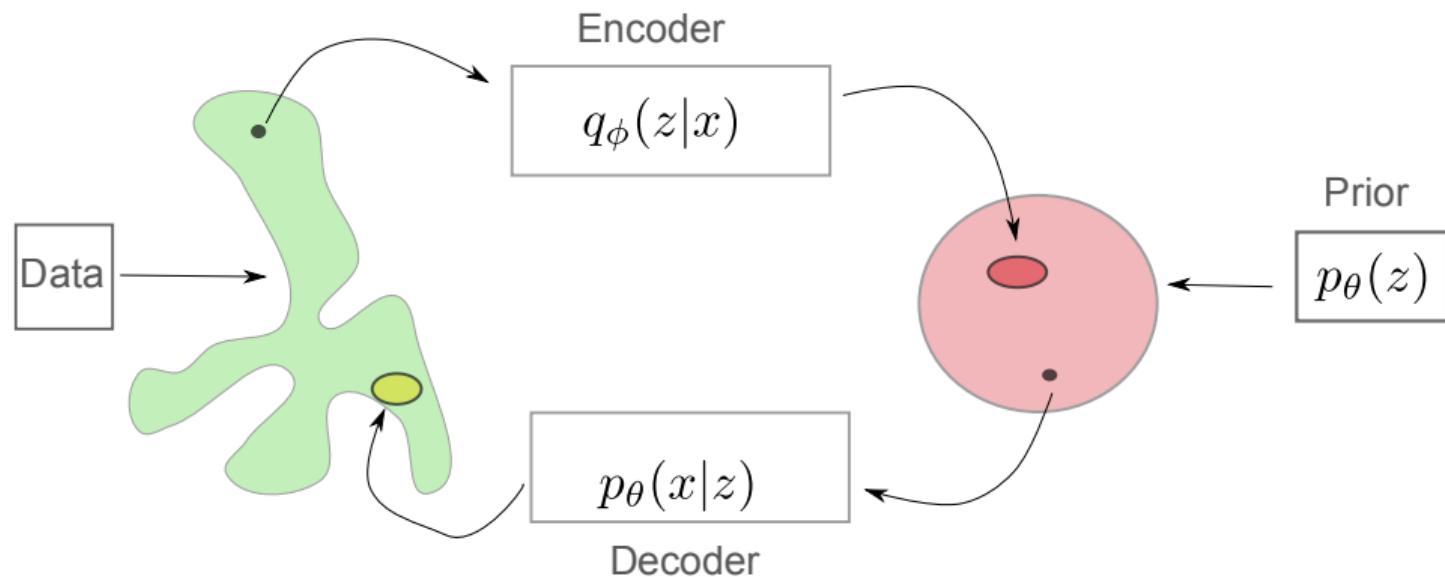
- ⑤ The goal of all these estimations is to have probability distributions that can map from the original data space to the latent space and back



- ⑥ This can be implemented as type of autoencoder: **Variational AutoEncoder (VAE)**

- ⑤ An autoencoder is a fully connected network that reproduces the input in the output (minimizes the reconstruction)
- ⑤ It has two parts
  - The **Encoder** transforms the input into a low dimensional code (succesive layers reduce the number of neurons)
  - The **Decoder** transform the code into the original input (succesive layers increase the number of neurons)
- ⑤ An autoencoder is deterministic, not probabilistic, so it can not be sampled

- ④ A Variational AutoEncoder is composed by two networks
- ④ The **encoder** learns the function  $q_\phi(z|x)$  obtaining an estimate for its parameters, this is also called the inference model or recognition network
  - From the parameters estimate of the encoder random samples can be drawn
- ④ The **decoder** learns the function  $p_\theta(x|z)$  that corresponds a generative model
  - The samples generated from the result from the encoder are passed through the decoder
- ④ After training we keep the decoder, we can obtain new samples processing samples from a prior distribution  $p_\theta(z)$



- In order to implement the VAE we need to pick a distribution for  $q$  that is easy to work with
- For continuous variables an evident choice is to use the gaussian distribution

$$q_\phi(z|x) = \mathcal{N}(z, \mu, \Sigma)$$

- To make things simpler we can use a gaussian with diagonal covariance (all latent variables are independent)

$$q_\phi(z|x) = \mathcal{N}(z, \mu, diag(\sigma)) = \prod_i q_\phi(z_i|x) = \prod_i \mathcal{N}(z_i, \mu, \sigma_i)$$

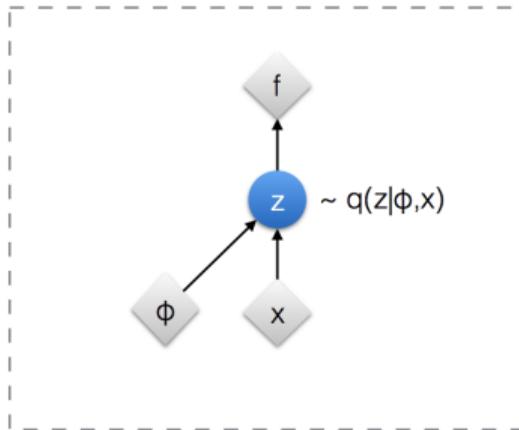
- So the encoder will estimate  $\mu$  and  $\sigma$  (actually  $\log(\sigma)$ ) and a sample will be obtained from the distribution to run the decoder

- ④ Given that the sampling is a probabilistic operation, we can not use it with backpropagation
- ④ In order to avoid this problem we can assume that we have a source of gaussian noise that provides us with samples
- ④ This gaussian distribution is chosen  $\epsilon \sim \mathcal{N}(0, 1)$
- ④ So we can obtain a sample computing

$$z = \mu + \sigma \odot \epsilon$$

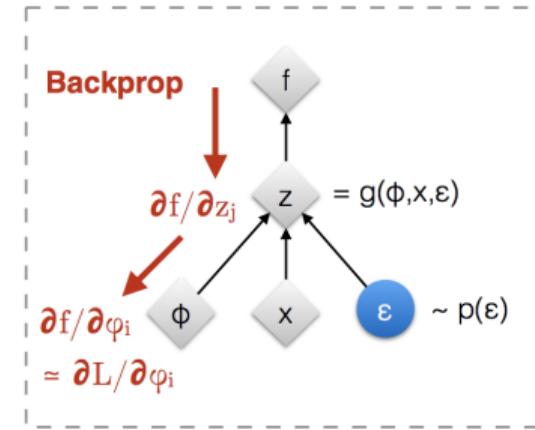
- ④ This is similar to what happened with the denoising autoencoder, but now the noise is added after the encoding

Original form

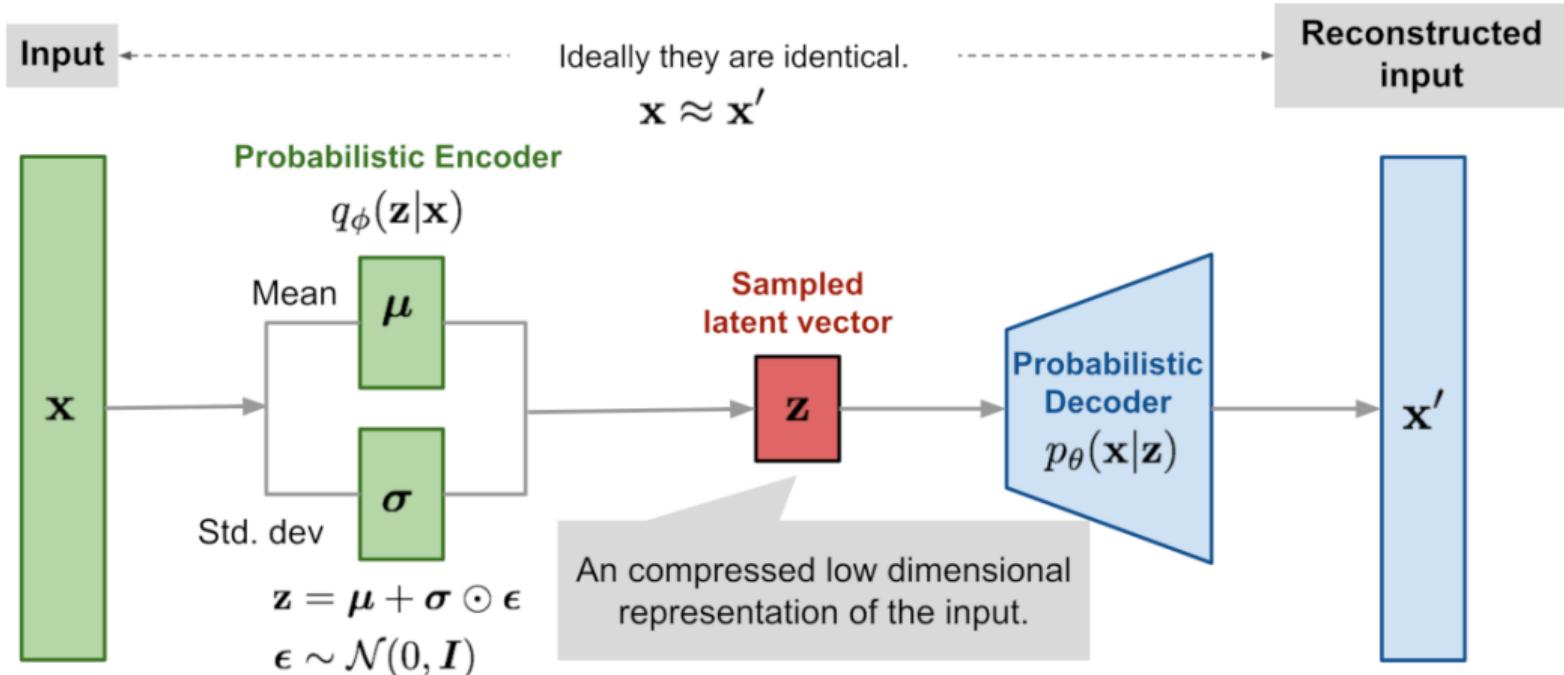


: Deterministic node  
 : Random node

Reparameterised form



[Kingma, 2013]  
[Bengio, 2013]  
[Kingma and Welling 2014]  
[Rezende et al 2014]



- Given the objective function of the VAE we use the regularization term to force the mapping distribution to get closer to the latent variables distribution

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{z \sim \phi(z|x)}[\log p_{\theta}(x|z)] - \beta KL(q_{\phi}(z|x)||p_{\theta}(z))$$

- The goal is to disentangle the latent variables, so they have a more semantic interpretation if possible
- The larger the parameter the more we force the similarity among the two distributions, With  $\beta = 1$  we have the original variational autoencoder
- For instance in the image domain obtaining latent variables that control brightness, color, position, ...

- ⑤ We can also introduce supervised information add to the representation a way to control what is generated
- ⑥ This can be done by conditioning the probability distribution in the encoder and the decoder using a discrete variable that indicates the class of the example

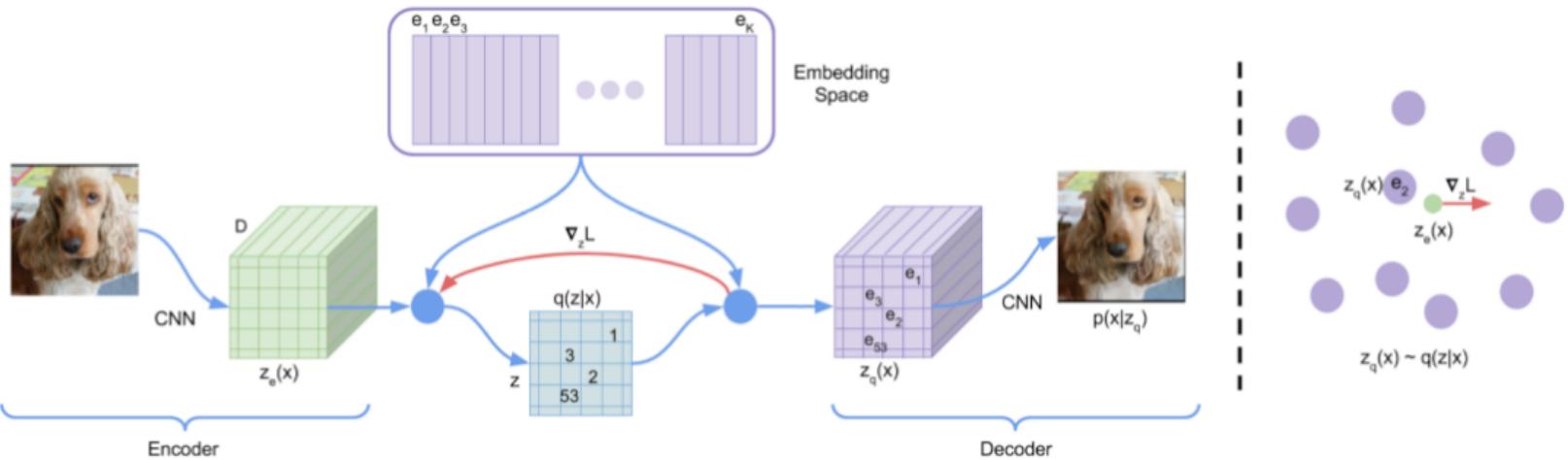
$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{z \sim \phi(z|x,c)} [\log p_{\theta}(x|z)] - KL(q_{\phi}(z|x,c) || p_{\theta}(z|c))$$

State of the art VAE

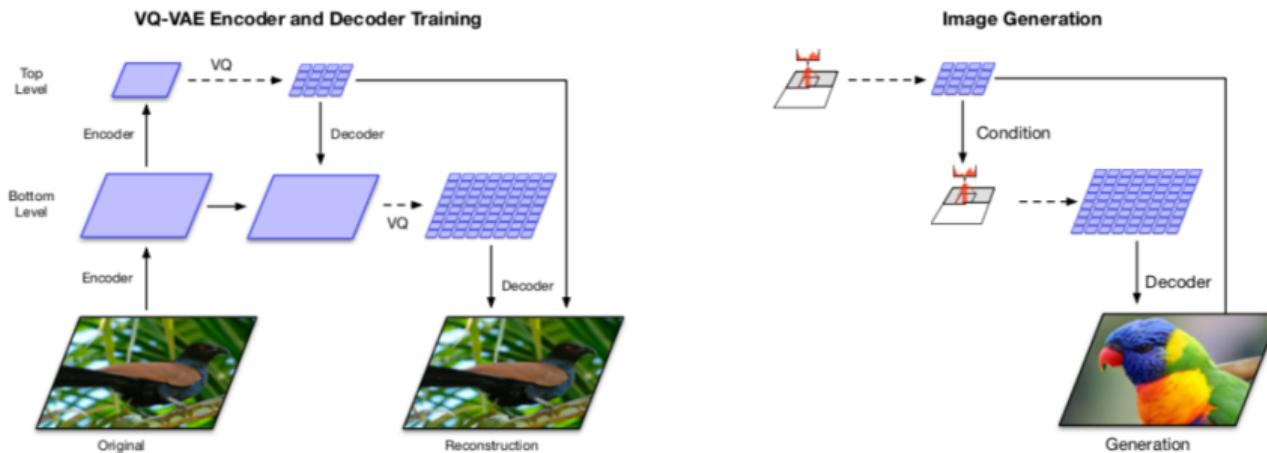
---

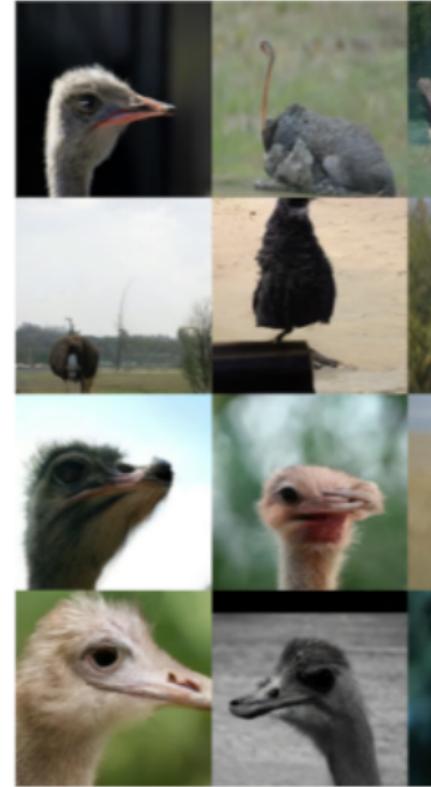
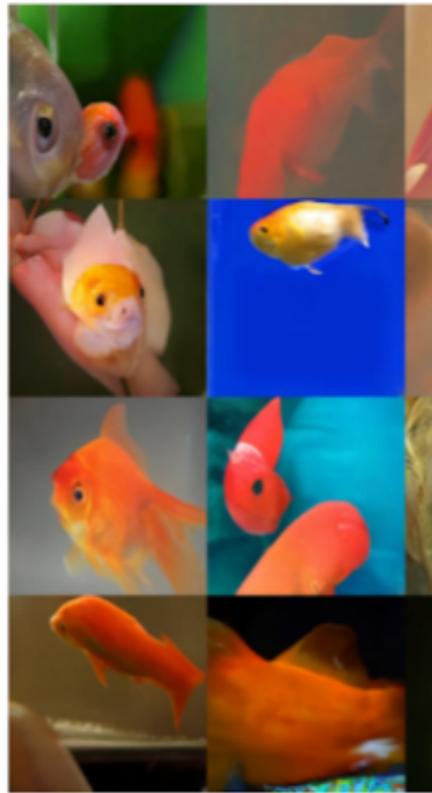
- VQ-VAE (Vector Quantised-Variational AutoEncoder) learns discrete latent variables, this can be more adequate for many problems (NLP, speech, ...)
- Vector quantization is a method for mapping K-dimensional vectors to a finite code, a set of vector centroids
- The architecture maintains a codebook of  $k$  vectors with dimensionality  $D$
- The distribution  $q$  is then a discrete distribution

- The encoder outputs a vector of dimension  $D$  for each variable in  $x$
- The euclidean distance for the  $k$  elements in the codebook is computed for each variable and the nearest code is obtained to obtain the encoded sample
- Each variable of the encoded sample corresponds to a vector from the code
- The encoded sample is passed through the decoder to generate the output



- ⑤ VQ-VAE 2.0 improves different elements of the initial architecture
  - Hierarchical multi-scale encoder with quantization at different resolutions
  - Autoregressive model in the decoder using pixelCNN and multiheaded attention
  - More memory and computation are needed





# UDL: Implicit Models

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI



# Introduction

---

- ⑤ All the generative models that we have seen are based on likelihood
  - Exact likelihood: Autoregressive and flow models
  - Approximate likelihood (ELBO): Variational AutoEncoders
- ⑥ All these models allow to:
  - Train to estimate a probability distribution
  - Compute likelihood of new samples
  - Obtain a latent representation
  - Obtain new samples

- There are applications where we only need samples, why not obtain a model that only cares about that
- We want models that
  - Generate samples not only from the training dataset (variations)
  - Able to interpolate samples between those seen on training
  - Obtain samples aware of the underlying factors of the process that generates the data
  - For instance: Different viewing angles, changes of illumination, object features (shape, color, elements composing the object)

- ④ Implicit models will be able to generate samples  $z$  from a reference fixed distribution  $z$  (gaussian, uniform)
- ④ Samples from  $z$  will pass through a DNN to compute the samples



- ④ What is different is that the DNN will not be learned to estimate explicitly a probability density function

- Given a set of samples  $\mathcal{X} = \{x_1, \dots, x_z\}$  from our target problem
- Given a sampler  $q_\phi(z) = DNN(z; \phi)$  where  $z \sim p(z)$
- $x = q_\phi(z)$  induces a density function  $p_{model}$
- There is no explicit distribution for  $p_{data}$  or  $p_{model}$ , only samples
- The goal is to make  $p_{data}$  and  $p_{model}$  as close as possible

- The main issue of defining our problem this way is that we do not have a simple way of comparing both distributions
- Before: Explicit distribution means to be able to measure likelihood and compare distributions (for instance using KL-Divergence), that gives an objective function to optimize
- Now: Implicit distribution means no likelihood, we can only obtain samples, means to use other similarity measures respect to the samples that will have a different behaviour than likelihood (no theory to back up decisions)

# Generative Adversarial Networks

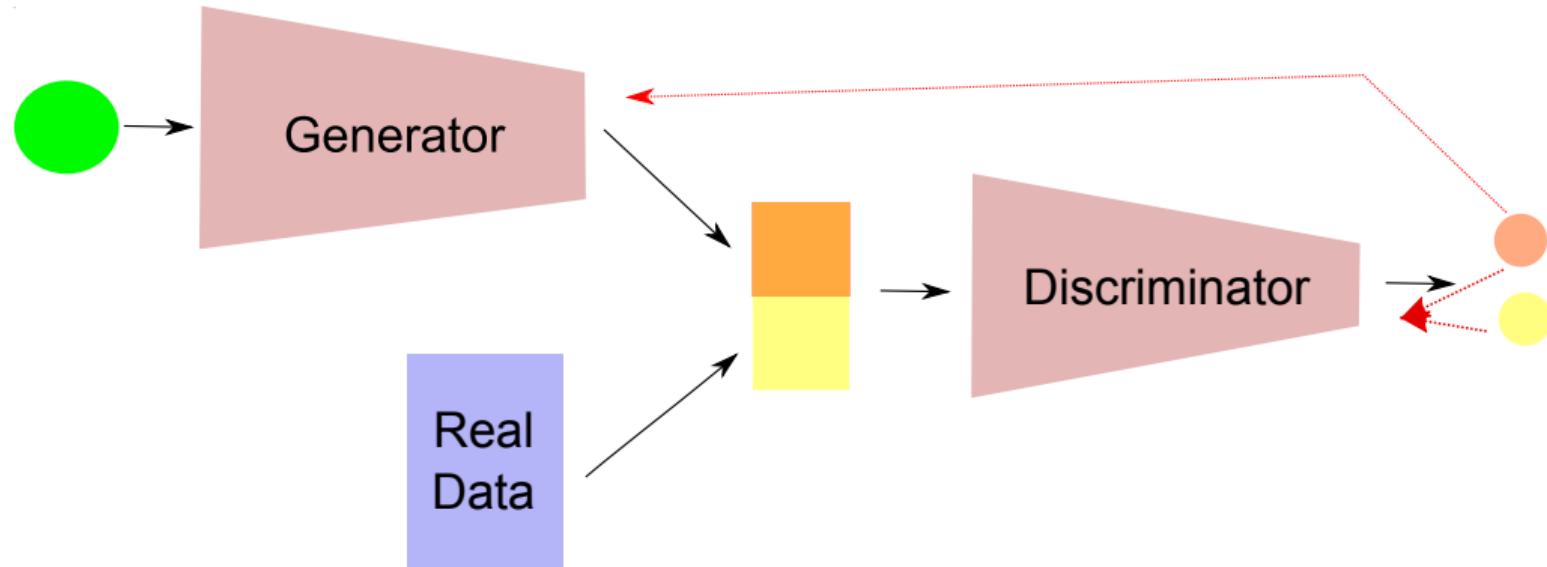
---

- Generative Adversarial Networks (GANs) were defined in the paper **Generative Adversarial Nets** (Goodfellow et al., 2014)
- The idea is to work with two models:
  - The **Generator** ( $G$ ) captures the data distribution and is able to generate samples
  - The **Discriminator** ( $D$ ) estimates the probability of a sample of being either from the original data or from  $G$
- The training is framed as an adversarial minimax game
  - the Generator tries to maximize the probability of their samples
  - the Discriminator tries to minimize that probability

- The goal of the minimax game can be defined as:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$$

- The discriminator  $D$  will try to maximize the log likelihood for a binary classification problem (real = 1, fake=0)
- $x \sim p_{data}$  are real data samples so  $\mathbb{E}_{x \sim p_{data}} [\log D(x)]$  should be increased
- $\mathbb{E}_{z \sim p(z)}$  is used to generate fake  $x$  using  $G(z)$ , the discriminator should increase  $\mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$
- At equilibrium if reached the discriminator should not be able to tell which samples are fake



---

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator,  $k$ , is a hyperparameter. We used  $k = 1$ , the least expensive option, in our experiments.

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

**end for**

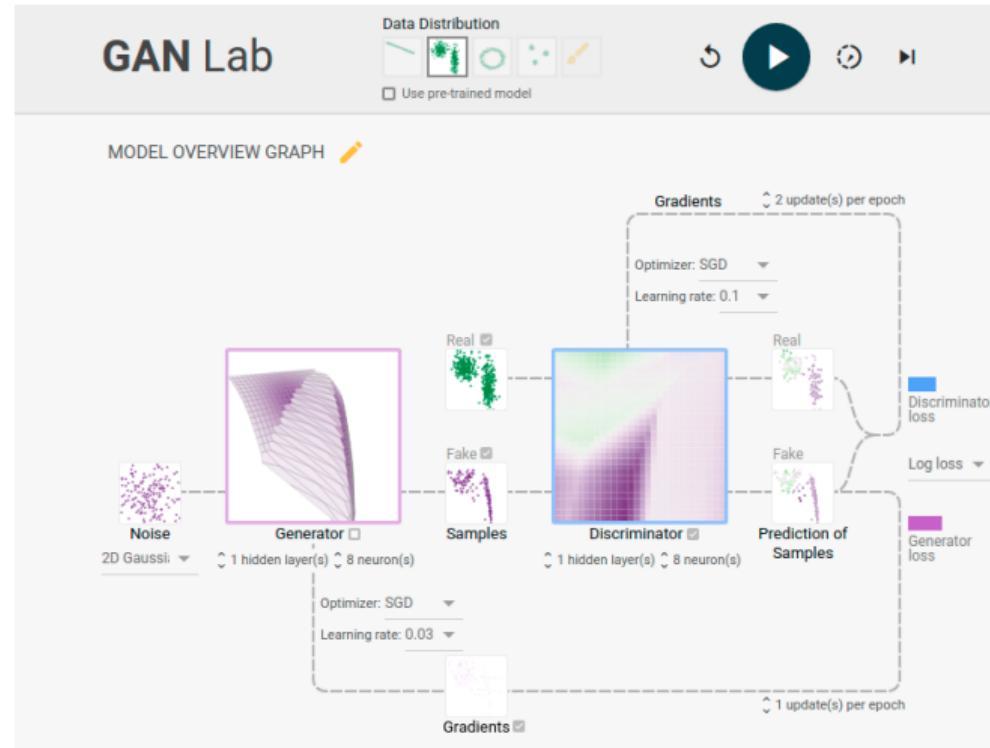
- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

---



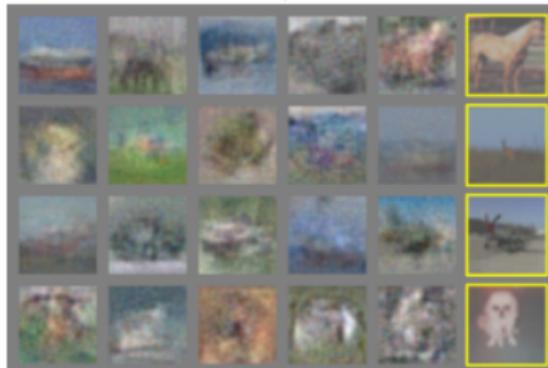
<https://poloclub.github.io/ganlab/>



a)



b)



c)



d)

[Goodfellow et al. 2014]

# Evaluation of GANs

---

- The goal is not only to fool the discriminator, but to obtain realistic samples
- We could do it by just generating a few samples that look real ignoring most of the probability space
  - This corresponds to a problem observed training GANs called **mode collapse**
- We should evaluate in some way how diverse are our samples and how well they cover the range of possibilities
- There is not a simple way to do this, specially in high dimensionality
- In some domains we could resort to human evaluation, but it is not an objective measure

- ⑤ In the image domain, we could use as reference an object classifier for measuring how recognizable and diverse is the content of the images generated
- ⑤ The **inception score** uses an InceptionV3 network trained on the ImageNet dataset (1000 classes)
- ⑤ The classification network will return for an image a distribution for the possible labels
- ⑤ If the images are good enough, the network will give a high probability to one of the classes, if not, the probability will be distributed along many classes
- ⑤ If the images are not diverse, the distribution of the predictions will have a low entropy

- We will classify after convergence a large sample of images obtained from the generator
- We will classify all the images with the inception network to obtain their class distribution  $p(y|x)$
- We can approximate the marginal  $p(y)$  of that distribution by averaging the distribution of all the samples

$$\hat{p}(y) = \frac{1}{N} \sum_i p(y|x_i)$$

- The Inception Score is the KL divergence between these two distributions

$$IS(x) = \exp\left(\frac{1}{N} \sum_i KL(p(y|x_i) || \hat{p}(y))\right) = \exp(H(y) - H(y|x))$$

- ⑤ Inception Score is not enough for evaluating image diversity (generating one image for each class gives a perfect score)
- ⑤ The Fréchet Inception Distance uses the features computed by the third pooling layer of the InceptionV3 architecture (2048 features)
- ⑤ The distance is computed from the mean and covariance of these features for the generated images and the images used to train the GAN

$$d^2((\mu, \Sigma), (\mu_w, \Sigma_w)) = \|\mu - \mu_w\|_2^2 + \text{tr}(\Sigma + \sigma_w - 2(\Sigma \Sigma_w)^{\frac{1}{2}})$$

## GANs problems

---

- ④ Mode collapse refers to a situation where the discriminator only generates a specific subset of the data
- ④ This problem can be theoretically justified by analysing the behaviour of the optimal discriminator (if we fully train the discriminator)
- ④ The Bayes optimal classifier for the original GAN objective function optimizes the Jensen-Shannon Divergence among the distribution of the generator and the distribution of the real data
- ④ Optimizing this measure encourages seeking only some modes of the data distribution if the generator is not expressive enough

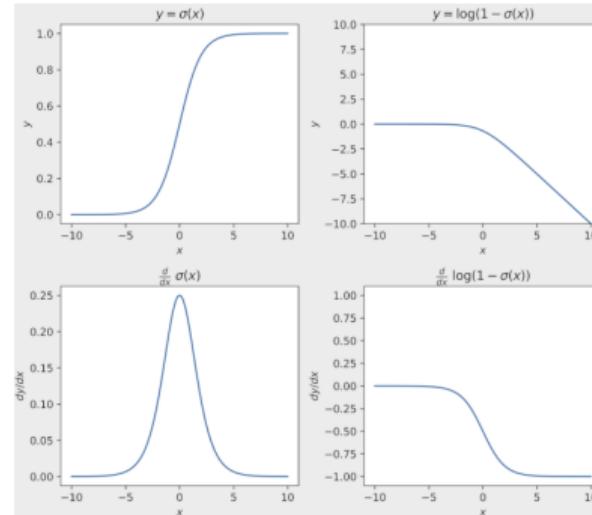
$$\begin{aligned}
 V(G, D) &= \mathbb{E}_{x \sim p_{data}} [\log D(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))] \\
 &= \int_x p_{data}(x) \log D(x) dx + \int_z p(z) \log(1 - D(G(z))) dz \\
 &= \int_x p_{data}(x) \log D(x) dx + \int_x p_g(x) \log(1 - D(x)) dx \\
 &= \int_x [p_{data}(x) \log D(x) + p_g(x) \log(1 - D(x))] dx
 \end{aligned}$$

$$\nabla_y [a \log y + b \log(1 - y)] = 0 \Rightarrow y^* = \frac{a}{a + b}$$

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

$$\begin{aligned} V(G, D^*) &= \mathbb{E}_{x \sim p_{data}} [\log D^*(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D^*(G(z)))] \\ &= \mathbb{E}_{x \sim p_{data}} [\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] + \mathbb{E}_{z \sim p(z)} [\log \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}] \\ &= -\log(4) + KL(p_{data} || \frac{p_{data} + p_g}{2}) + KL(p_g || \frac{p_{data} + p_g}{2}) \\ &= -\log(4) + JS(p_{data} || p_g) \end{aligned}$$

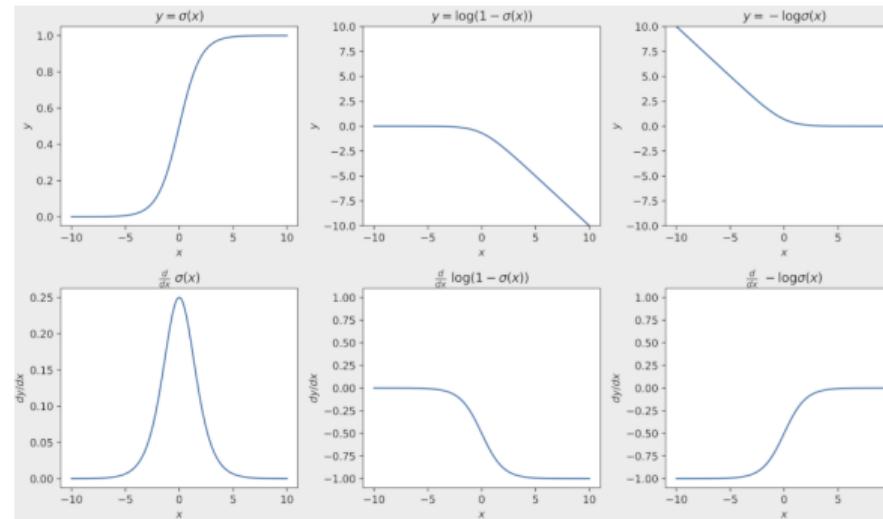
- The discriminator could get very confident about the fake samples
- We have a binary problem where the discriminating function is a sigmoid function
- This function gets very sharp when the discriminator is too good (saturates), so the derivative gets to basically 0 for most of the fake samples



- ④ This means that the information that can be passed to the discriminator about how to create better samples is not useful
- ④ One solution is to be careful about how many optimization steps are done each iteration (this is hard to do)
- ④ We will have to experiment as a hyper parameter how many iterations we perform for the discriminator before passing gradient information to the generator

- ⑤ An alternative solution that works in practice is to modify the objective function of the discriminator ,so it does not saturate
- ⑥ The function that is used for the discriminator is

$$\max_G \mathbb{E}_{z \sim p(z)} [\log(D(G(z)))]$$



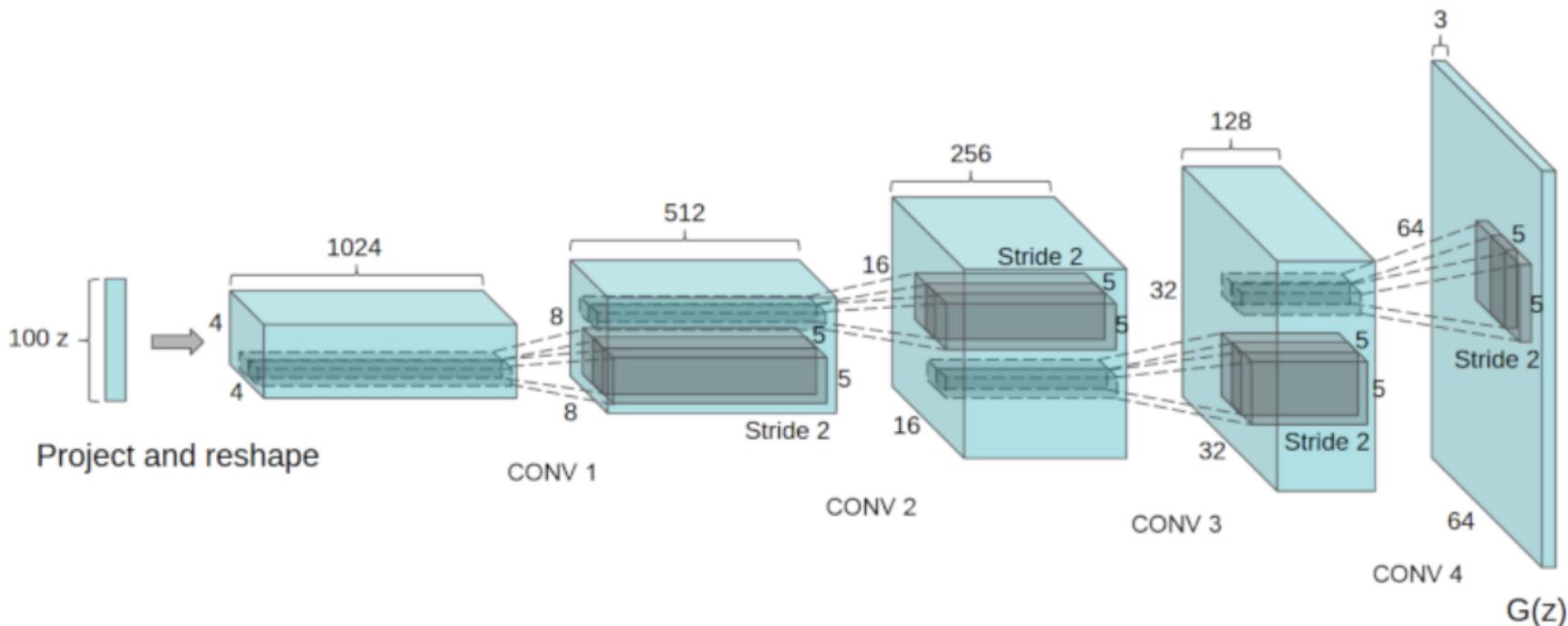
- ④ This reverses the function that is used putting all the gradient on the fake samples side
- ④ In practice, it is not a problem to have zero gradients on the side of the real samples
- ④ Now the game is not a zero-sum game given that the optimization of generator and discriminator are different

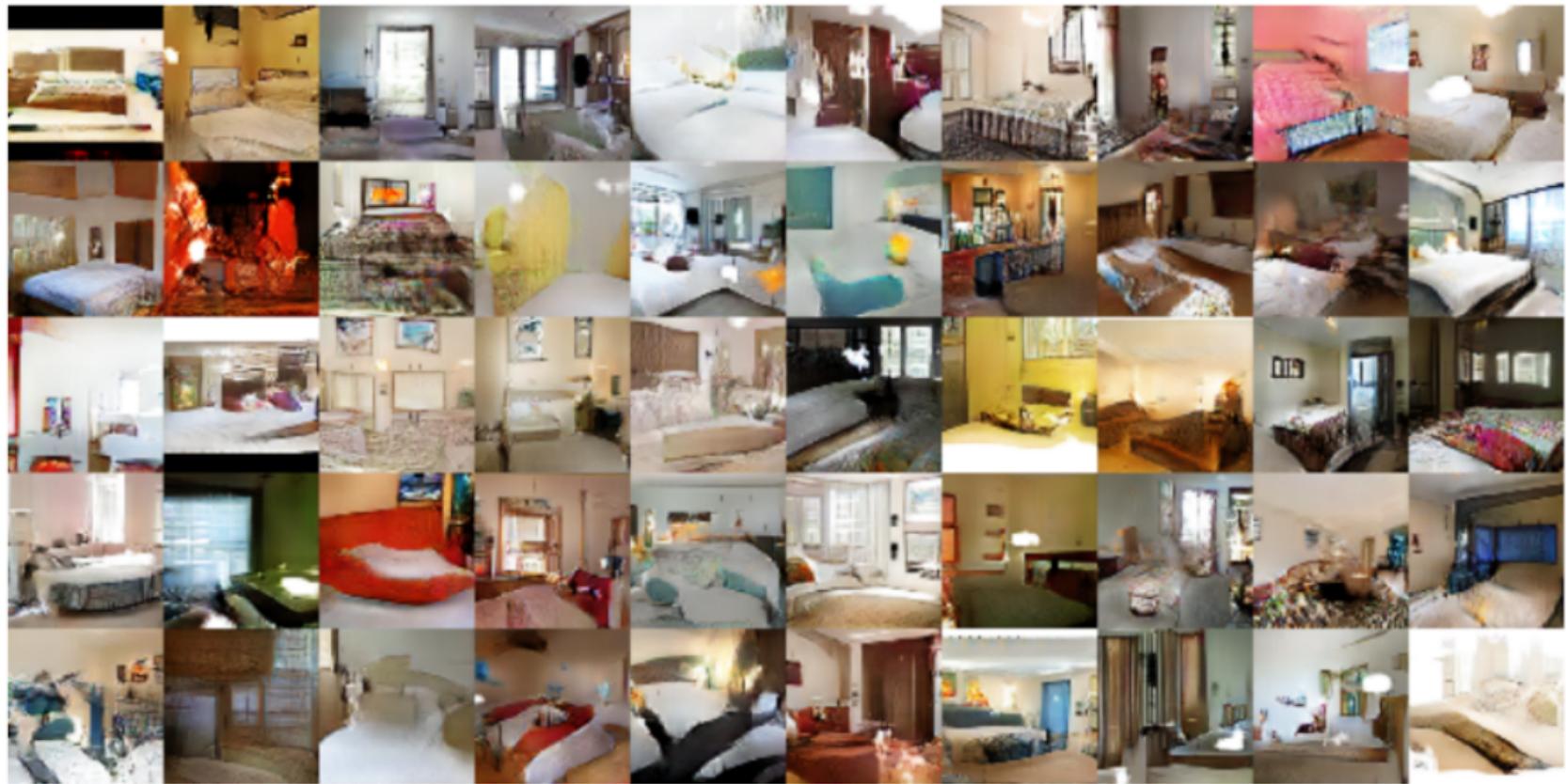
# Evolution of GANs

---

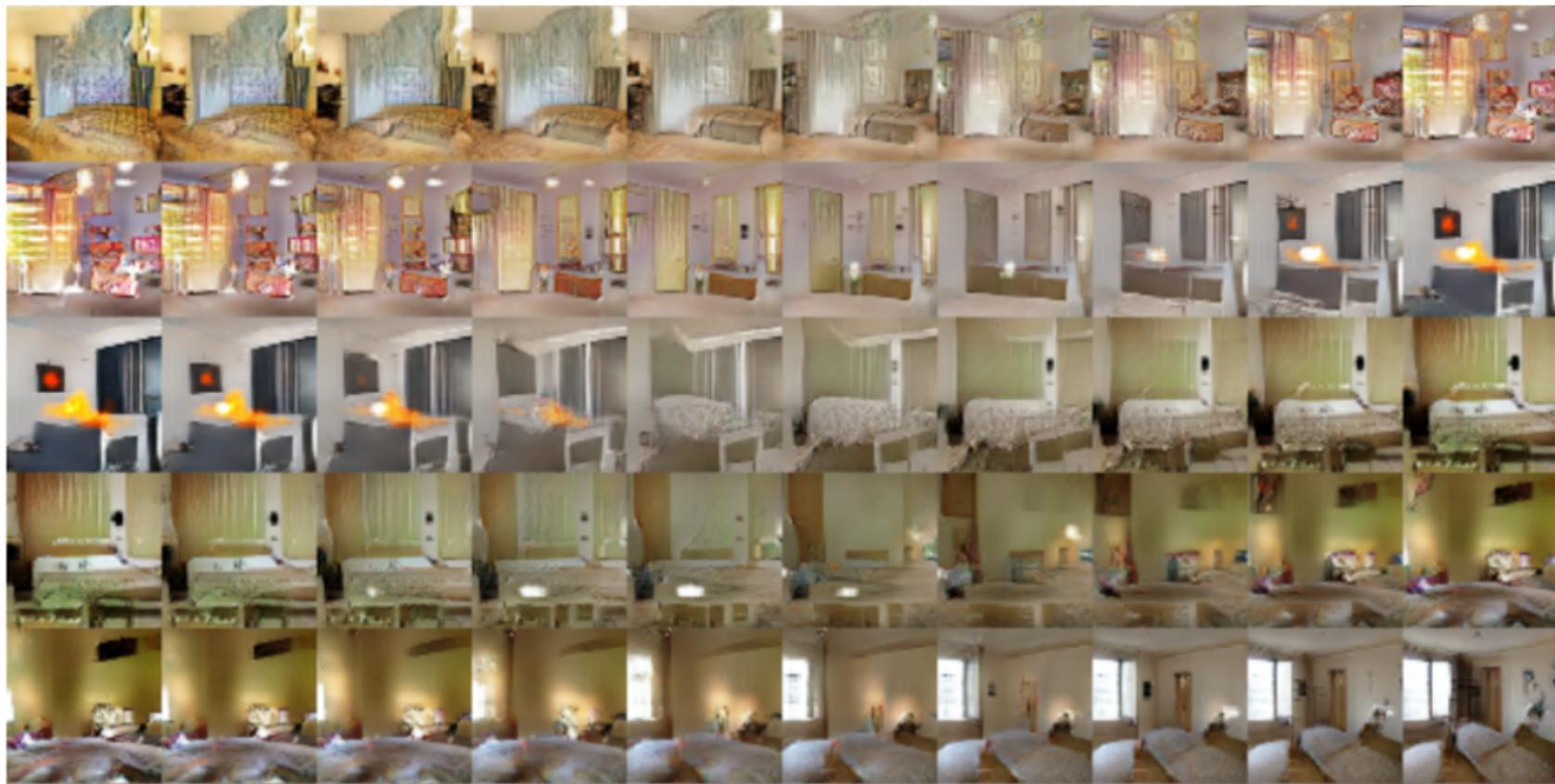
- ④ Since the definition of GANs a lot of progress has been made
- ④ Most of the original problems have been addressed by specific architectural choices for the generator and specific training decisions
- ④ This has generated a lot of insight about what works and does not work for training GANs
- ④ We will comment some successful architectures

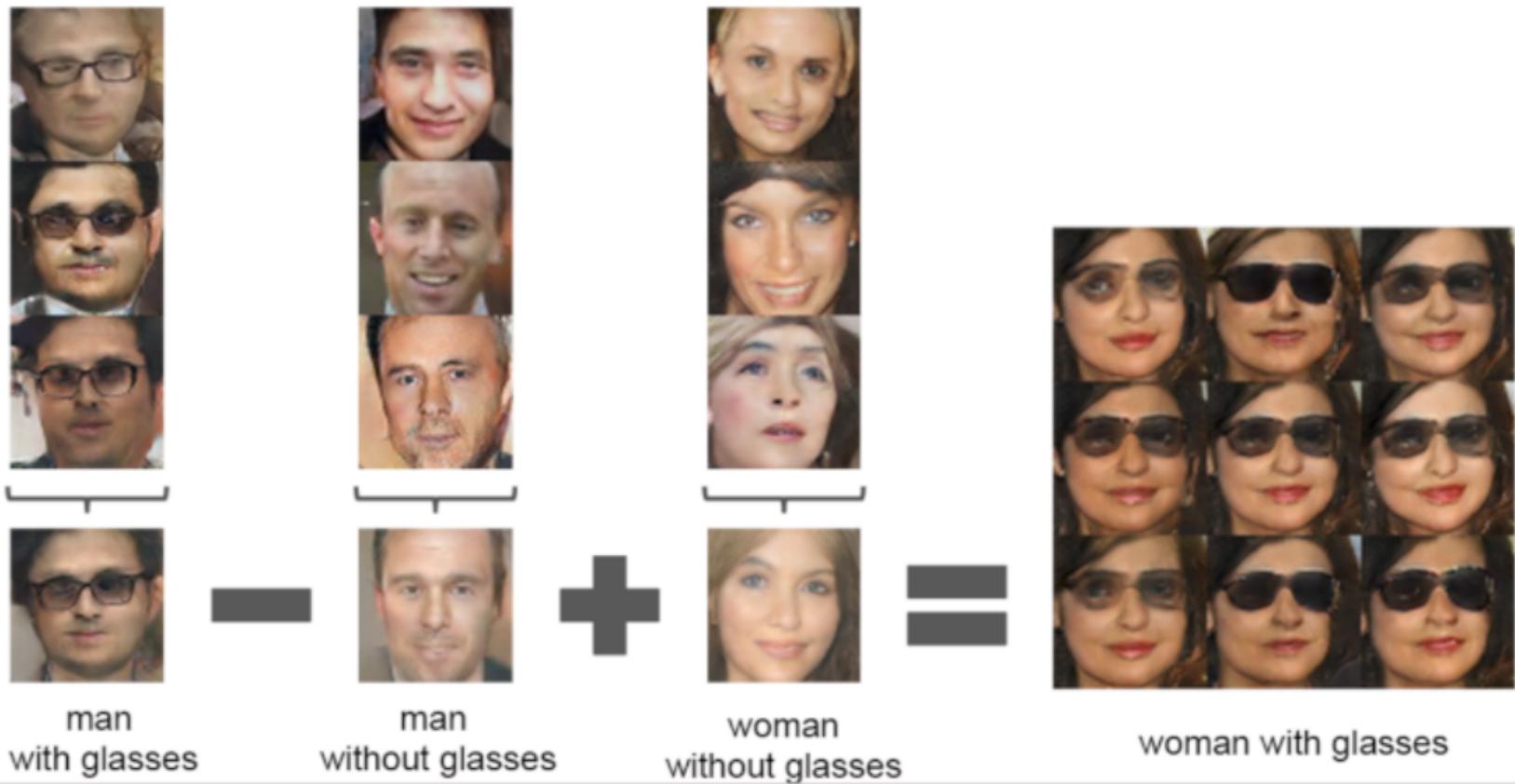
- ⑤ Experiments lead to the realization that many of the tricks of the trade of supervised deep learning with CNNs do not work for GANs
  - Do not use max or mean pooling
  - Upsample using transpose convolutions in the generator
  - Downsample with strided convolutions and average pooling
  - Non-Linearity: ReLU for generator, Leaky-ReLU for discriminator
  - Output Non-Linearity: tanh for Generator, sigmoid for discriminator
  - Batch Normalization used to prevent mode collapse
  - Batch Normalization is not applied at the output of  $G$ , and input of  $D$
  - Separated batches for real and fake data on the discriminator
- ⑥ The bad: Unstable training, lots of specific hyper-parameters to work, brittle architecture



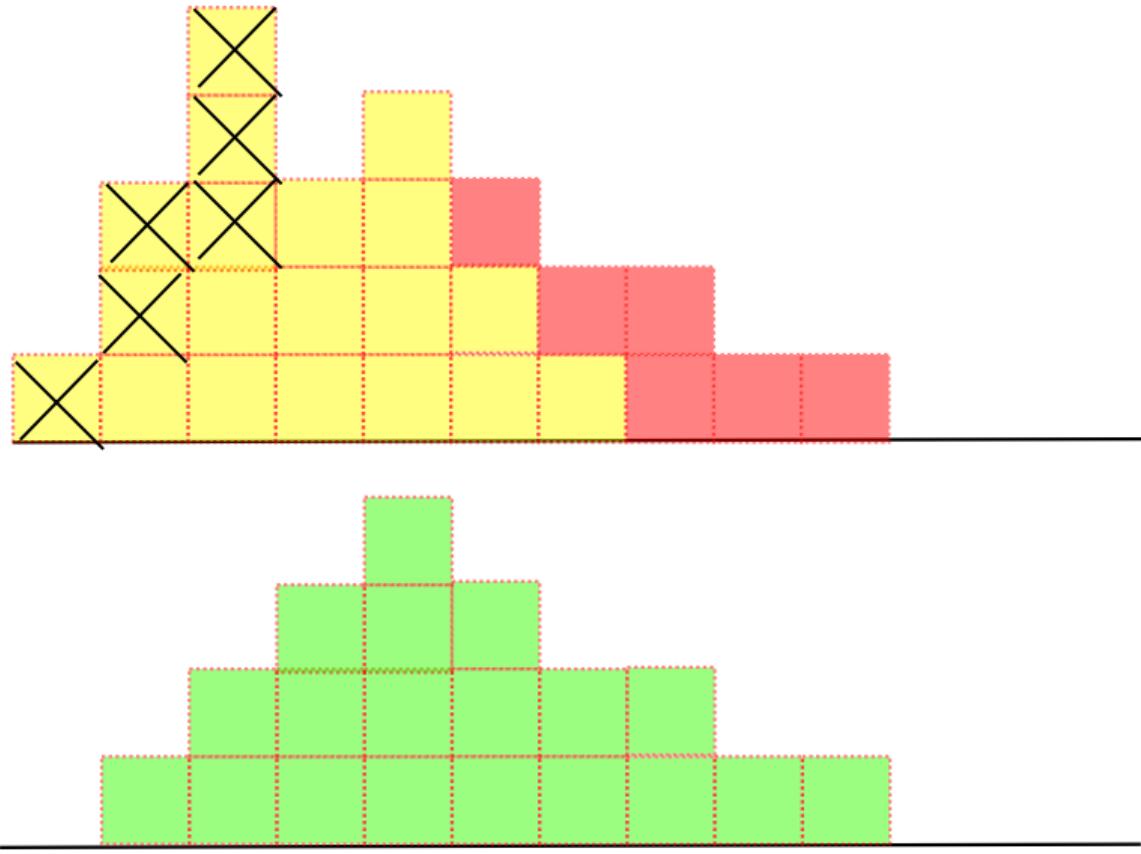


DCGAN smooth interpolation





- ④ There are alternative ways of measuring the similarity among the probability distribution of the real and generated data
- ④ The [Wasserstein distance](#) is inspired on the earth mover distance from Optimal Transport
- ④ Intuitively, in the domain of probability distributions this distance measures the cost of moving the probability mass of one distribution to obtain the other



- The goal is to obtain the optimal distance among two distributions, this can be formulated as:

$$W(P_a, P_b) = \inf_{\gamma \in \Pi(P_a, P_b)} \mathbb{E}_{(x,y) \sim \gamma} [|x - y|]$$

Where  $\Pi(P_a, P_b)$  represent the set of distributions  $\gamma(x, y)$  whose marginals are  $P_a$  and  $P_b$

- This is intractable to estimate (search over all possible joint distributions)
- We can reformulate this problem defining a dual version using the Kantorovich Rubinstein Duality as:

$$W(P_a, P_b) = \sup_{\|f_L\| \leq 1} \mathbb{E}_{x \sim P_a} [f(x)] - \mathbb{E}_{x \sim P_b} [f(x)]$$

- This means to search over 1-Lipschitz functions

- Ⓐ A function is K-Lipschitz if for distance functions  $d_x$  and  $d_y$

$$d_y(f(a), f(b)) \leq K \times d_x(a, b)$$

- Ⓐ It is also an intractable problem to search over K-Lipschitz functions
- Ⓐ We could search over a set parametrized functions  $f_W$  that is a lower bound of these functions (defined by a neural network) so:

$$\max_w \mathbb{E}_{x \sim P_a}[f_w(x)] - \mathbb{E}_{x \sim P_b}[f_w(x)] \leq \sup_{\|f_L\| \leq K} \mathbb{E}_{x \sim P_a}[f(x)] - \mathbb{E}_{x \sim P_b}[f(x)] = L \times W(P_a, P_b)$$

- Ⓐ This is a problem that we can optimize using backpropagation

---

**Algorithm 1** WGAN, our proposed algorithm. All experiments in the paper used the default values  $\alpha = 0.00005$ ,  $c = 0.01$ ,  $m = 64$ ,  $n_{\text{critic}} = 5$ .

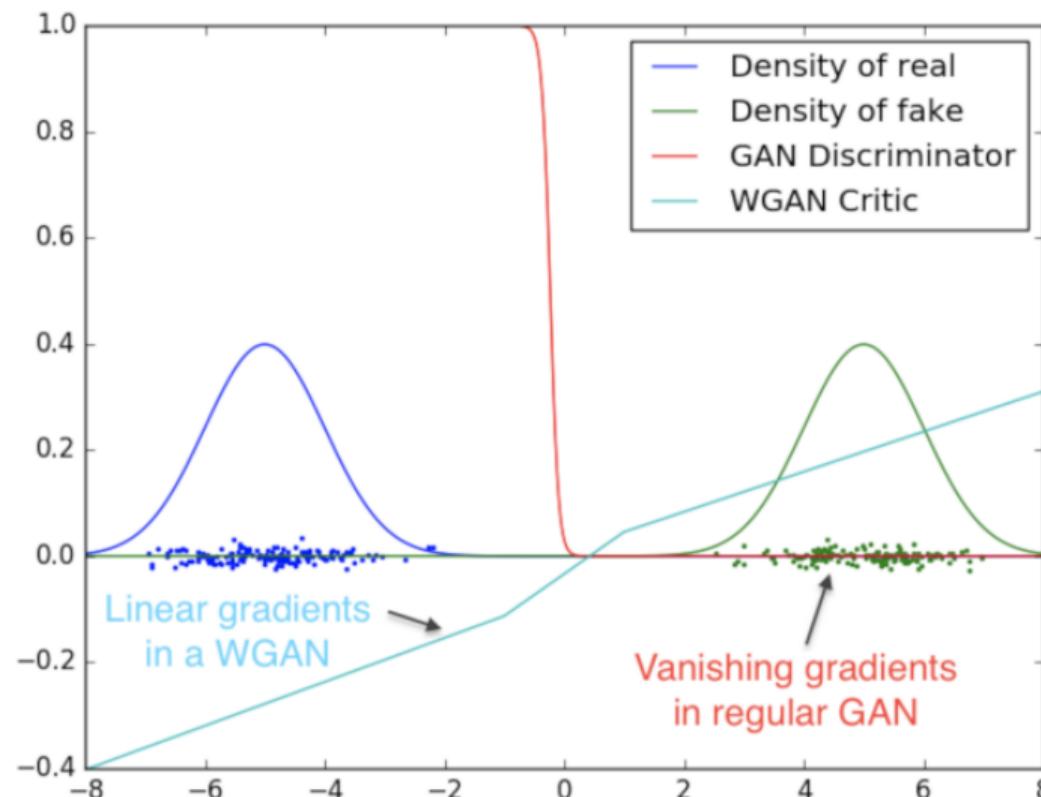
---

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

- 1: **while**  $\theta$  has not converged **do**
- 2:   **for**  $t = 0, \dots, n_{\text{critic}}$  **do**
- 3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
- 4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$
- 6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
- 7:      $w \leftarrow \text{clip}(w, -c, c)$
- 8:   **end for**
- 9:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
- 10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
- 11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
- 12: **end while**

---



- The optimization problem is changed to:

$$\min_G \max_D \mathbb{E}_{x \sim P_D}[D(x)] - \mathbb{E}_{\tilde{x} \sim P_G}[D(\tilde{x})]$$

- Wasserstein distance correlates with samples quality, there is no such thing with DCGAN
- WGAN is not too dependent on some architectural or training choices, more robust than DGAN (eg. batch normalization)
- Results of both GANs are similar
- The bad: To enforce the constraint of k-Lipschitz by clipping gradients is a hack

- ⑤ An improvement for the initial model of WGAN consists in to add a penalty term to the objective function
- ⑤ A 1-Lipschitz function must have gradients with norm at most 1 everywhere
- ⑤ We can try to force that adding to the objective function

$$\mathbb{E}_{x \sim P_D}[D(x)] - \mathbb{E}_{\tilde{x} \sim P_G}[D(\tilde{x})] + \lambda \mathbb{E}_{\tilde{x} \sim P_{\tilde{x}}}[(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1)^2]$$

- ⑤ This constraint is maintained sampling points in straight lines between samples from  $P_D$  and  $P_G$

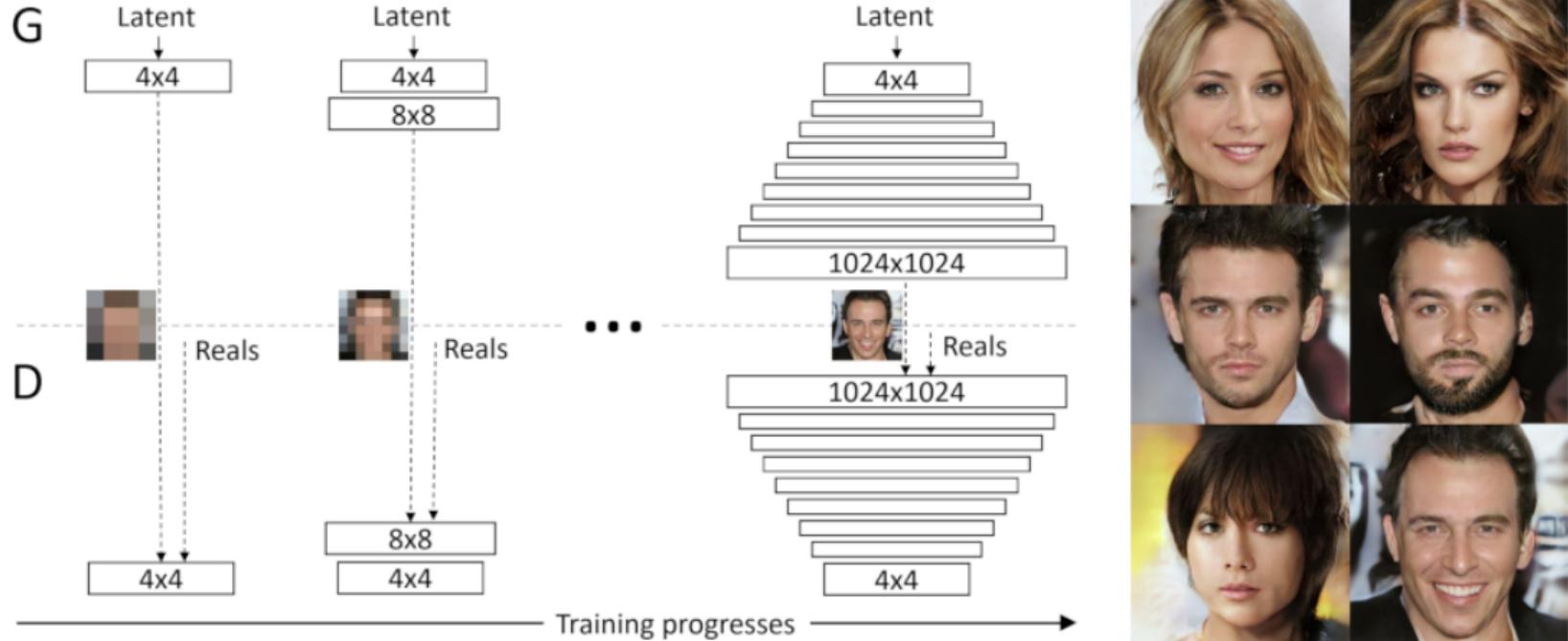
## ① Pros

- Very robust to architectural choices respect to the kind of linearity used, the depth of the network, number of convolutional filters
- Batch normalization is not needed
- Good quality samples according to the Inception Score
- Widely adopted by other approaches with great success

## ② Cons

- Gradient penalty could make convergence slow
- Gradient penalty computed from a heuristic distribution from samples

- ④ The resolution of images generated by GANs is usually small
- ④ With higher resolutions is easier for the discriminator to distinguish the fake samples
- ④ Gradients are not informative for guiding the generator
- ④ An alternative way of training GANs for higher resolution is to train a succession of GANs with increased resolutions
- ④ Each step a GAN to a specific resolution is trained, the parameters of the network are frozen, and a new layer is added for new training



## Progressive GANs: Progressive Samples



[Karras et al. 2017]

- ⑤ SN-GAN introduces spectral normalization in the layers to maintain the Lipschitz property of the functions computed by the discriminator
- ⑥ This introduces a more theoretically founded justification to the architecture
- ⑦ The spectral norm of a matrix is defined as:

$$\sigma(A) = \max_{\|h\|_2 < 1} \|Ah\|_2$$

- ⑧ The solution corresponds to the first singular value of A

- A neural network is a composition of functions where each layer has a matrix  $W$  that corresponds to its parameters
- A possible non-linearity can be applied to each layer
- This defines a network as a function:

$$f(x, \theta) = W^{L+1}a_L(W^L(a_{L-1}(\cdots a_1(W^1x) \cdots)))$$

- By definition Lipschitz norm is:

$$\|g\|_{Lip} = \sup_h \sigma(\nabla g(h))$$

- For a linear layer  $g(h) = Wh$  the norm will be  $\sigma(W)$

- ⑤ If the Lipschitz norm of the activation function is equal to 1 (for instance, it is for ReLU and Leaky ReLU, and some others are K-Lipschitz) we can use the inequality:

$$\|g_1 \circ g_2\|_{Lip} \leq \|g_1\|_{Lip} \cdot \|g_2\|_{Lip}$$

- ⑥ We can approximate the norm of the network as:

$$\begin{aligned}\|f\|_{Lip} &\leq \|W^{L+1}h_L\|_{Lip} \cdot \|a_L\|_{Lip} \cdot \|W^Lh_{L-1}\|_{Lip} \cdots \|a_1\|_{Lip} \cdot \|W^1h_0\|_{Lip} \\ &= \prod_{l=1}^{L+1} \|W^l h_{l-1}\|_{Lip} = \prod_{l=1}^{L+1} \sigma(W_l)\end{aligned}$$

- ⑦ This means that if for all the layers the matrix parameter  $W$  is normalized by the spectral norm  $\sigma(W)$  the functions that is computed by the network is bounded by 1

- ④ To compute the eigenvalues of a matrix is expensive if we want to obtain the exact value, involves computing SVD that is  $O(n^3)$
- ④ The first eigenvalue can be approximated using the [power iteration method](#)
- ④ This method starts with a random vector that is multiplied by the matrix and then normalized, this is repeated for a number of steps

$$v_{k+1} = \frac{Wv_k}{\|Wv_k\|}$$

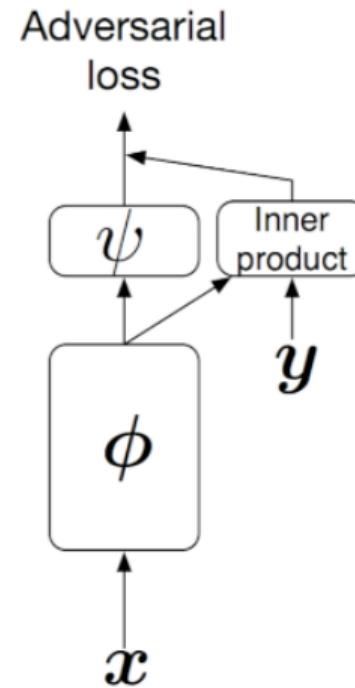
- ④ In practice, for this task, just one iteration of this procedure obtains an adequate approximation

- SN-GAN introduces a modification to the objective function of the discriminator

$$V(G, D) = \mathbb{E}_{x \sim p_{data}} [\min(0, D(x) - 1)] + \mathbb{E}_{z \sim p(z)} [\min(0, -1 - D(G(z)))]$$

- This corresponds to using the hinge loss (SVM classifier) instead of the cross entropy
- Empirically can be seen that the results are more aligned with FID (better quality for images)

- SN-GAN produced also good results generating with conditioning
- The idea is to introduce information about the class of the images as additional input
- SGAN appends a one hot encoding of the class in the discriminator

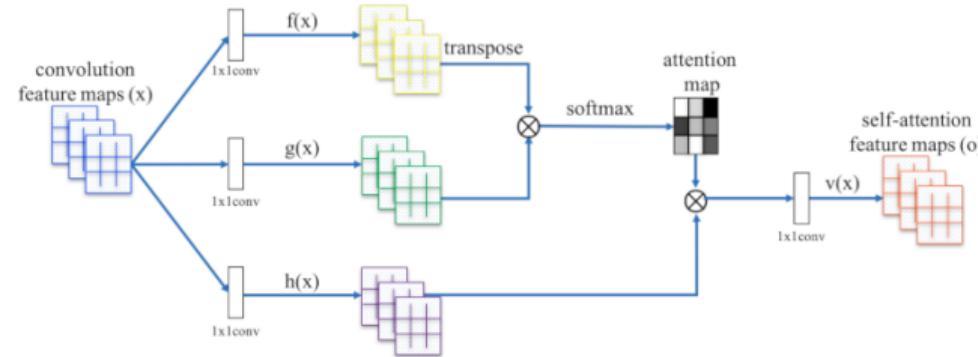


- Builds on previous results

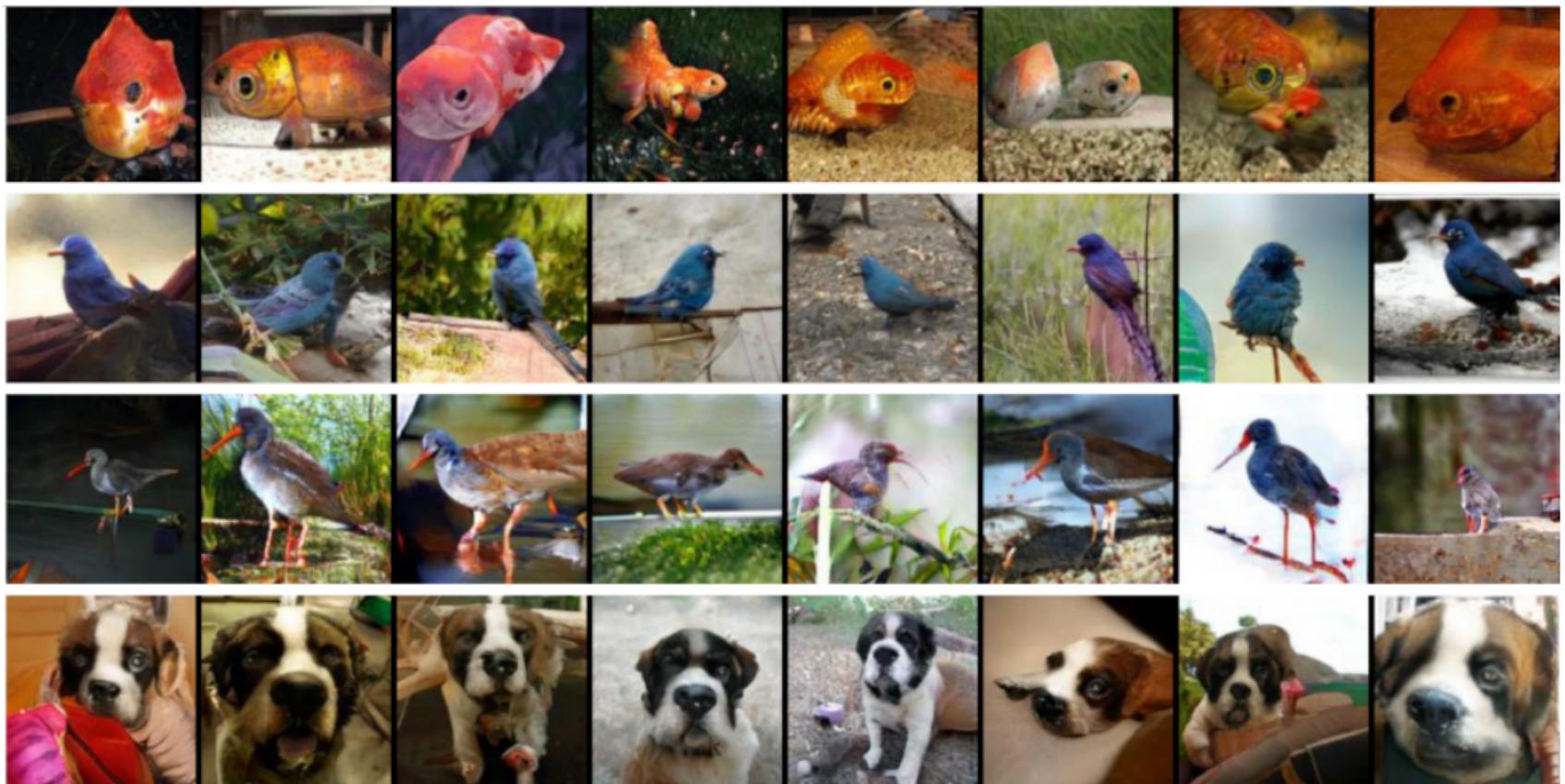
- Extend spectral normalization also to the generator
- Uses hinge loss as objective function

- Introduces self attention between convolutional layers for the generator and the discriminator

- Features computed by the convolutional filters are passed through  $1 \times 1$  convolutions and are used as query, key and values through the attention later



[Zhang et al. 2018]



- Self Attention+Spectral Normalization+Hinge Loss
- Increasing the batch size helps (probably because it adds more modes to each training step, with better gradients)
- Increasing the depth of the model also helps
- Regularization by penalizing non-orthogonal weight matrices with a twist

$$\|W^\top W - I\|_F^2 \Rightarrow \|W^\top W \odot (1 - I)I\|_F^2$$

- **Truncation Trick:**
  - During training the samples for the generator are from a normal distribution with variance  $\sigma^2$
  - During test the values over a threshold  $[-c, c]$  are truncated (resampled to a value inside the limit)
  - This improves quality at the expense of variety

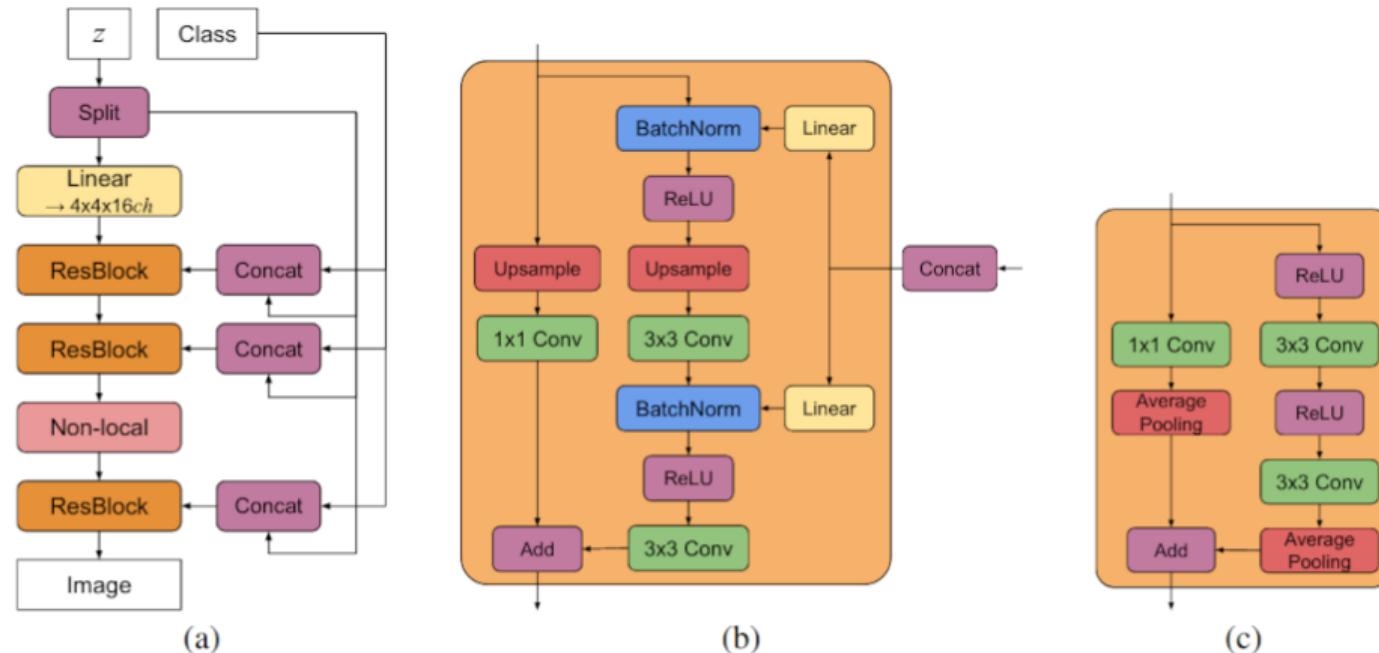
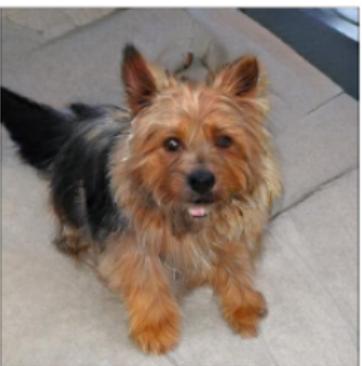
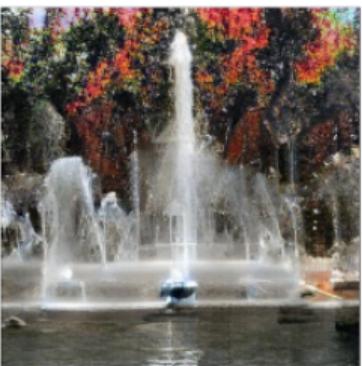


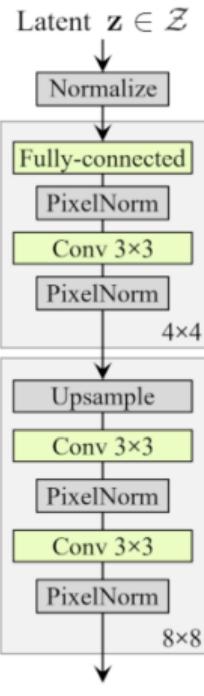
Figure 15: (a) A typical architectural layout for BigGAN’s **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN’s **G**. (c) A Residual Block (*ResBlock down*) in BigGAN’s **D**.

$z \in \mathbb{R}^{160} \sim \mathcal{N}(0, I)$	RGB image $x \in \mathbb{R}^{512 \times 512 \times 3}$
Embed( $y$ ) $\in \mathbb{R}^{128}$	
Linear $(20 + 128) \rightarrow 4 \times 4 \times 16ch$	ResBlock down $ch \rightarrow ch$
ResBlock up $16ch \rightarrow 16ch$	ResBlock down $ch \rightarrow 2ch$
ResBlock up $16ch \rightarrow 8ch$	ResBlock down $2ch \rightarrow 4ch$
ResBlock up $8ch \rightarrow 8ch$	Non-Local Block $(64 \times 64)$
ResBlock up $8ch \rightarrow 4ch$	ResBlock down $4ch \rightarrow 8ch$
Non-Local Block $(64 \times 64)$	ResBlock down $8ch \rightarrow 8ch$
ResBlock up $4ch \rightarrow 2ch$	ResBlock down $8ch \rightarrow 16ch$
ResBlock up $2ch \rightarrow ch$	ResBlock down $16ch \rightarrow 16ch$
ResBlock up $ch \rightarrow ch$	ResBlock $16ch \rightarrow 16ch$
BN, ReLU, $3 \times 3$ Conv $ch \rightarrow 3$	ReLU, Global sum pooling
Tanh	Embed( $y$ ) $\cdot h + (\text{linear} \rightarrow 1)$
(a) Generator	(b) Discriminator

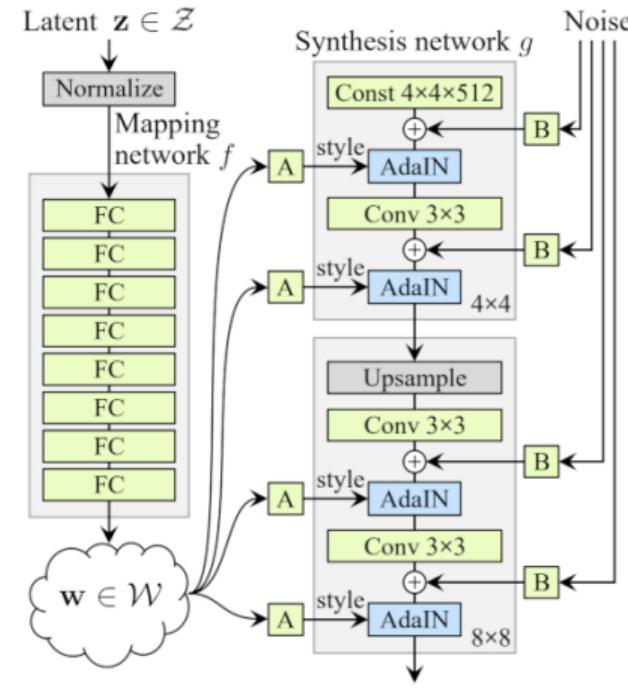


[Brock et al. 2019]

- StyleGAN departs from the usual architecture for the generator introducing a control of how the latent code modifies the image
- The generator starts with a constant learned code
- The random samples are passed through a fully connected network and feed to the generator at different resolutions (style vectors)
- Noise is added at different resolutions to improve quality (reduces blurred details)
- **Adaptive Instance Normalization:** an affine transformation that normalizes the examples according to the style vector



(a) Traditional

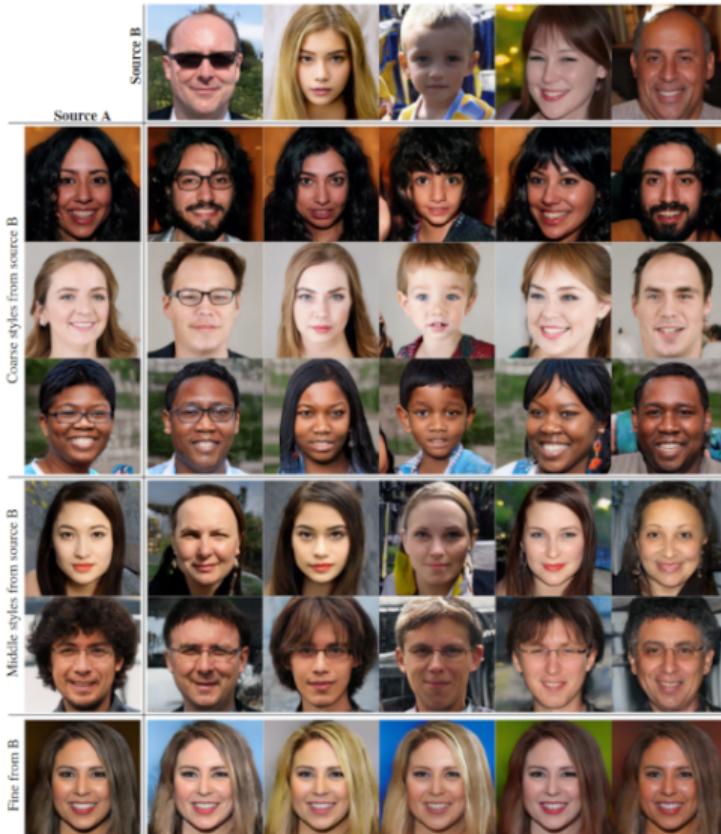


(b) Style-based generator



[Karras et al. 2019]

# StyleGAN: Mixing different latent vectors at different levels

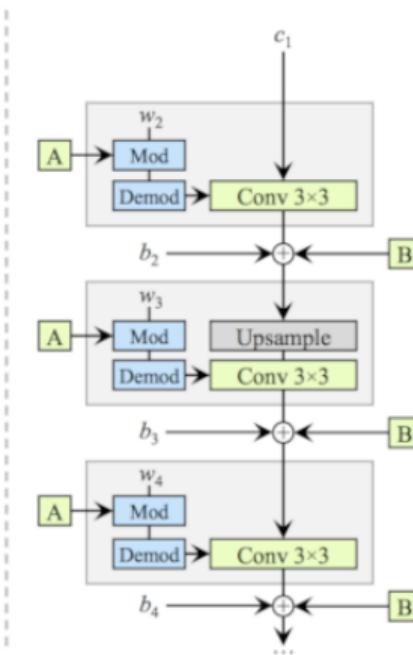
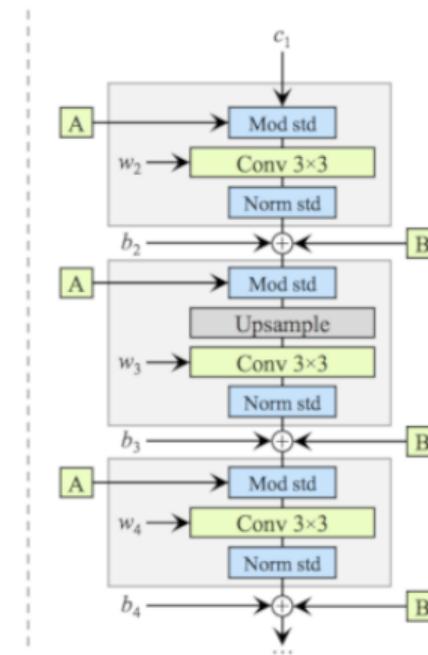
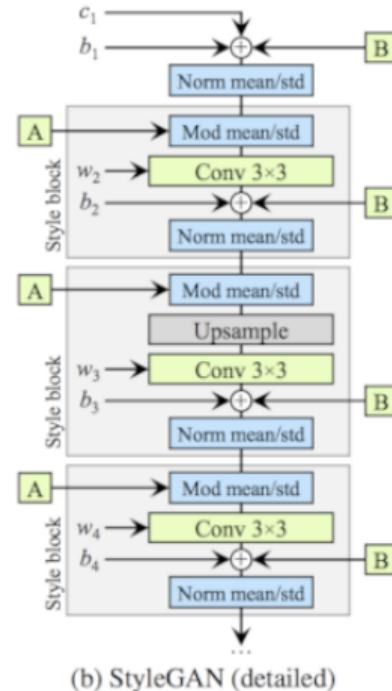
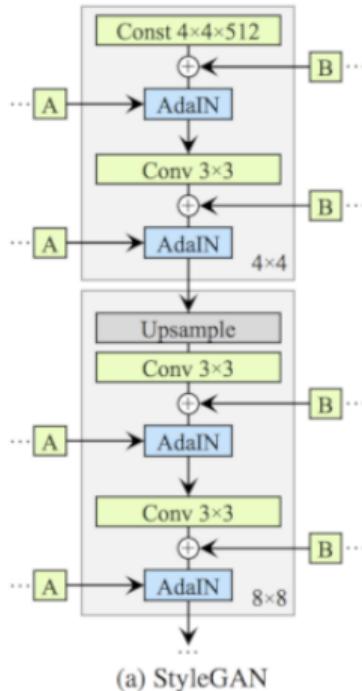


[Karras et al. 2019]

- StyleGAN sometimes generates images with artefacts that can be used to distinguish fake images
- These defects appear because of normalizations and how the image is grown at different scales



[Karras et al. 2020]



[Karras et al. 2020]



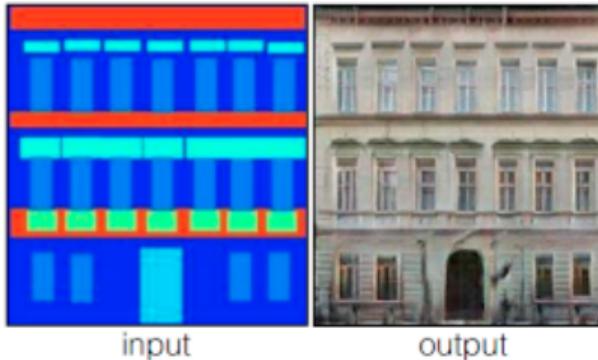
[Karras et al. 2020]

# Conditional GANs

---

- Some GANs models we have talked about allow using information about the labels to generate specific classes
- Conditioning does not need to be limited to labels
- For instance in the Image-to-Image problem we want to obtain a new image from another by translating the pixels from the source to the target image
- We can use conditional GANs to create a generator that gets as input the source and obtain the target

Labels to Facade



input

output

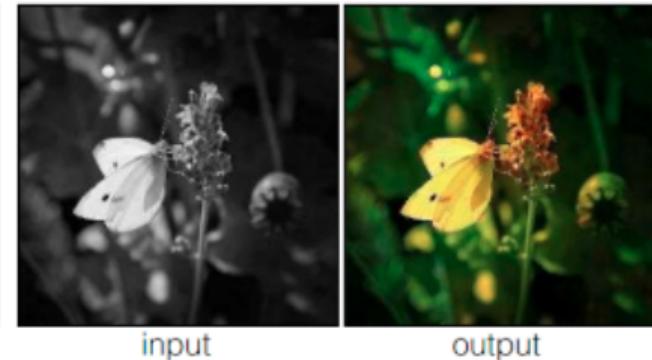
Day to Night



input

output

BW to Color



input

output

Edges to Photo

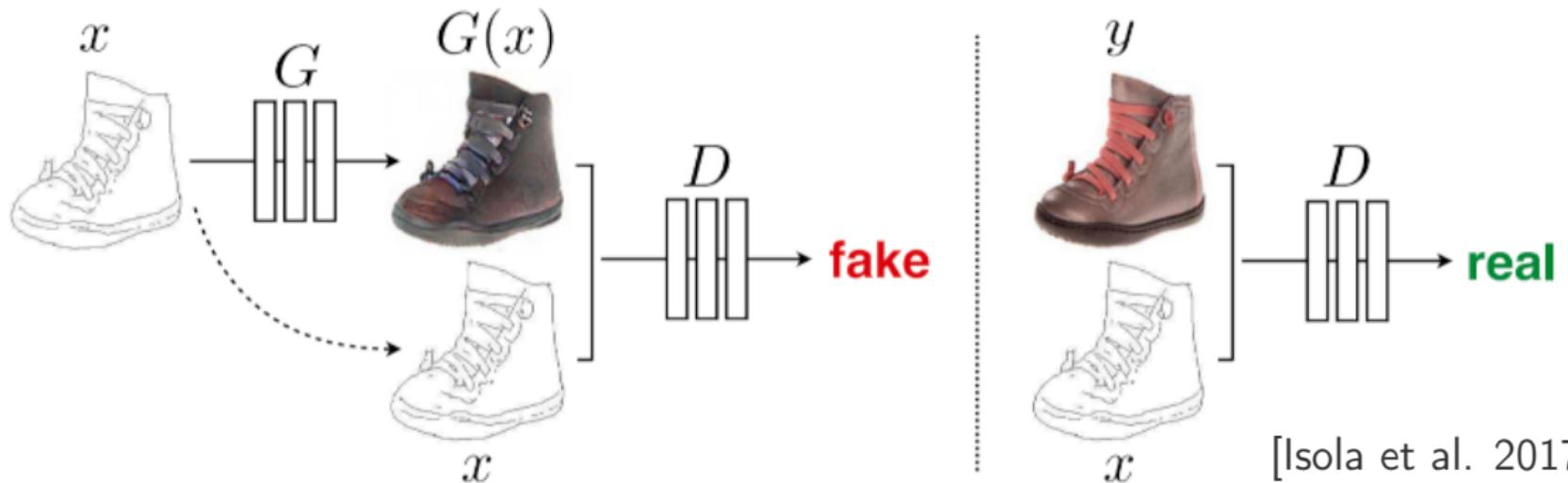


input

output

[Isola et al. 2017]

- For computing the generator now we need a random vector  $z$  and a conditioning image  $x$  at the input
- The generator will obtain a fake image that will be processed by the discriminator
- The discriminator need to be able to distinguish between real and fake images
- To use directly the GANs objective function is not enough to obtain good images
- For instance, if we want colorized images from grey scale images, the generator could just generate random good images without matching the content



[Isola et al. 2017]

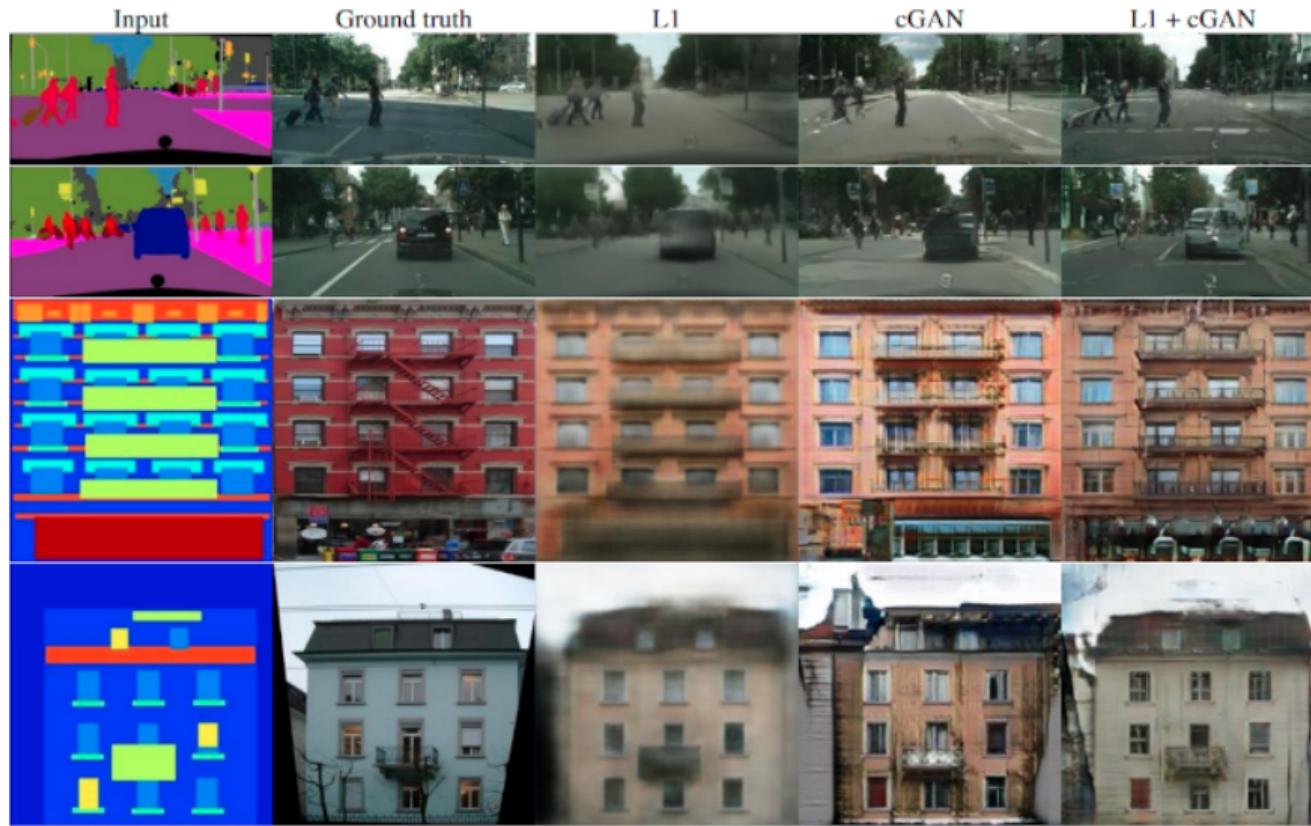
- ① The objective function that is optimized is:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))]$$

- ② Now we have real and fake image pairs
- ③ Additionally, the generator needs to be as close as possible to the conditioning image
- ④ This can be forced by measuring the distance among the two images and penalizing the objective function
- ⑤ The  $L1$  distance is used as penalization term ( $L2$  generates more blurriness)

$$\arg \max_G \min_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

- ④  $L1$  loss allows capturing low frequency characteristics of the images, but it is still difficult to obtain non blurred images (high frequencies)
- ④ The discriminator is simplified to detect these difference between true and fake images
- ④ This can be obtained by a discriminator that works with  $n \times N$  patches of the images instead of full images
- ④ This has the additional advantage of reducing the number of parameters in the discriminator



[Isola et al. 2017]

## #edges2cats



Christopher Hesse trained our model on converting edge maps to photos of cats, and included this in his [interactive demo](#). Apparently, this is what the Internet wanted most, and #edges2cats briefly [went viral](#). The above cats were designed by Vitaly Vidmirov (@vvvid).

## Interactive Anime



Bertrand Gondouin trained our method to translate sketches—Pokemon, resulting in an interactive drawing tool.

## Alternative Face



Mario Klingemann used our code to translate the appearance of French singer Francoise Hardy onto Kellyanne Conway's infamous "alternative facts" interview. Interesting articles about it can be read [here](#) and [here](#).

## Person-to-Person



Brannon Dorsey recorded himself mimicking frames from a video of Ray Kurzweil giving a talk. He then used this data to train a Dorsey→Kurzweil translator, allowing him to become a kind of puppeteer in control of Kurzweil's appearance.

## Background masking



Kaihu Chen performed a number of interesting experiments using our method, including getting it to mask out the background of a portrait as shown above.

## Color palette completion

Input	Generated	Ground truth

Colormind adapted our code to predict a complete 5-color palette given a subset of the palette as input. This application stretches the definition of what counts as "image-to-image translation" in an exciting way: if you can visualize your input/output data as images, then image-to-image methods are applicable! (not that this is necessarily the best choice of representation, just one to think about.)

- ④ To obtain paired images for image to image translation is difficult
- ④ A possible solution is to learn two mappings ( $G, F$ ) that can translate between two sets of images
- ④ Maintaining the consistency between the results of the mapping a generator from one set to another can be trained (cycle consistency)

$$\text{Image}_x \rightarrow G(\text{Image}_x) \rightarrow F(G(\text{Image}_x)) \rightarrow \text{Image}_x$$

- ① Two adversarial networks are trained that correspond to the two mappings
- ② Each adversarial network optimize an adversarial loss, a generator that obtains images from domain  $X$  to domain  $Y$  and a discriminator that learns to distinguish images in domain  $Y$  and vice versa

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_y[\log D_Y(y)] + \mathbb{E}_x[\log(1 - D_Y(G(x)))]$$

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_x[\log D_X(x)] + \mathbb{E}_y[\log(1 - D_X(G(y)))]$$

- ⑤ In order to force the mapping from one domain to the another and back a loss based on the  $L1$  distance is added

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_x[||F(G(x)) - y||_1 + \mathbb{E}_x[||G(F(y)) - x||_1]$$

- ⑥ The complete optimization goal is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F)$$

- ⑦ This can be interpreted as training two autoencoder at the same time that obtain that map the input through an intermediate representation that corresponds to a second domain

Monet  $\curvearrowright$  Photos



Monet  $\rightarrow$  photo

Zebras  $\curvearrowright$  Horses



zebra  $\rightarrow$  horse

Summer  $\curvearrowright$  Winter



summer  $\rightarrow$  winter



photo  $\rightarrow$  Monet



horse  $\rightarrow$  zebra



winter  $\rightarrow$  summer



Photograph



Monet



Van Gogh



Cezanne



Ukiyo-e

# Feature Learning

---

- Many domains decompose into a set of features that describe how the result varies
  - E.g. position, angle, illumination, color...
- We could use GANs to recover these variables as a latent representation of the result
- The idea is to add a set of random variables as input in the generator, so they can be learned from the process
- So the input of the GAN is a source of noise  $z$ , and a set of structured random variables  $c$  that represent the structure of the domain (discrete or continuous)

- ⑤ We need to force the process to use the latent code as part of the optimization obtaining some semantic information
- ⑥ We can use mutual information to define the relationship between the generated image, and the latent code

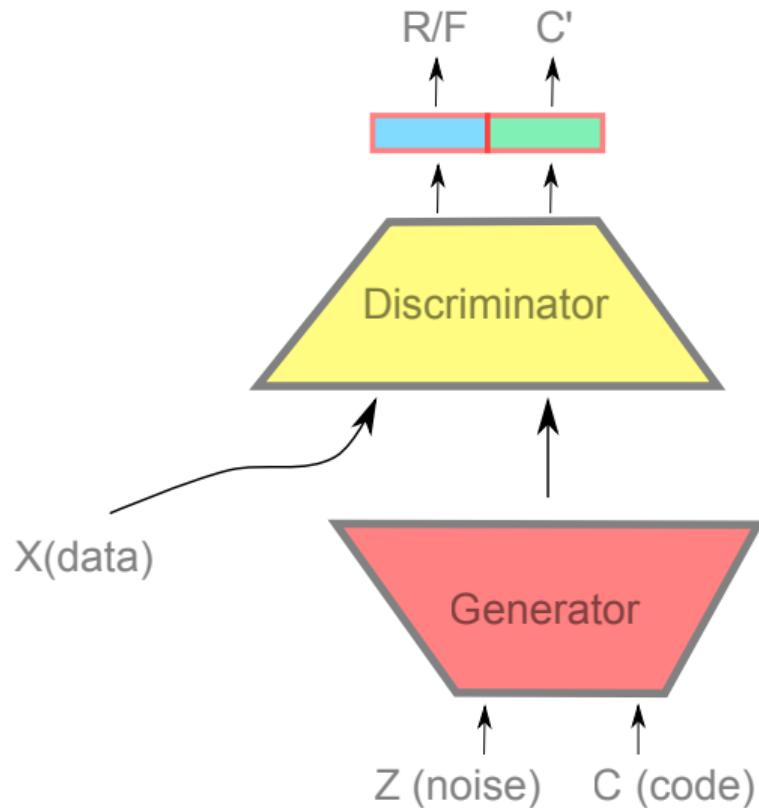
$$I(X; Y) = H(Y) - H(Y|X)$$

- ⑦ The goal is to obtain a dependence among the generated image and the latent code so it captures the variability of the data

$$I(c; G(z, c)) = H(c) - H(c|G(z, c)))$$

- ⑤ To obtain this value is intractable, so we can use Variational Inference using another probability distribution as approximation
- ⑥ This auxiliary distribution can be represented by a neural network that can share parameters with the discriminator
- ⑦ The optimization goal will include the GAN loss and the approximation of the mutual information

$$\min_{G,Q} \max_D V_{infoGAN}(D, C, Q) = V(D, G) - \lambda I(G, Q)$$





(a) Azimuth (pose)

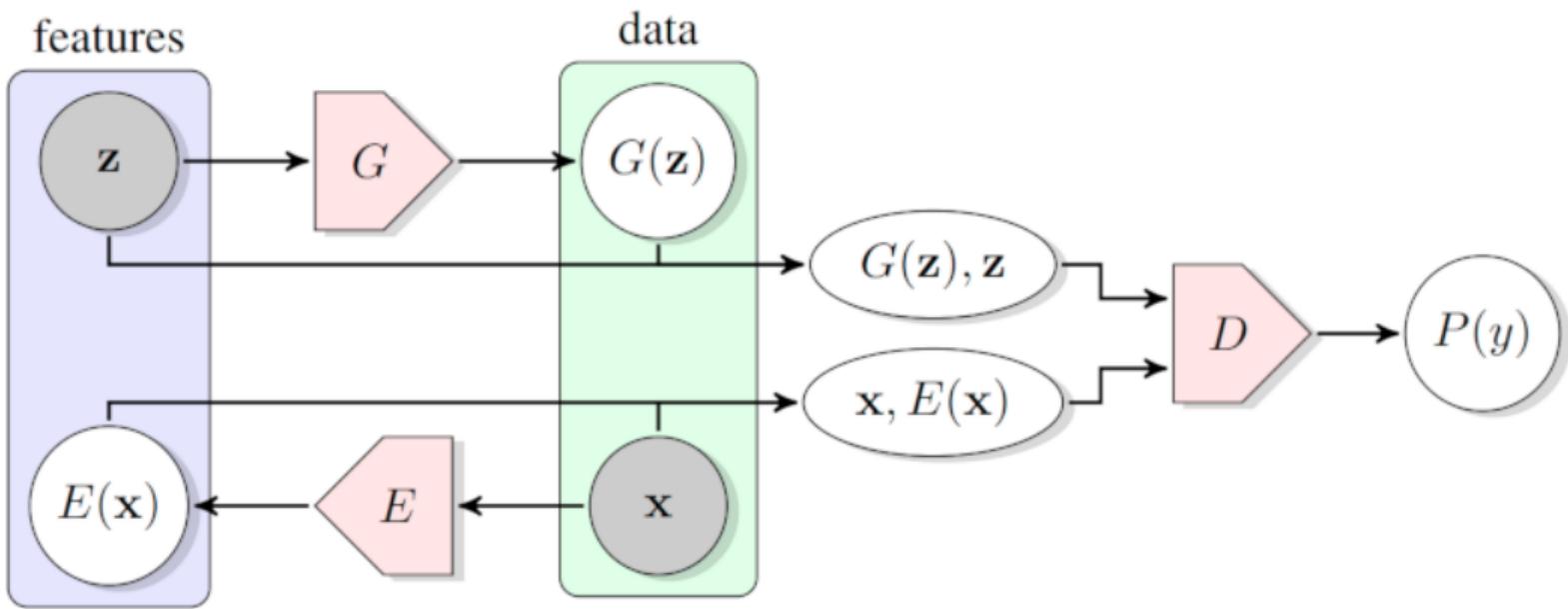
(b) Elevation



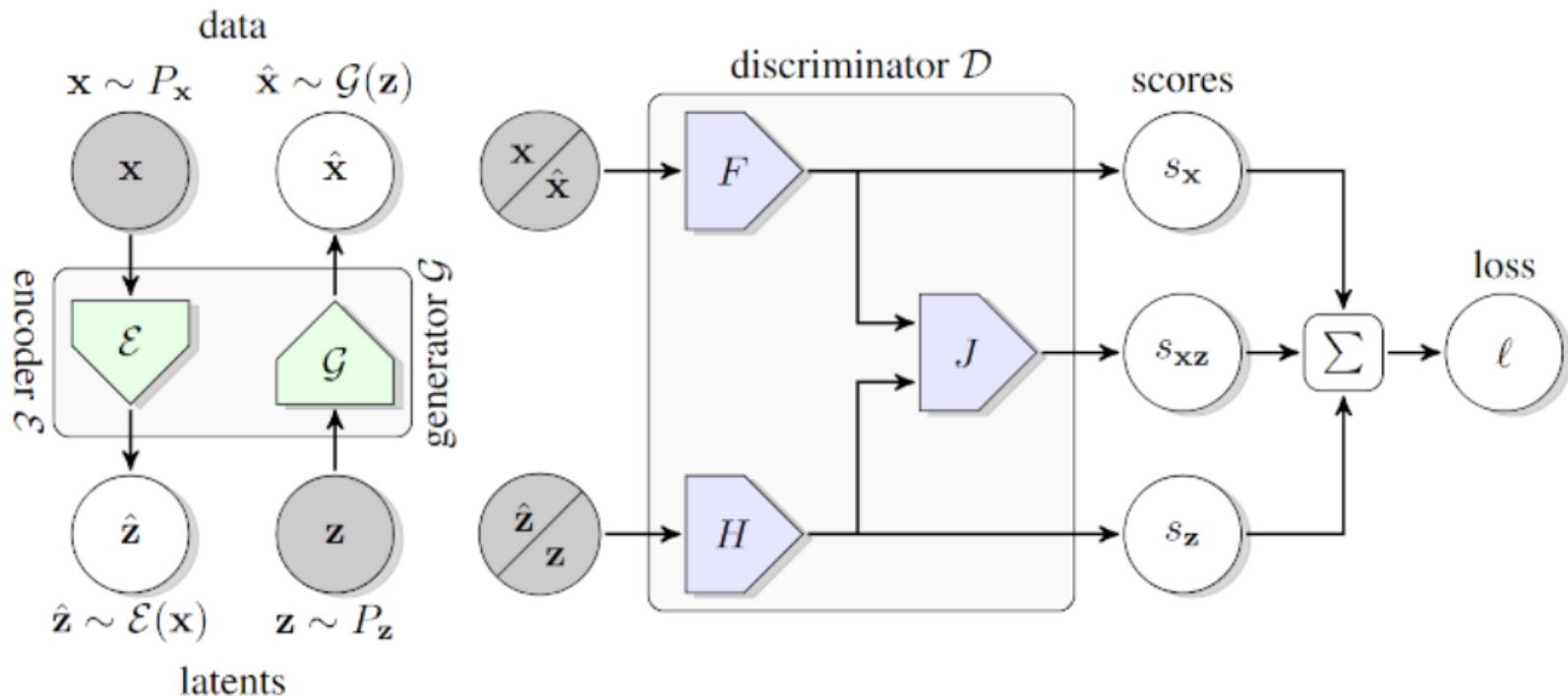
(c) Lighting

(d) Wide or Narrow

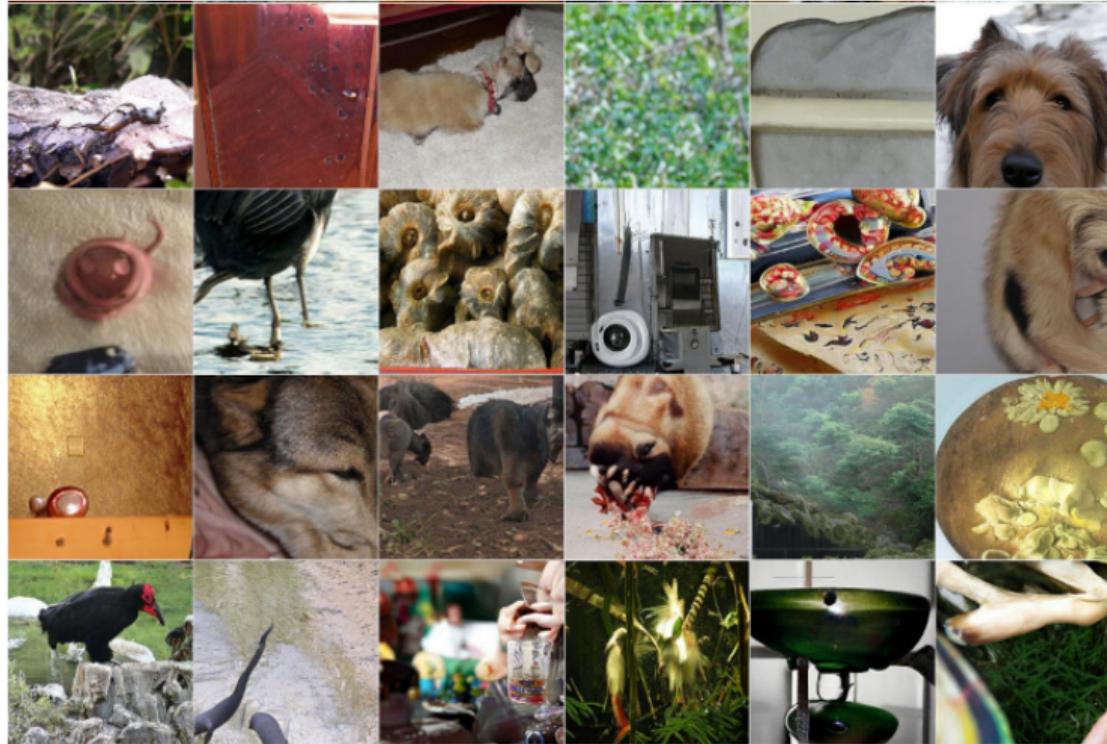
- Bidirectional GANs add an encoder to GANs, so a latent representation can be learned
- The encoder obtains a latent representation of the data and is optimized in the same process
- The discriminator receives as input the latent codes, and the fake and real images
- The objective function is modified, so we optimize the cross entropy of pairs (learned code, real image), and (noise, fake image)
- Learned features can be used for other tasks



[Donahue et al. 2016]



[Donahue et al. 2019]



[Donahue et al. 2019]



[Donahue et al. 2019]

# UDL: Self Supervised Learning

---

Javier Béjar

URL - 2021 Spring Semester

CS - MAI

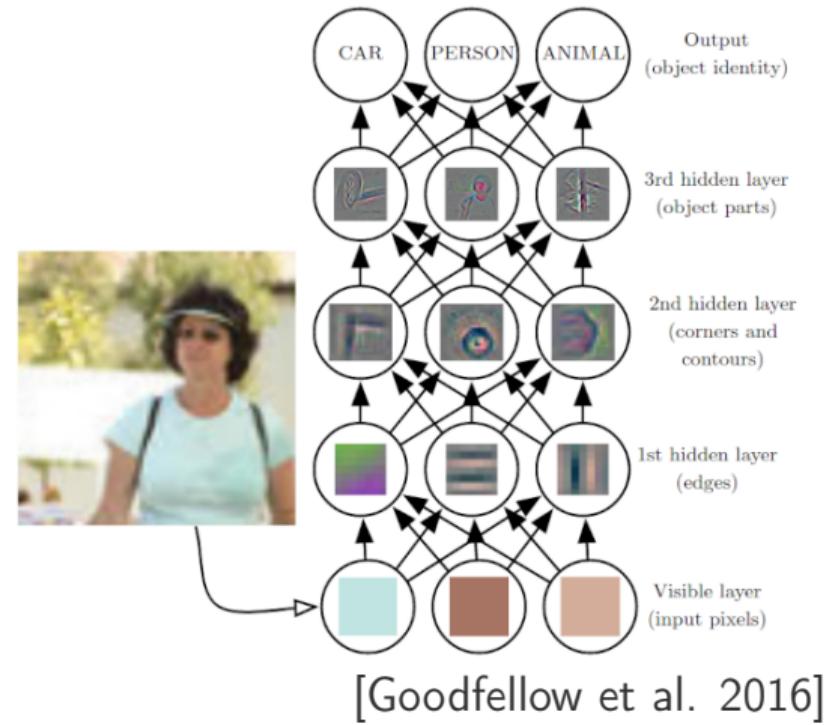


# Introduction

---

- ① To learn features from data in an unsupervised way that can be used on different downstream tasks (general and semantically rich features)
- ② What tasks can be used to learn features from unsupervised data
- ③ How supervised tasks can be improved with a pretrained network
  - Data efficiency (less labeled data)
  - Performance (as good as features from labeled data)

- It is easier to learn a task from examples if they are represented by adequate features
- Deep Learning is about learning good representations from the data
  - Representations are obtained at different levels of granularity
  - The hierarchy of features is refined each level to obtain more semantically relevant features



- ⑤ It is very expensive to label datasets for a specific task (especially for deep learning)
  - Define categories, decide label criteria, create applications for labelling, hire people to labels...
- ⑤ Quality labeled data can be very expensive
  - Anyone can label a cat, but not a tumor
- ⑤ Very large unlabelled datasets can be collected (on the wild or from data repositories)
- ⑤ Supervised learning is not the natural way to learn for many tasks

- ④ Self Supervised Learning is a field of Unsupervised Learning where data provides the supervision
- ④ Usually the task involves **hiding a part of the example** so a neural network predicts it from the remaining information
- ④ The difference among the methods is what pretext loss or proxy tasks is used for learning
- ④ Depending on the quality of the task good features can be learned without any supervision

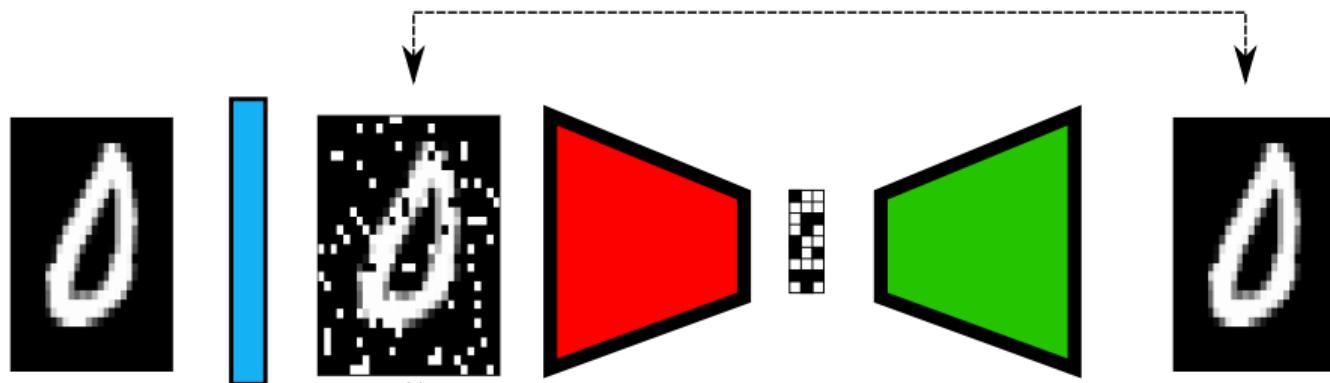
- ④ The success of supervised deep learning relies on good pre-trained features useful for relevant tasks after tuning (object detection, segmentation...)
- ④ This depends heavily on sufficient and quality labelled data
- ④ Goal of Self Supervised Learning:
  - Learn equally good (or better) features without any supervision
  - Obtain good models that can be obtained (tuned) with less labelled data
  - Obtain more general features because tasks are not focused on specific problems

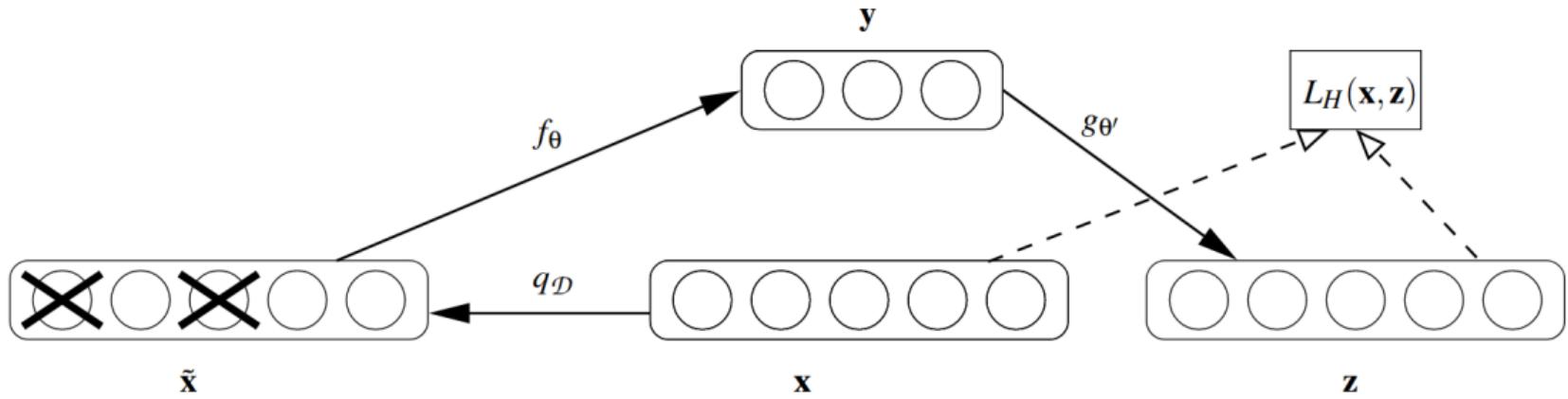
- ① Reconstruction from a corrupted or partial version from an example
- ② Visual common sense tasks
- ③ Contrastive Learning

# Reconstruction

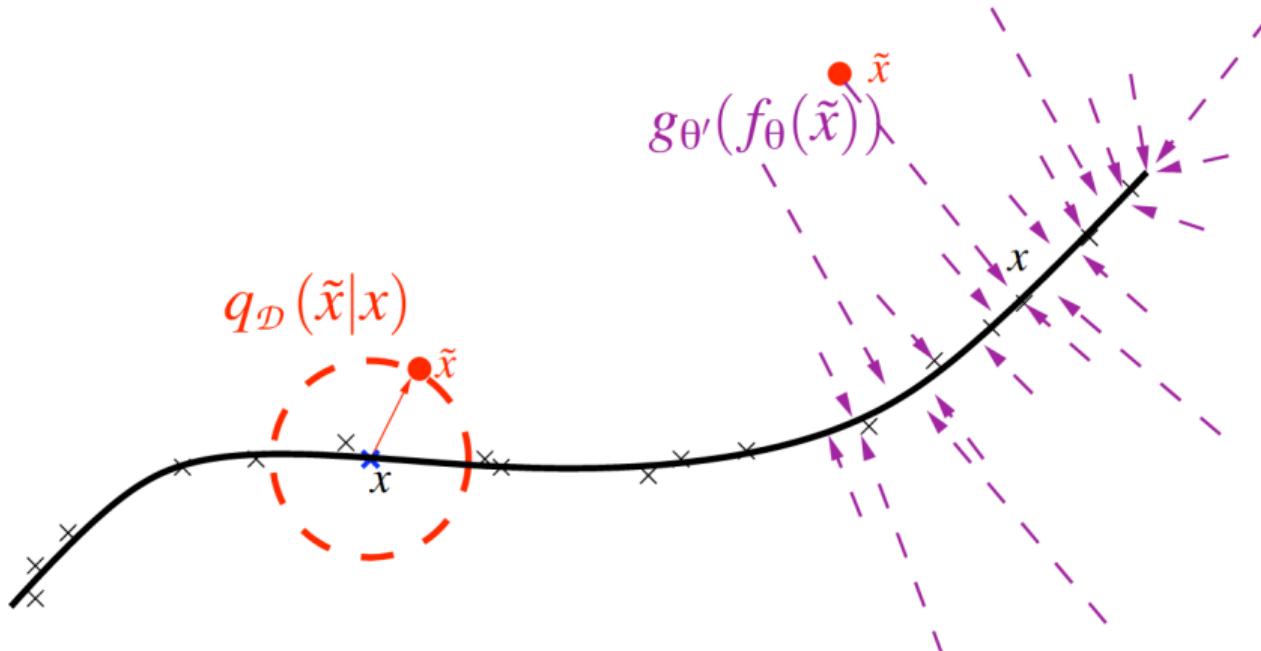
---

- An autoencoder learns features that represent the input in a more compact way
- These features could represent specific statistical characteristics of the dataset (overfitted)
- We can improve the features by learning a more difficult task: Learn to reproduce a corrupted version





[Vincent et al. 2010]



[Vincent et al. 2010]

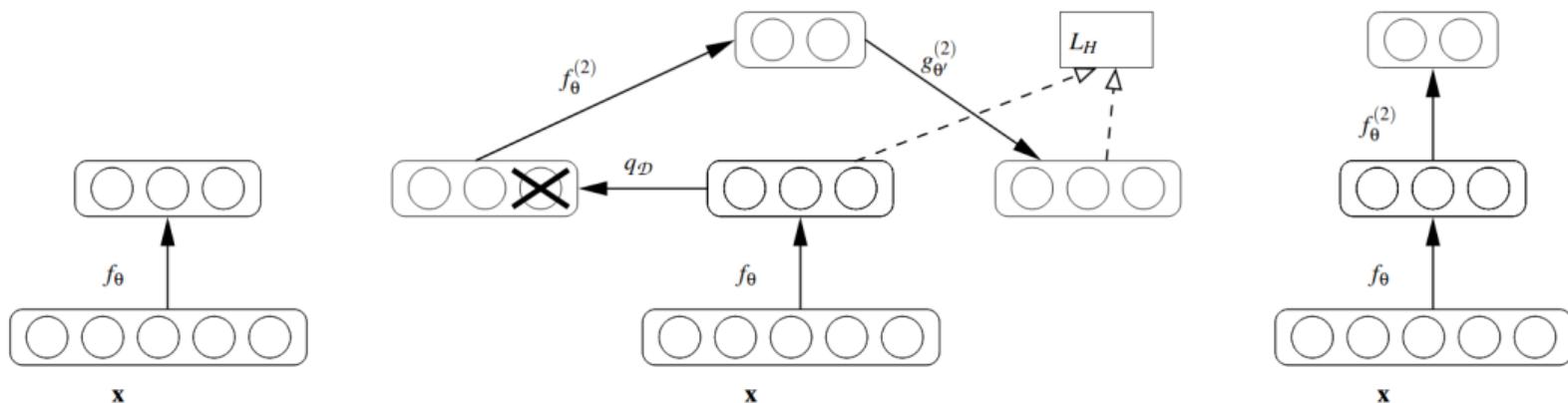
- ④ Minimize MSE weighting noise and original elements

$$\mathcal{L}_{2,\alpha} = \alpha \sum_{i \in corr(x)} (x_i - z_i)^2 + \beta \sum_{i \notin corr(x)} (x_i - z_i)^2$$

- ⑤ Minimize the cross entropy for binary data

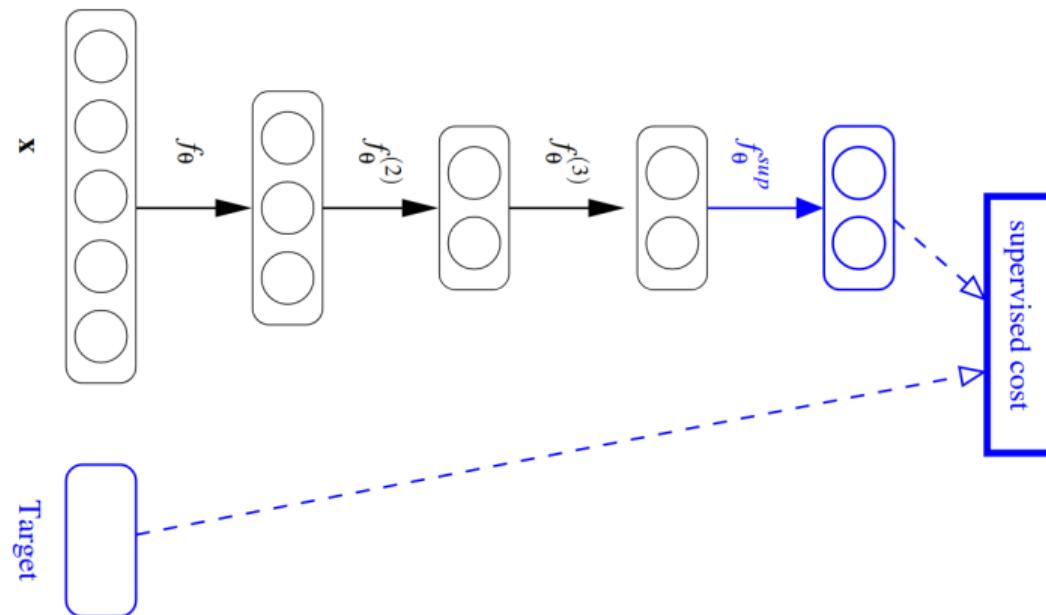
$$\begin{aligned}\mathcal{L}_{2,\alpha} &= -\alpha \sum_{i \in corr(x)} [(x_i \log z_i) + (1 - x_i) \log(1 - z_i)] \\ &\quad - \beta \sum_{i \notin corr(x)} [(x_i \log z_i) + (1 - x_i) \log(1 - z_i)]\end{aligned}$$

- The process can be repeated freezing previous layers and adding a new later that reconstructs from the output of the previous network

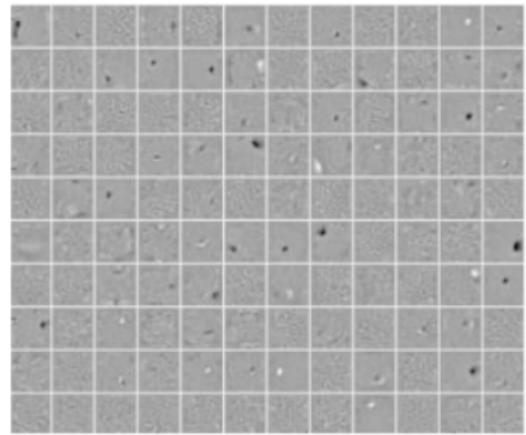


[Vincent et al. 2010]

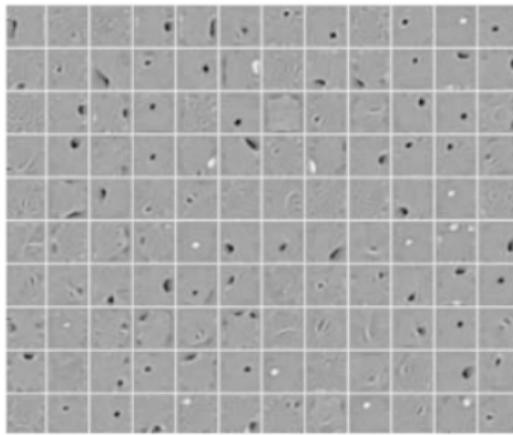
- ⑤ The resulting autoencoder can be used as features for a supervised task



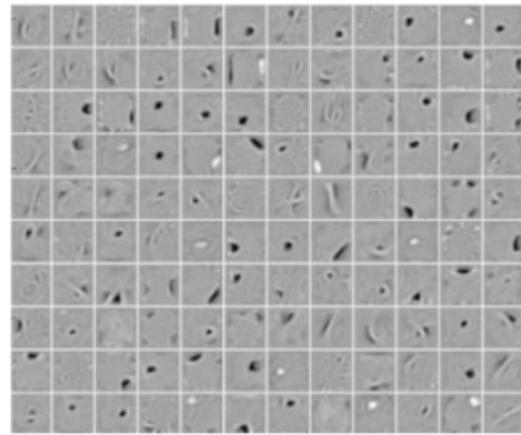
[Vincent et al. 2010]



(a) No corruption



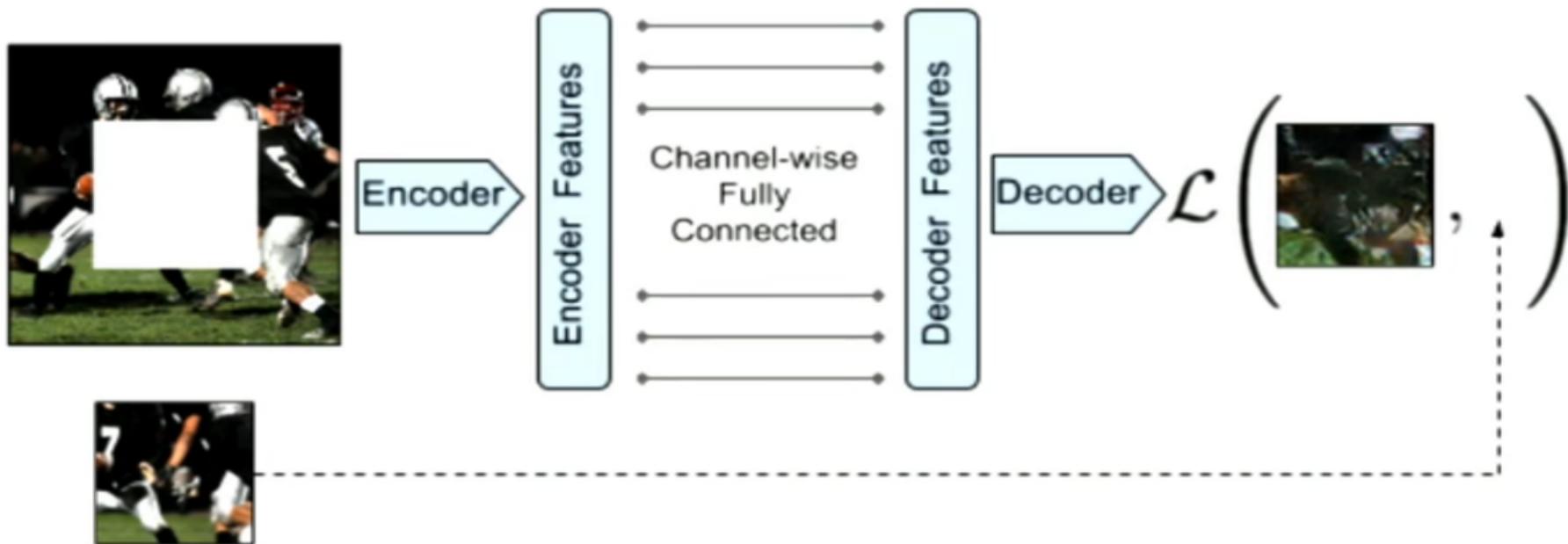
(b) 25% corruption



(c) 50% corruption

[Vincent et al. 2010]

- ⑤ Image inpainting is a task where a part of the image is masked, and it has to be generated using the rest of the image (e.g. image restoration)
- ⑥ To be able to predict a part of the image needs of the understanding of its content
- ⑦ An encoder-decoder network can be trained on different inpainting tasks to extract relevant features unsupervisedly (only the image is needed)



[Pathak et al. 2010]



(a) Central region



(b) Random block



(c) Random region

[Pathak et al. 2016]

- ④ Mean square distance loss (blurry reconstructions)

$$\mathcal{L}_{rec} = \|\mathbf{M} \odot (\mathbf{x} - F((\mathbf{1} - \mathbf{M}) \odot \mathbf{x}))\|_2$$

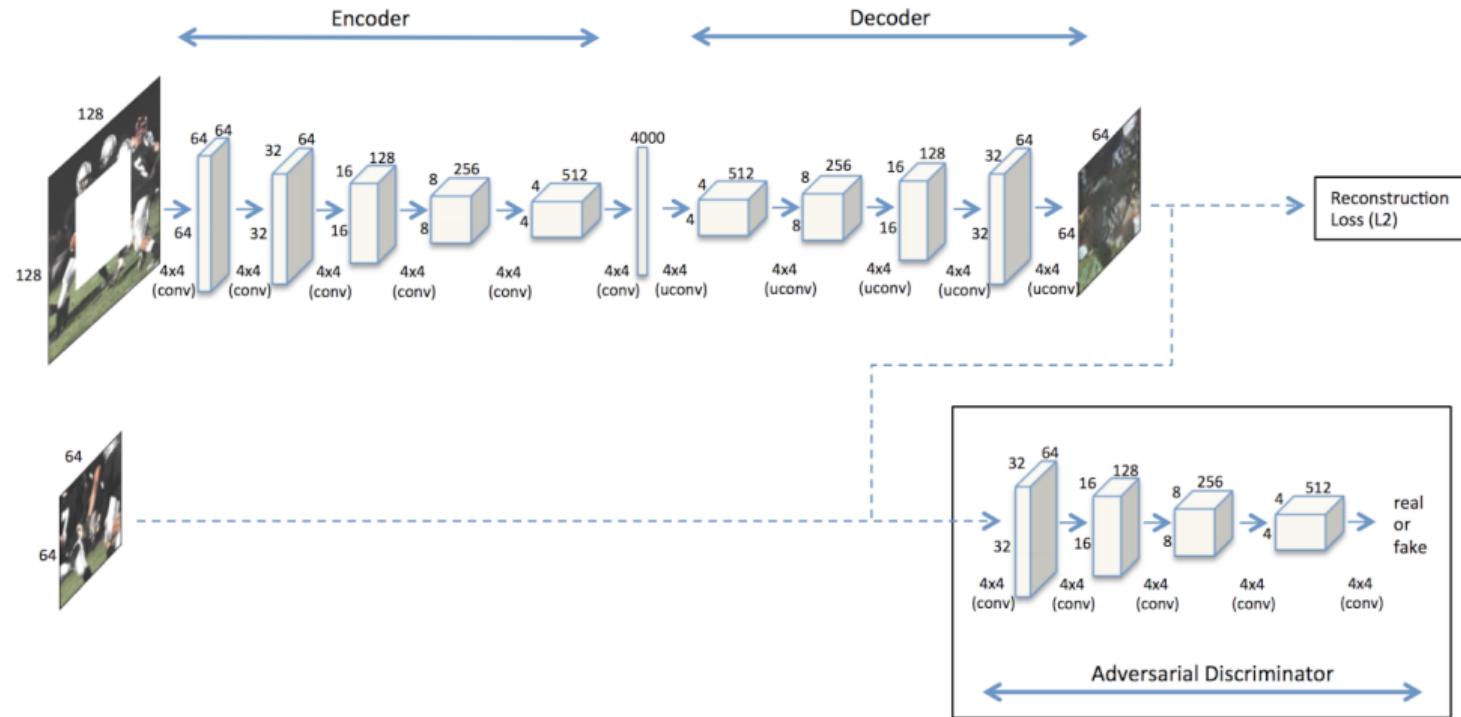
- ⑤ Adversarial loss

$$\mathcal{L}_{adv} = \max_D \mathbb{E}_{x \in \mathcal{X}} [\log D(x) + \log(1 - D(F((\mathbf{1} - \mathbf{M}) \odot \mathbf{x})))]$$

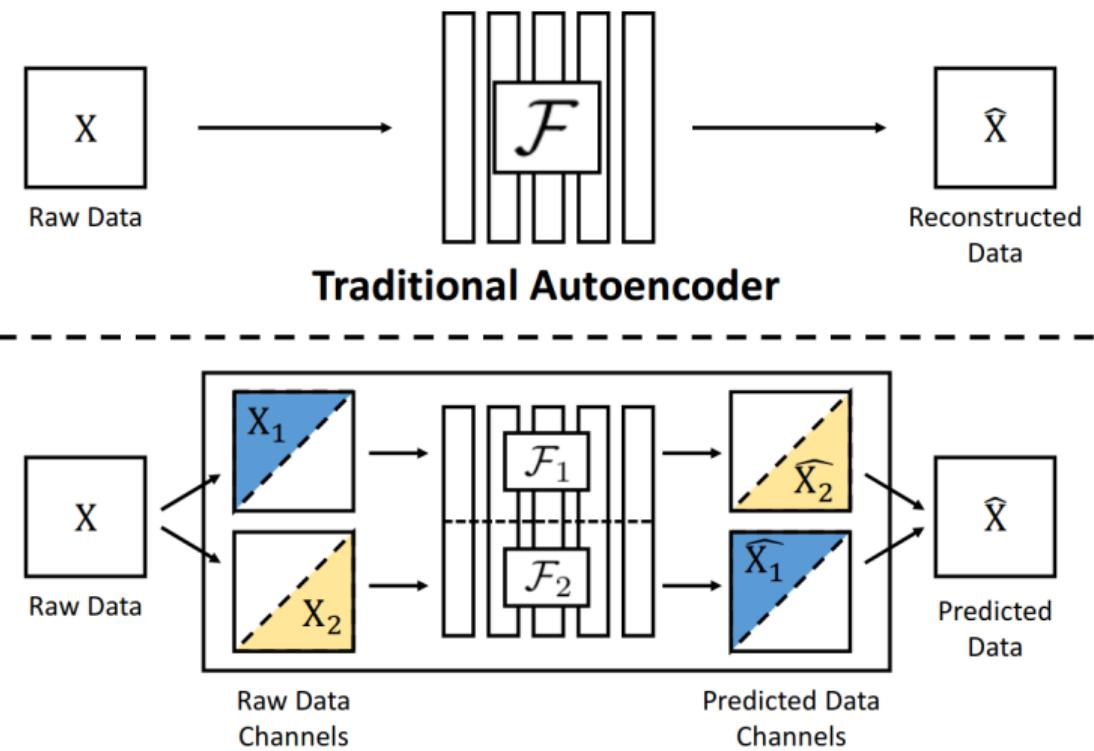


[Pathak et al. 2016]

## Context Encoders: Architecture



[Pathak et al. 2016]



[Zhang et al. 2017]

- Each part of the network has as input a part of the channels of the image and predicts the rest (Grayscale/Color, Depth/Color)
- The image is reconstructed as the concatenation of the predictions (all channels)
- Different variations of losses based on  $L_2$

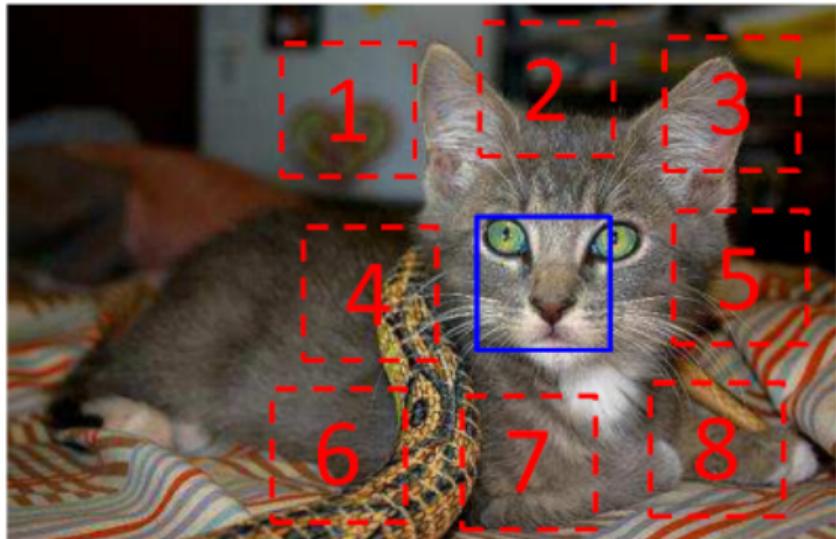
$$\mathcal{L} = \|\mathcal{F}(X_1) - X_2\|_2 + \|\mathcal{F}(X_2) - X_1\|_2$$

$$\mathcal{L} = \lambda \|\mathcal{F}(X_1) - X_2\|_2 + \lambda \|\mathcal{F}(X_2) - X_1\|_2 + (1 - 2\lambda) \|\mathcal{F}(X) - X\|_2$$

# Visual Common-sense Tasks

---

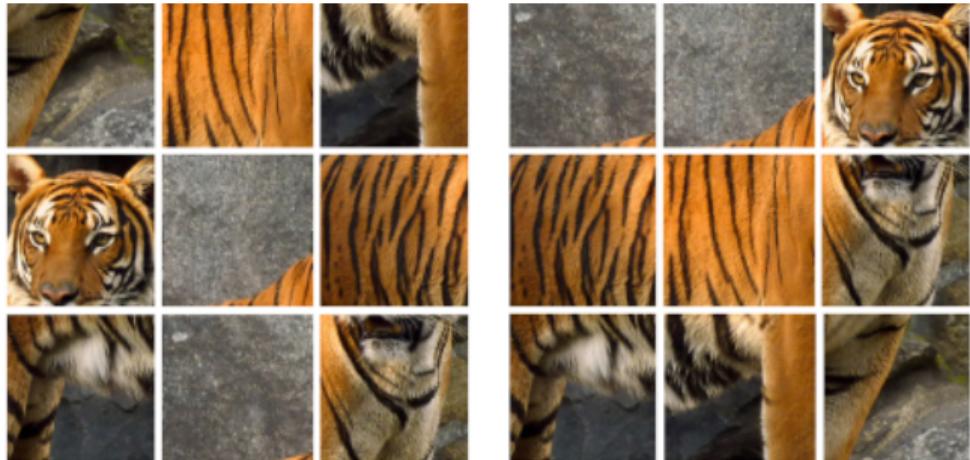
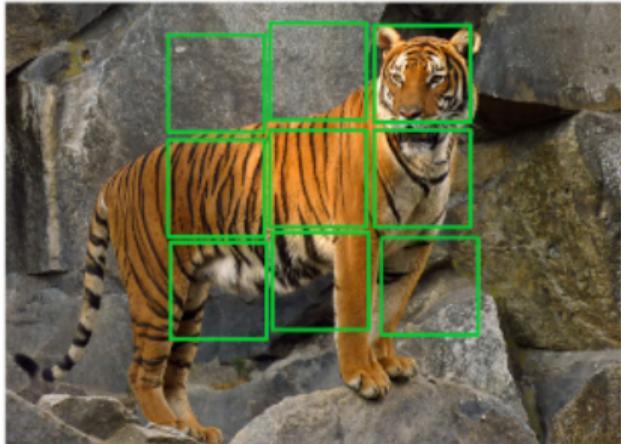
- Learn spatial associations between patches of an image
- Predict the position of a patch from another reference patch
- Non overlapped and jittered patches



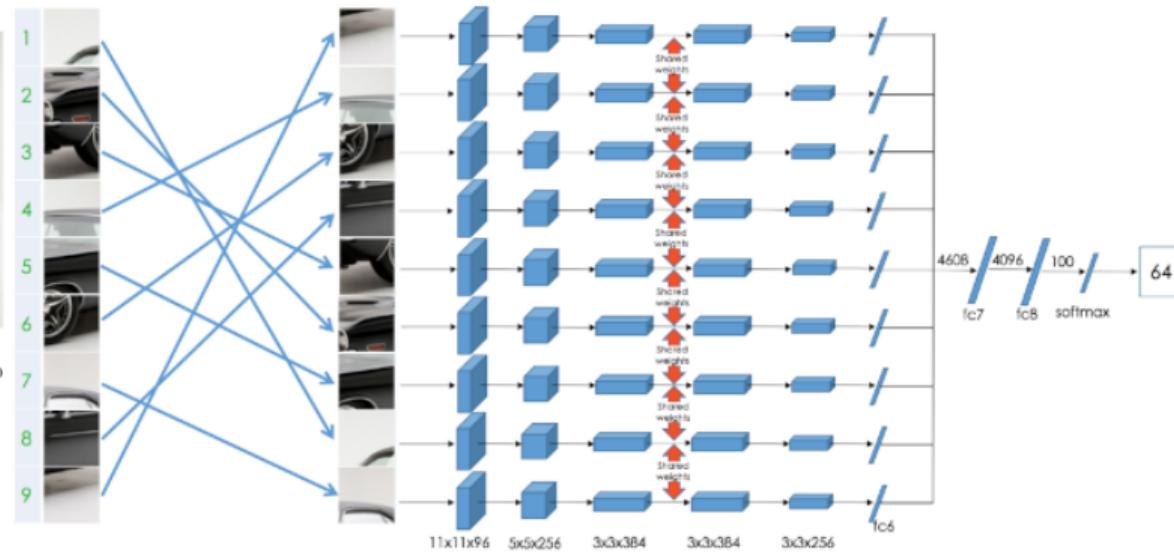
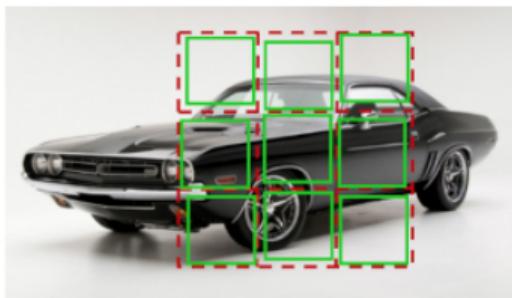
$$X = (\text{[Patch 5]}, \text{[Patch 3]}); Y = 3$$

[Doersch et al. 2017]

- Obtain the correct spatial order for a set of patches (jigsaw puzzle)

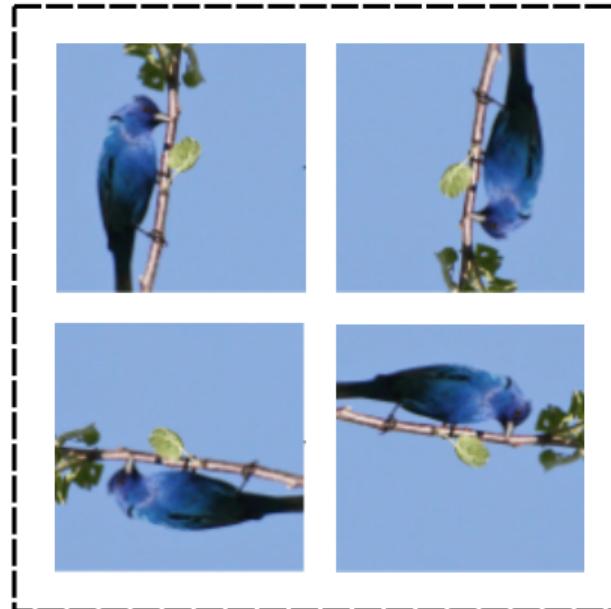


[Noroozi et al. 2017]

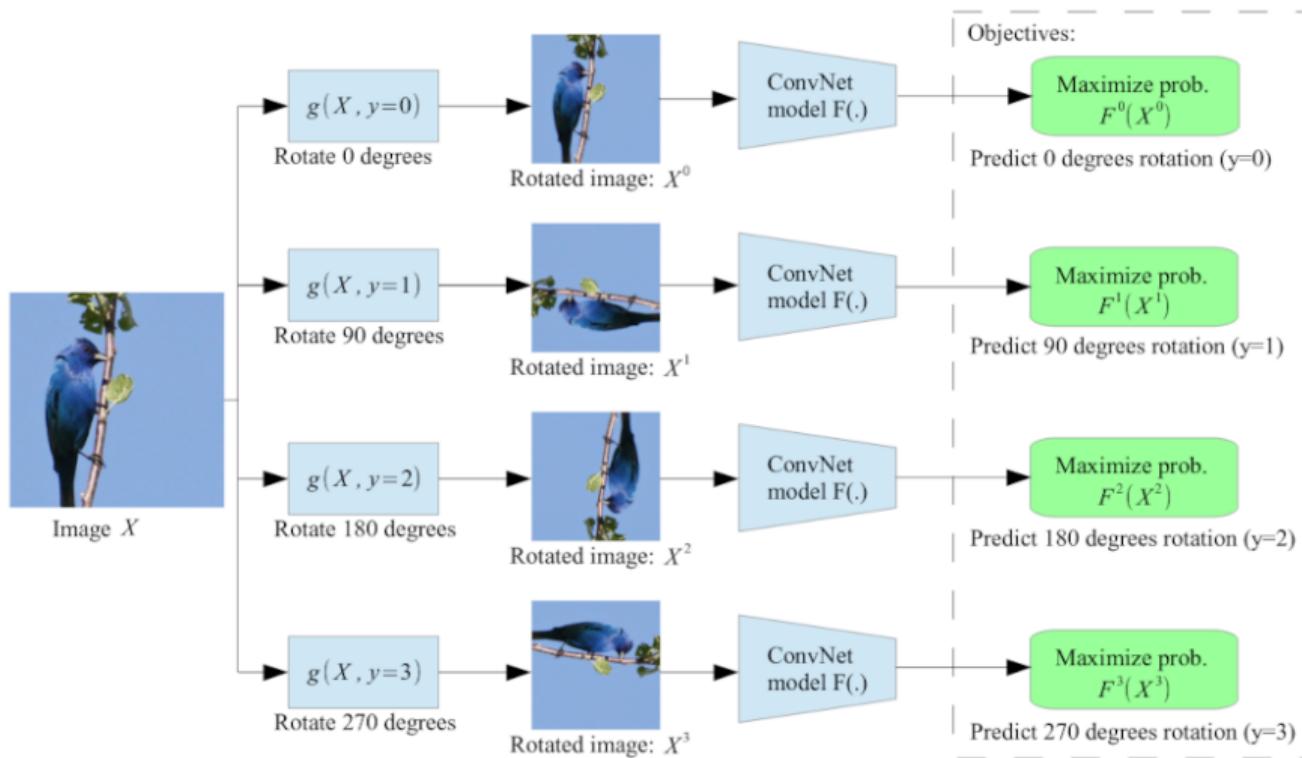


[Noroozi et al. 2017]

- ① Generate a dataset with images rotated in 2D at different angles (labels)
- ② Build a convolutional net able to predict the rotation angle of an image



[Gidaris et al. 2018]

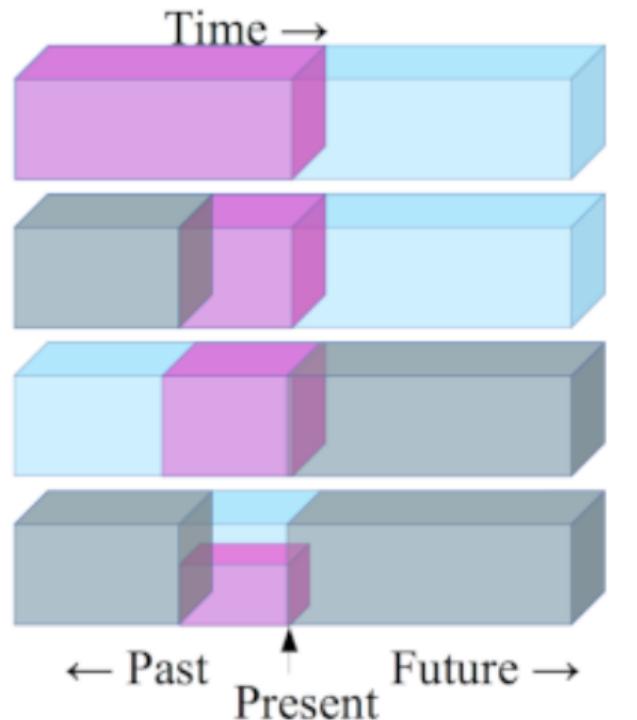


	Classification (%mAP)	Detection (%mAP)	Segmentation (%mIoU)
Trained layers	fc6-8	all	all
ImageNet labels	78.9	79.9	56.8
Random		53.3	43.4
Random rescaled [Krähenbühl et al. (2015)]	39.2	56.6	45.6
Egomotion (Agrawal et al., 2015)	31.0	54.2	43.9
■ Context Encoders (Pathak et al., 2016b)	34.6	56.5	44.5
Tracking (Wang & Gupta, 2015)	55.6	63.1	47.4
■ Context (Doersch et al., 2015)	55.1	65.3	51.1
Colorization (Zhang et al., 2016a)	61.5	65.6	46.9
BIGAN (Donahue et al., 2016)	52.3	60.1	46.9
■ Jigsaw Puzzles (Noroozi & Favaro, 2016)	-	67.6	53.2
NAT (Bojanowski & Joulin, 2017)	56.7	65.3	49.4
■ Split-Brain (Zhang et al., 2016b)	63.0	67.1	46.7
ColorProxy (Larsson et al., 2017)		65.9	38.4
■ Counting (Noroozi et al., 2017)	-	67.7	51.4
■ (Ours) RotNet	<b>70.87</b>	<b>72.97</b>	<b>54.4</b>
			<b>39.1</b>

# Contrastive Learning

---

- ▶ Predict any part of the input from any other part.
- ▶ Predict the **future** from the **past**.
- ▶ Predict the **future** from the **recent past**.
- ▶ Predict the **past** from the **present**.
- ▶ Predict the **top** from the **bottom**.
- ▶ Predict the **occluded** from the **visible**
- ▶ **Pretend there is a part of the input you don't know and predict that.**



- Word embeddings are vectorial word representations that allow representing a vocabulary as points in an N-dimensional space
- Each word has a code that can be used as the input of a neural network (or other learning model)
- The old ways: Singular Value Decomposition from cooccurrence matrices, Latent Semantic Analysis, Latent Dirichlet Allocation
  - Sparsity, high computational costs, noise, infrequent words
- The new ways: Unsupervised Deep learning

- ④ Probability of a sequence of words (sentence) is based on the probability of the words, and their cooccurrences in the sequence
- ④ Models based on estimating the probability of words (unigrams/bigrams)

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i)$$

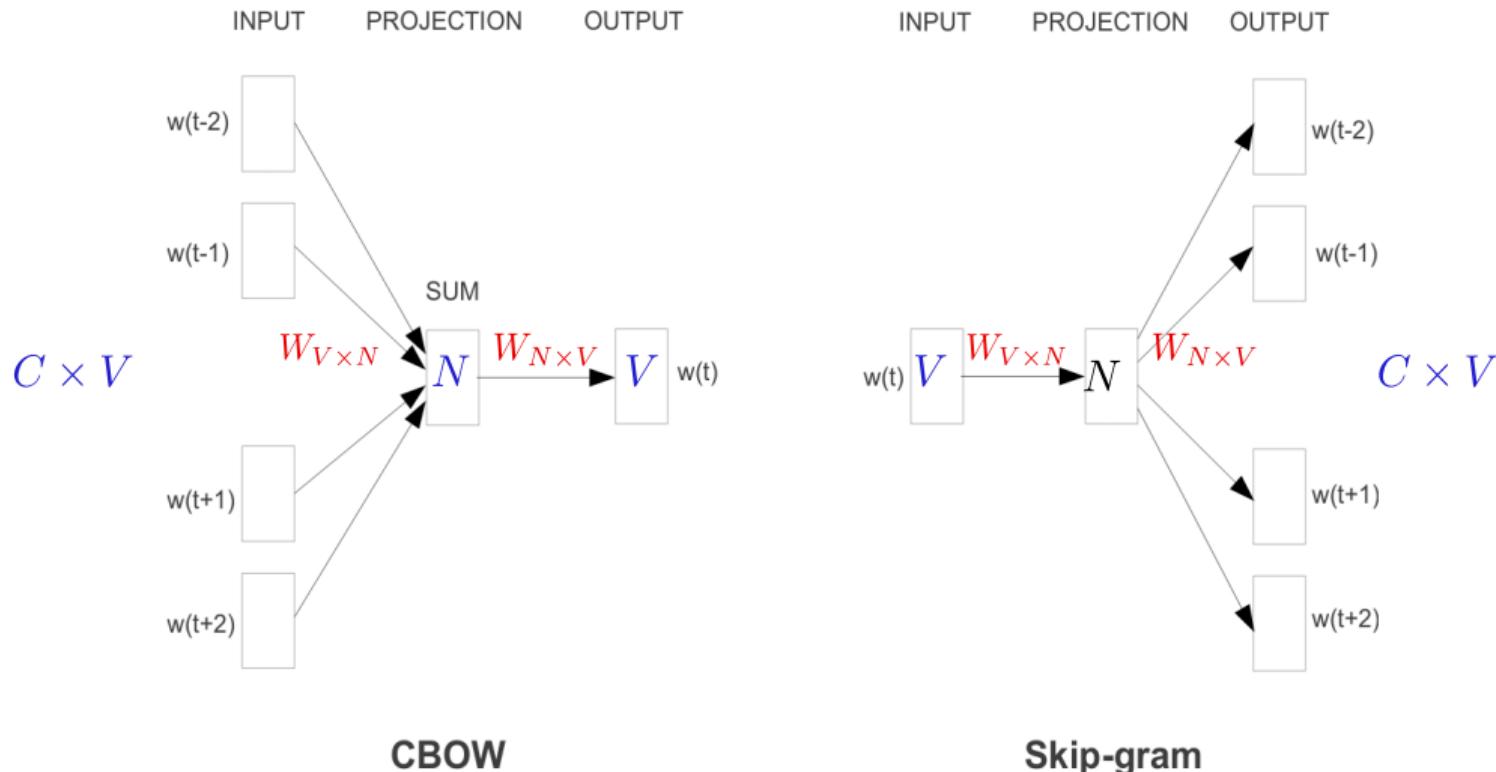
$$P(w_1, w_2, \dots, w_n) = \prod_{i=2}^n P(w_i | w_{i-1})$$

- ⑤ Continuous bag of words (generalizes the bag of words representation)
  - A word is predicted using the surrounding words

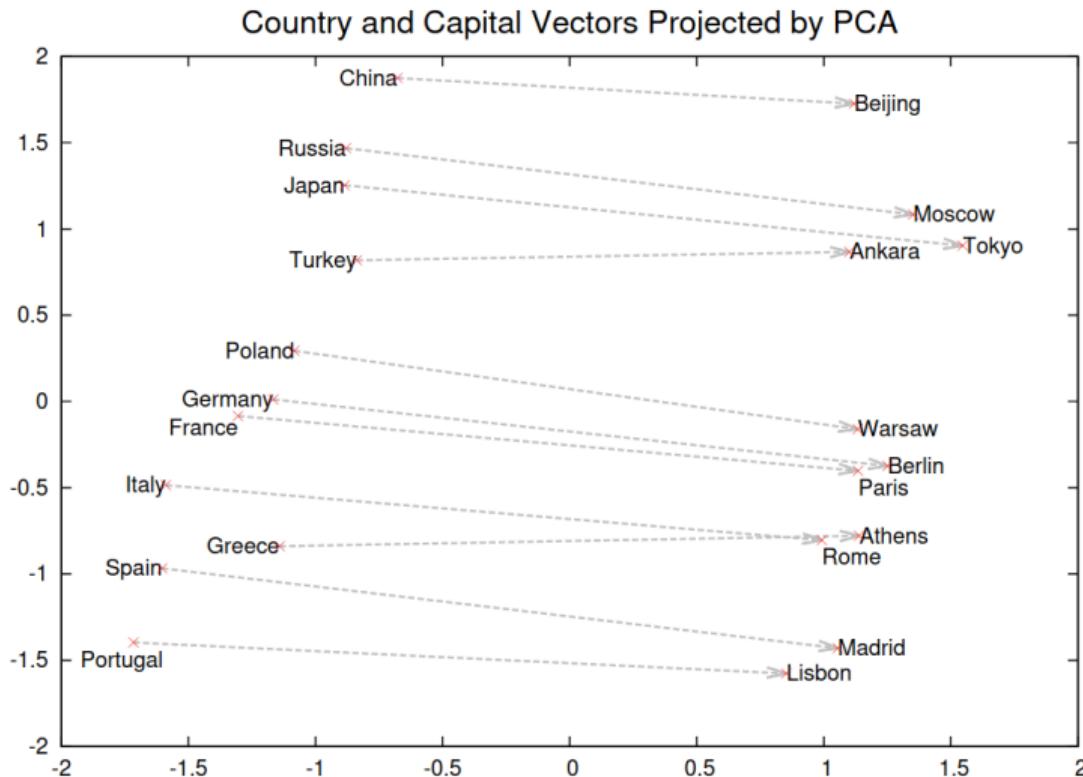
$$P(w_i | w_{i-c}, \dots, w_{i-1}, \dots, w_{i+1}, w_{i+c})$$

- ⑥ Skip gram
  - From a word we predict all the surrounding words

$$P(w_{i-c}, \dots, w_{i-1}, \dots, w_{i+1}, w_{i+c} | w_i)$$

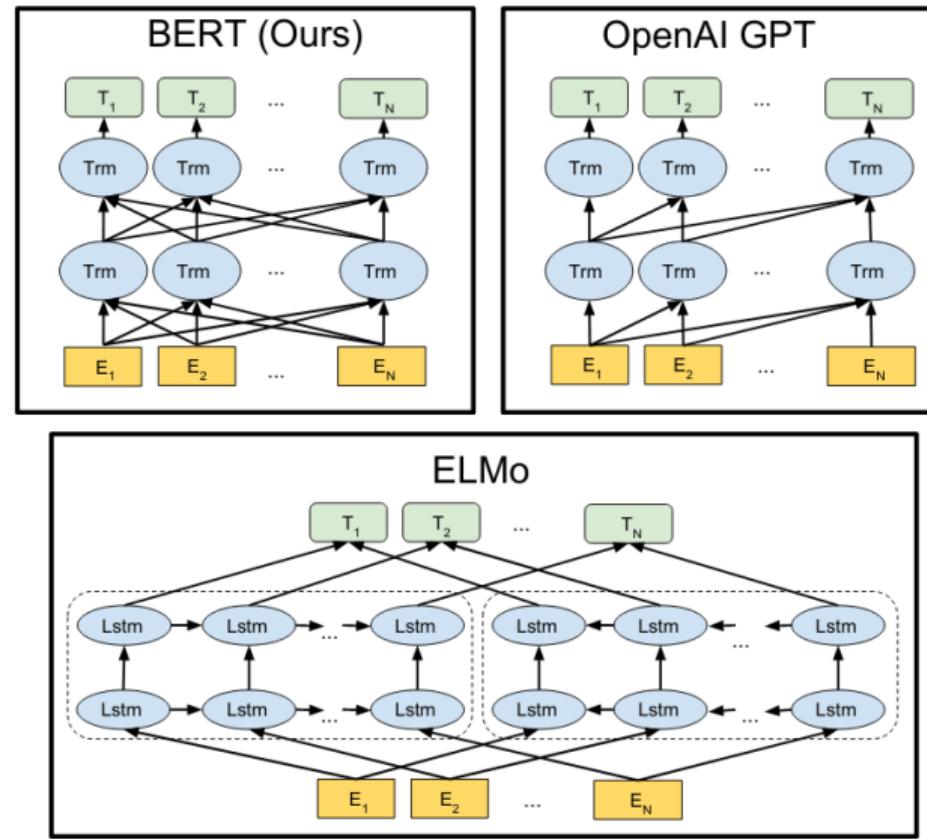


[Mikolov et al. 2013]

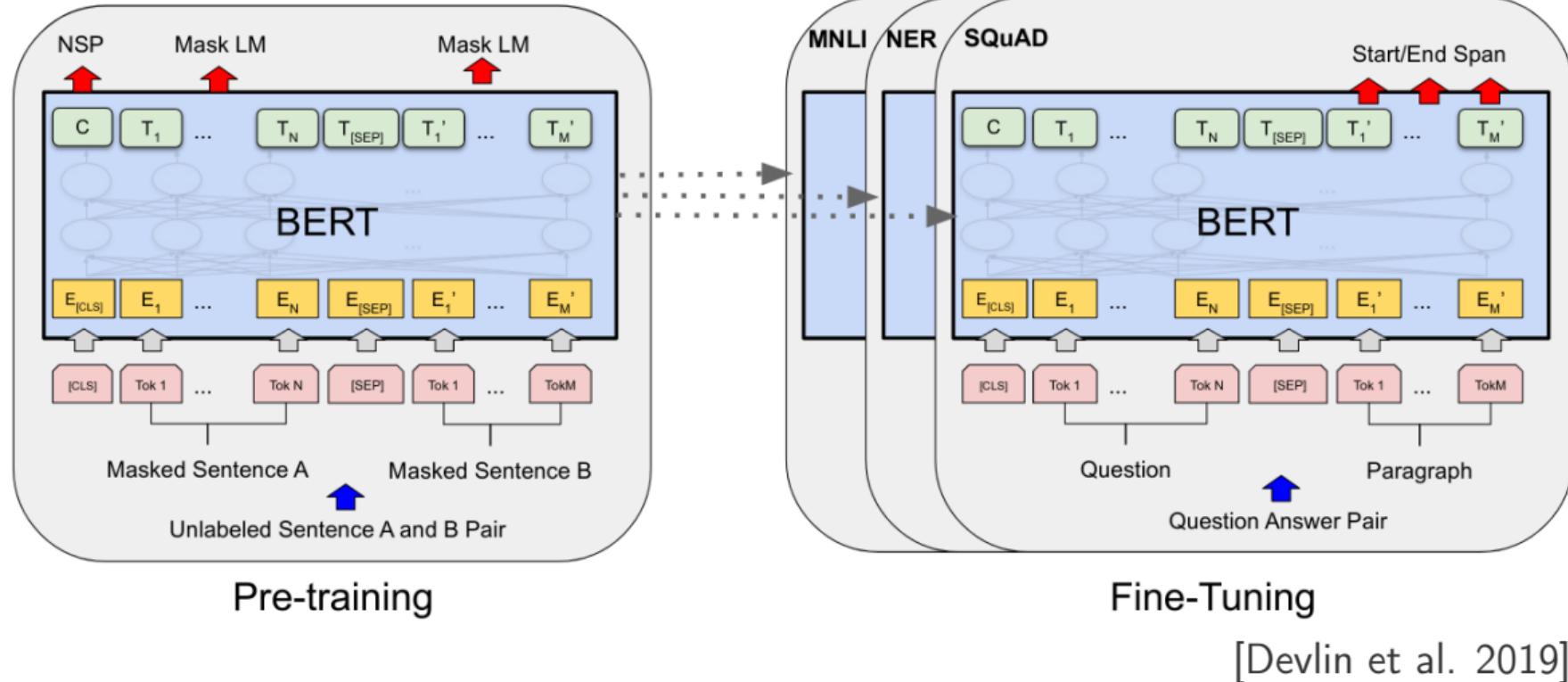


[Mikolov et al. 2013]

- A growing generation of models of word embeddings has emerged from this initial work
- Large corpus of text allow obtaining very good embeddings (BookCorpus (7,000 books), Wikipedia, 1 Billion words benchmark)
  - ELMo (Embedding for Language Models)
    - Two layer Bidirectional LSTMs, bidirectional language model
  - GPT(1-3) (Generative Pre-trained Transformer)
    - Autoregressive model, Transformer architecture (multi-headed attention), trillions of parameters, pre-trained for several tasks
  - BERT (Bidirectional Encoder Representations from Transformers)
    - Transformer architecture, bidirectional language model, fine-tuning for specific NLP tasks



[Devlin et al. 2019]



Pre-training

Fine-Tuning

[Devlin et al. 2019]

- CPC introduces the task of optimizing the prediction of examples from a distribution respect to negative examples (out of distribution)
- This can be related to distance learning where we want to minimize the distance between similar samples (same class), and maximize the distance to dissimilar samples (other classes)
- This can be done using contrastive losses, that will obtain transformations from the original data to a space where distances reflect the examples relationships
- The code obtained will be used for different downstream tasks

- CPC assumes that the examples  $x_t$  belong to a sequence
- Each element of the sequence can be transformed by a non-linear encoder to a lower dimensional space

$$z_t = g_{enc}(x_t)$$

- All the elements of the context can be summarized (encoded) by an autoregressive model

$$c_t = g_{ar}(z_{\leq t})$$

- The goal is to approximate the ratio between the conditional distribution of a future element given the context  $p(x_{t+k}|c_t)$ , and apriori distribution of the future element  $p(x_{t+k})$

$$f_k(x_{t+k}) \propto \frac{p(x_{t+k}|c_t)}{p(x_t)}$$

- ⑤ This can not be done directly, so it is approximated using a log bilinear model

$$f_k(x_{t+k} = \exp(z_{t+k}^\top W_k c_t)$$

- ⑥ We can not estimate directly the probability distribution of the examples, but Noise-Contrastive Estimation (NCE) can be used by using positive and negative samples
- ⑦ The loss that is optimized is based on using sets of samples of size  $N$  where  $N - 1$  are negative samples

$$\mathcal{L}_N = -\mathbb{E}_X \left[ \log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right]$$

## ⑤ Audio:

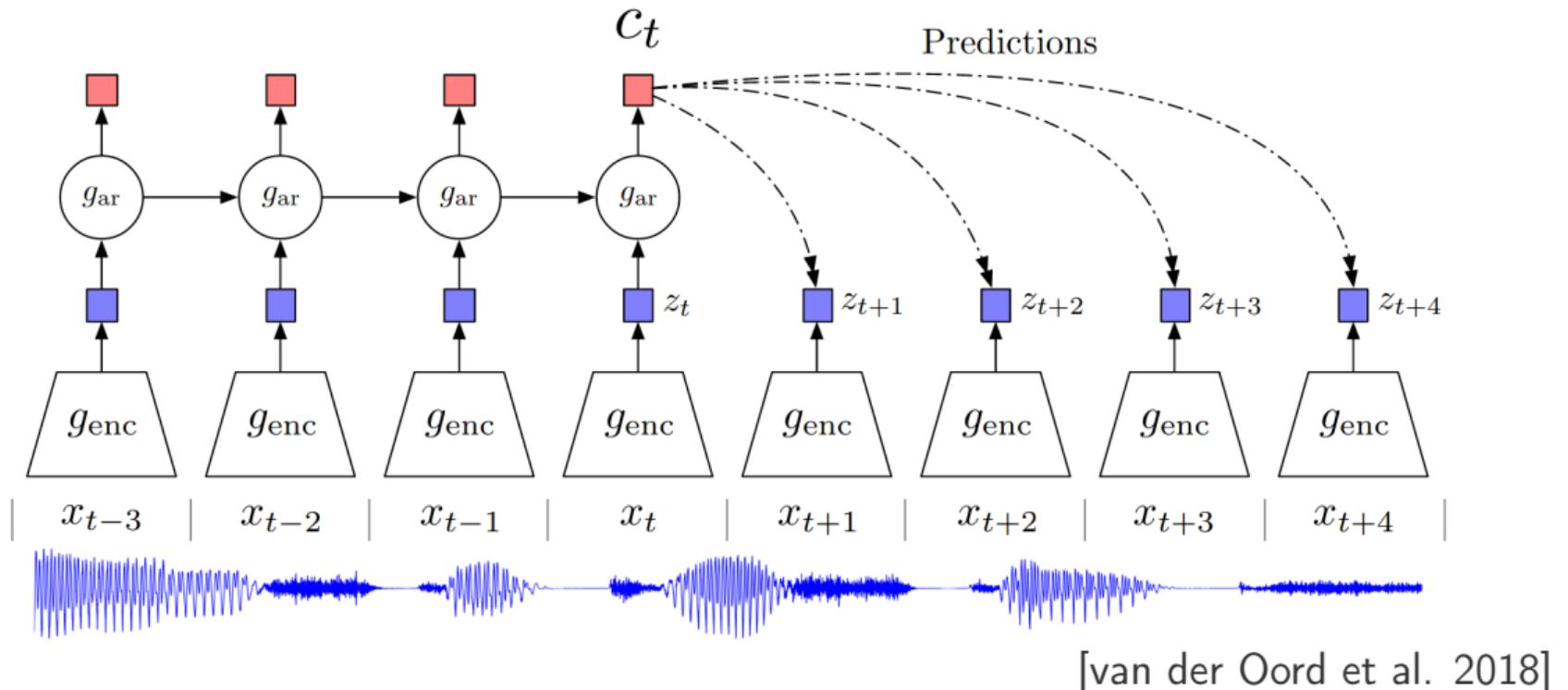
- Phoneme aligned chunks, prediction of 12 steps, encoder: CNN, autoregressive: GRU RNN, negative sampling: other elements of sequence, other speakers

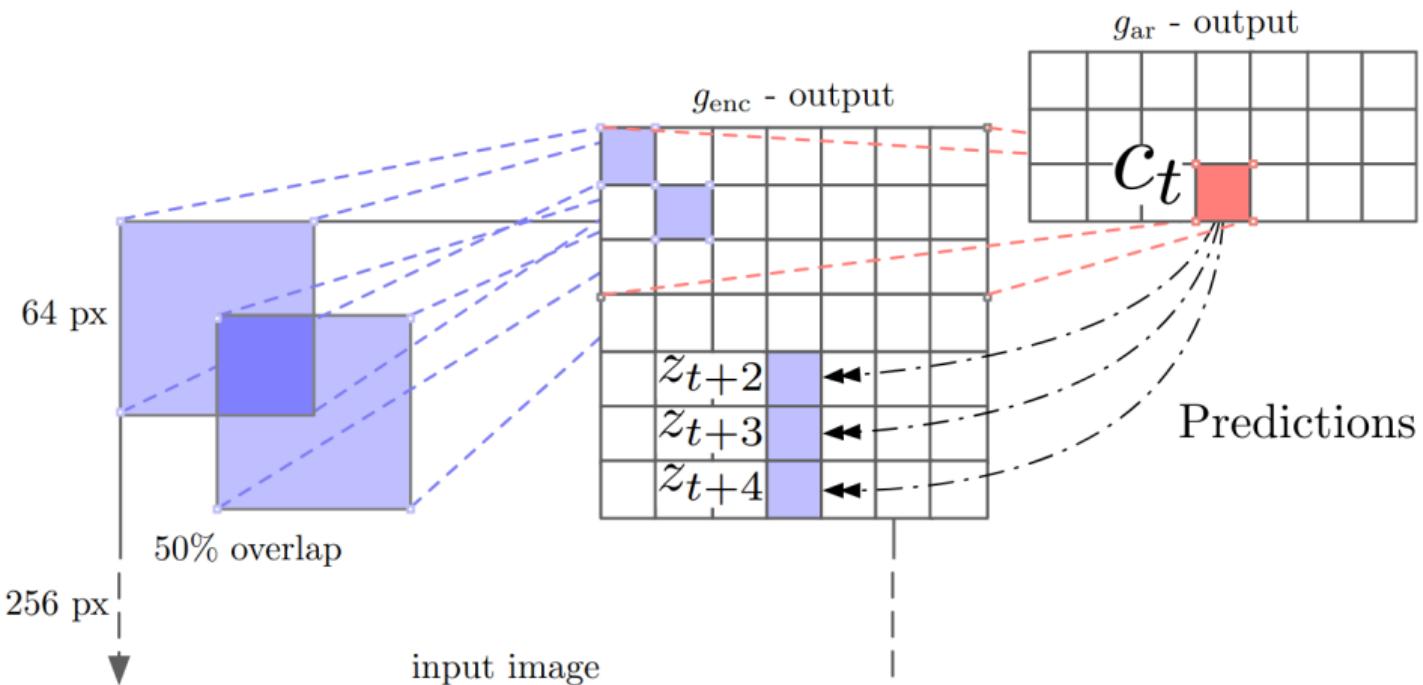
## ⑥ Video:

- Image patches, prediction of patches, encoder: Resnet v2, autoregressive: PixelCNN, negative sampling: random patches

## ⑦ NLP:

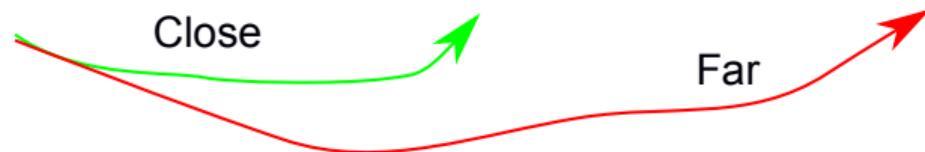
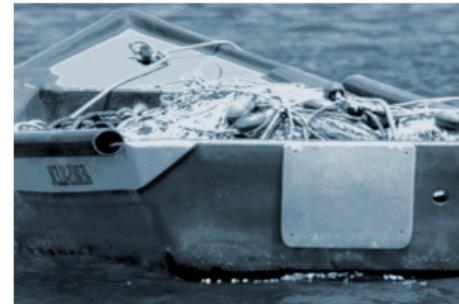
- Word embeddings, prediction sentence embeddings of three following sentences, encoder: sentence encoder with 1D convolutions, autoregressive: GRU RNN, negative sampling: other sentences of sequence, other documents sentences





[van der Oord et al. 2018]

- To use patches or elements of a sequence can be data intensive
- Another approach is to work directly with whole examples
- In the same line of distance learning, we can obtain a code that learns to put close an image and different versions of itself and far different images



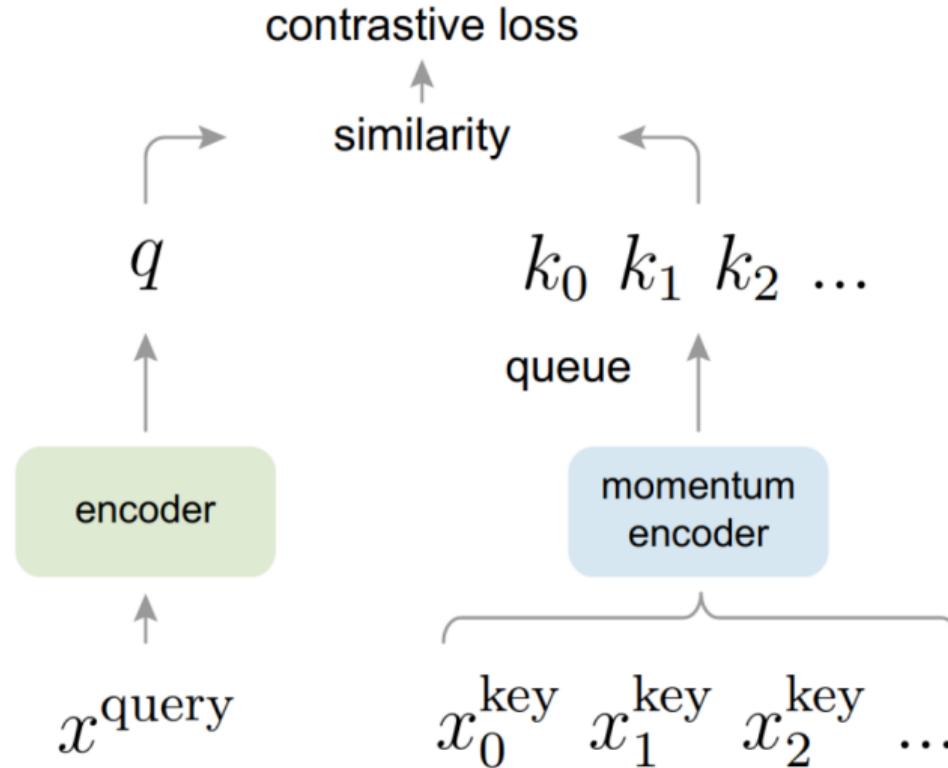
- Contrastive learning can be interpreted as the task of building dynamic dictionaries
- These dictionaries are composed by codes that are obtained using an encoding NN
- The learning process trains the encoder to dictionary lookups
- The obtained code for a new query should be similar to its matching key (a similar element), and different from other keys
- This is obtained by minimizing a contrastive loss
- The goal is obtain dictionaries that are large and consistent during training

- The samples used for contrast learning are stored on a queue
- This allows to have a large dictionary that does not depend on the batch size for training (samples are looked up dynamically from the dictionary)
- Samples from the oldest batch are replaced by the newest batch of examples
- The encoder for the query (current example), and the contrast samples are separated (but with the same architecture)
  - The encoder for the query is updated using gradient descent from the contrastive loss
  - The encoder for the contrastive samples is updated using a fraction of the corresponding parameters of the query encoder (slow change)
- The contrastive loss is computed for all examples (all-vs-all) to reduce computational cost

---

```
encoder_q.params = encoder_k.params;
for x in batches do
    q = encoder_q.forward(augment(x));
    k = encoder_k.forward(augment(x));
    pos = exp(q[1]*k[1]);
    neg = sum(exp(q, queue.sampling()));
    encoder_q.backprop(loss(pos, neg));
    encoder_k.params = m * encoder_k.params + (1-m) encoder_q.params;
    queue.replace_oldest(k)
```

---



[He et al. 2020]

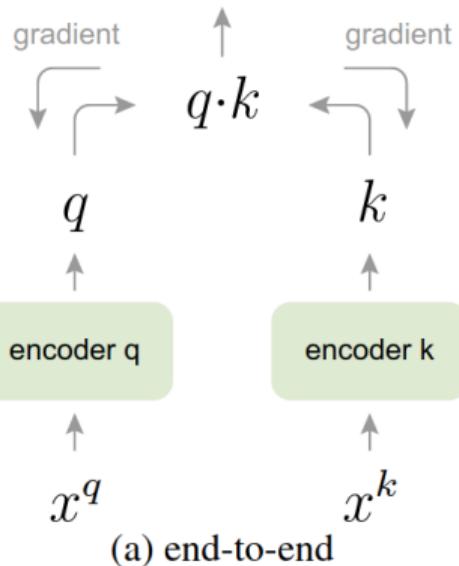
- ④ The loss is a variation of the Noise Contrastive Estimation Loss

$$\mathcal{L}(q, k^+, \{k^-\}) = -\log \frac{\exp(q \cdot k^+ / \tau)}{\exp(q \cdot k^+ / \tau) + \sum_{k^-} \exp(q \cdot k^- / \tau)}$$

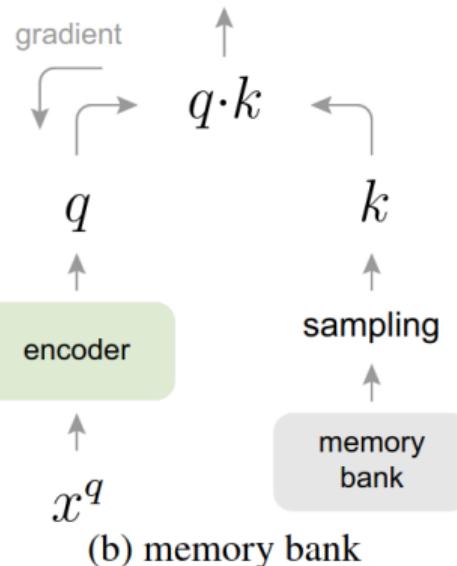
- ④  $\tau$  is a temperature hyper parameter
- ④ The momentum parameter  $m$  is usually maintained very high (0.99) so the change of the keys encoder is very slow

$$\theta_k = m\theta_k + (1 - m)\theta_q$$

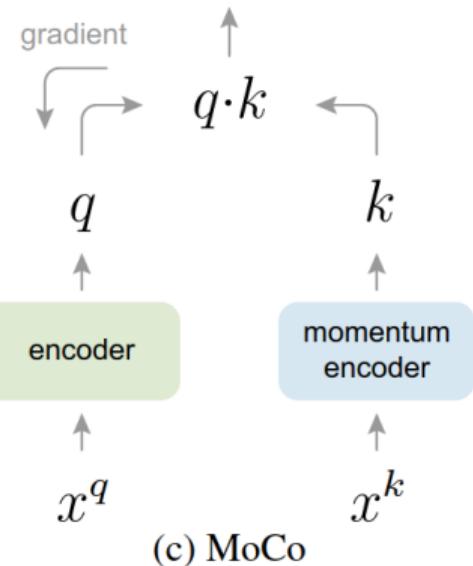
contrastive loss



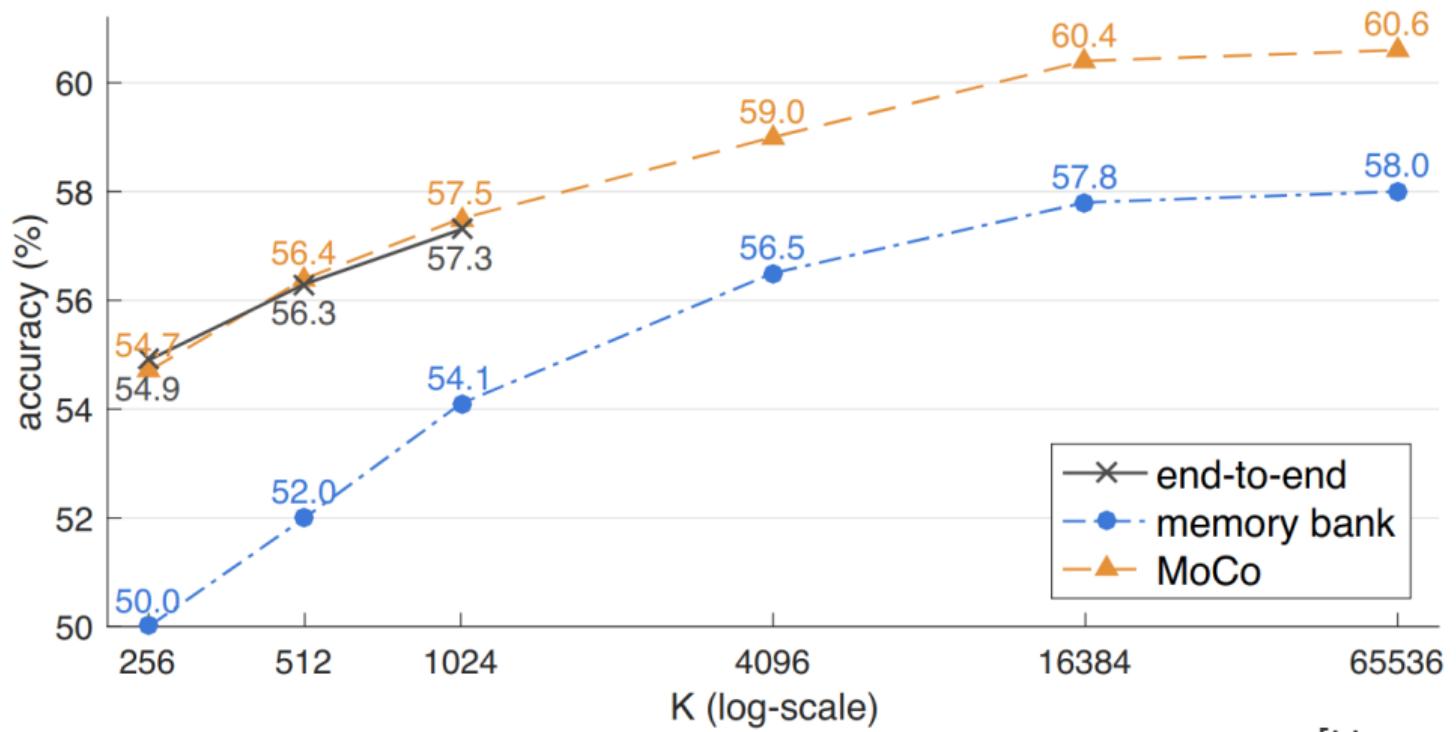
contrastive loss



contrastive loss



[He et al. 2020]



[He et al. 2020]

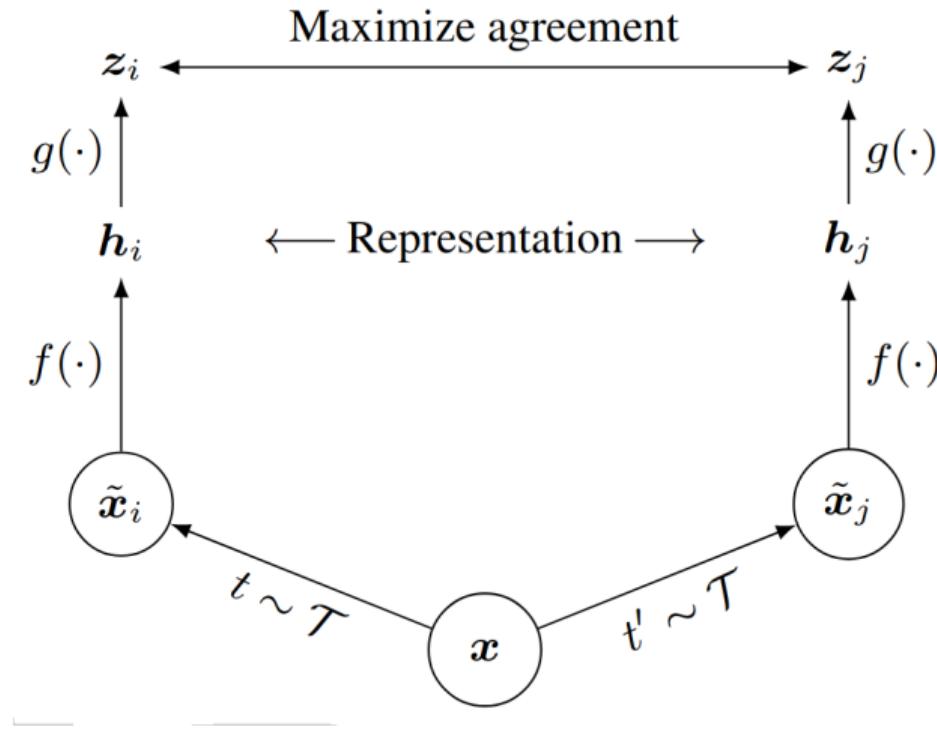
- ④ SimCLR uses instance discrimination, and the end to end version of contrast learning (negative samples in batch)
- ④ Uses Resnet as encoding network, and a non-linear transformation (MLP) between the code and the contrastive loss
- ④ Large batches (so there are many negative samples) and long training
- ④ Many data augmentation techniques, composed data augmentation helps (two different transformations to the images, crop+color distortion best)
- ④ Scaled cosine similarity ( $[-\tau, \tau]$  range)

- Normalized Temperature-Scaled Cross Entropy loss (NT-Xent)
- All augmented examples are used to compute the loss (queries and keys,  $2N$  examples)
- The loss for a positive pair

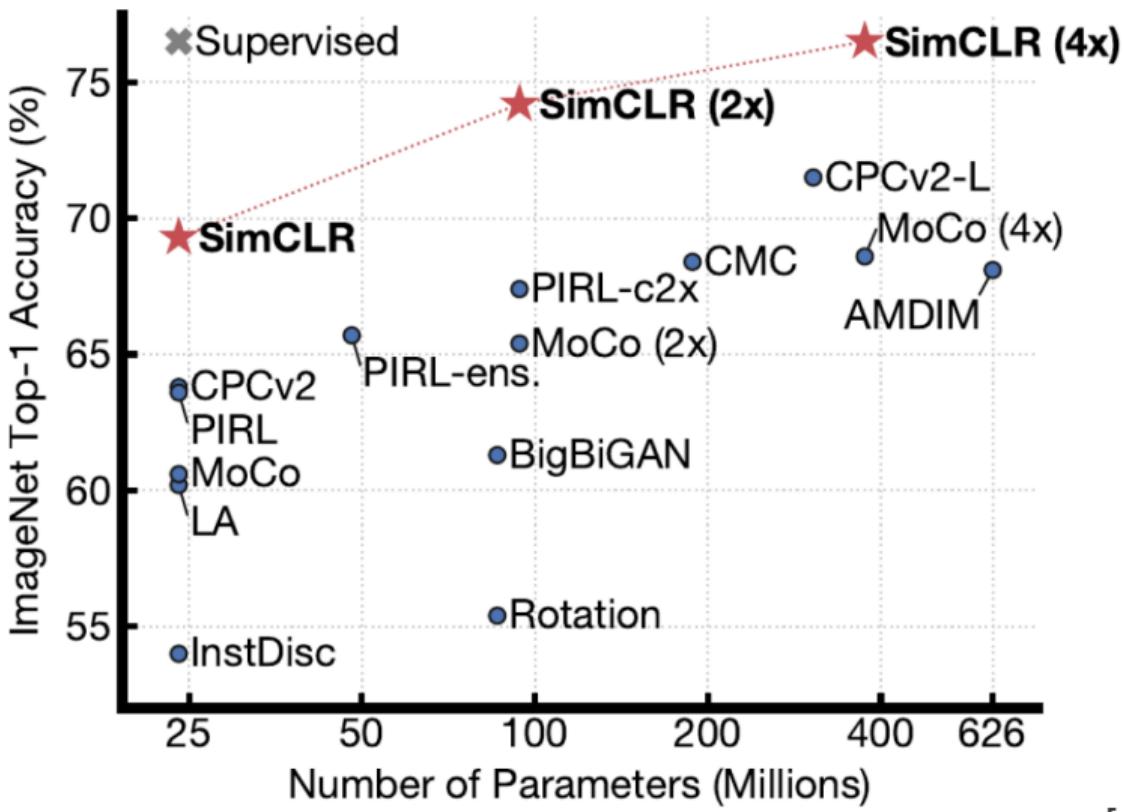
$$\mathcal{L}(i, j) = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}[k \neq i] \exp(\text{sim}(z_i, z_k)/\tau)}$$

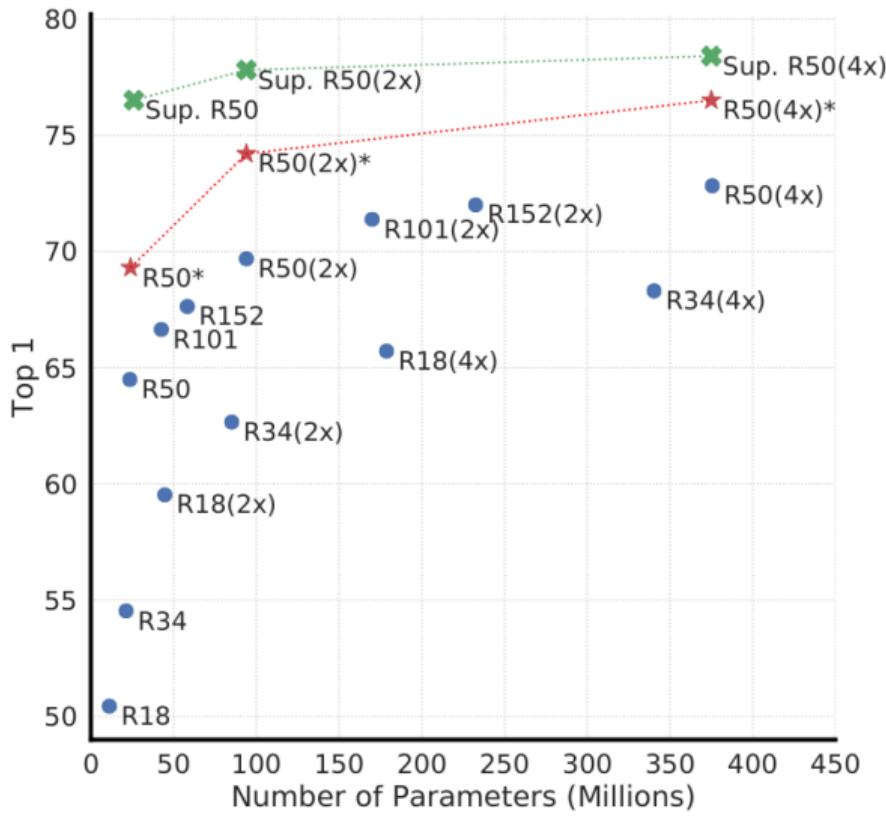
- The final loss is computed across all positive pairs (both directions)

$$\mathcal{L} = \frac{1}{2N} \sum_{k=1}^N [\mathcal{L}(k, 2k) + \mathcal{L}(2k, k)]$$



[Chen et al. 2020]





[Chen et al. 2020]

case	unsup. pre-train					ImageNet acc.
	MLP	aug+	cos	epochs	batch	
MoCo v1 [6]				200	256	60.6
SimCLR [2]	✓	✓	✓	200	256	61.9
SimCLR [2]	✓	✓	✓	200	8192	66.6
<b>MoCo v2</b>	✓	✓	✓	200	256	<b>67.5</b>

*results of longer unsupervised training follow:*

SimCLR [2]	✓	✓	✓	1000	4096	69.3
<b>MoCo v2</b>	✓	✓	✓	800	256	<b>71.1</b>

[Chen et al. 2020]