



國立交通大學  
National Chiao Tung University

# Unsupervised Learning

## ML 2021 Fall

Wei-Chen (Walton) Chiu  
邱維辰

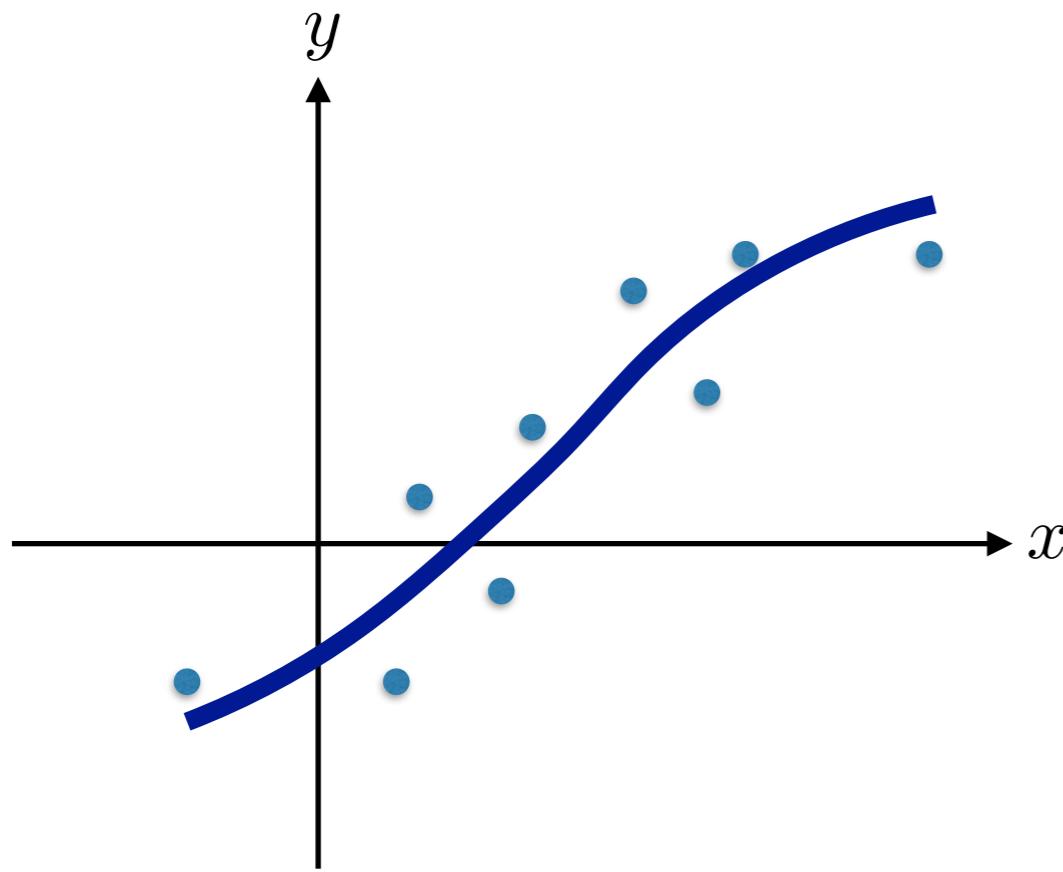
Enriched Vision Applications  
Laboratory



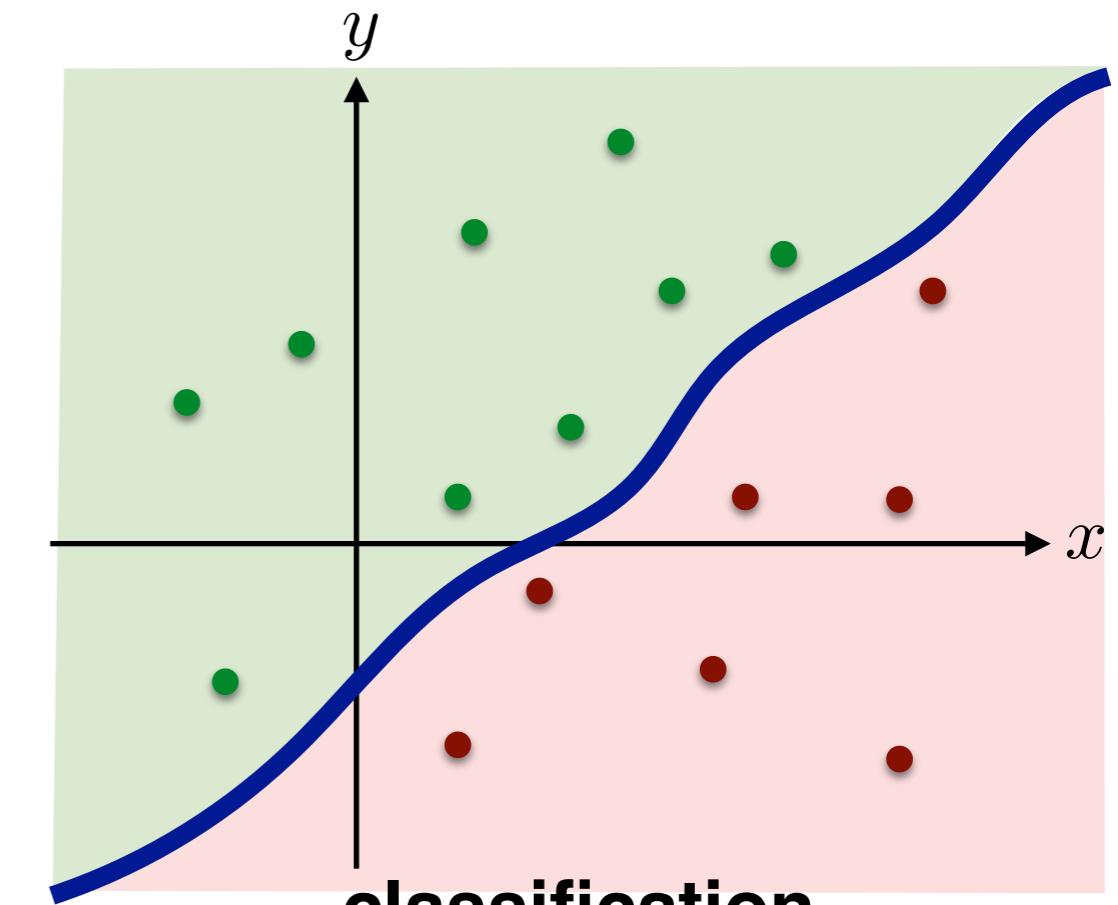
slides courtesy: Arati Singh@CMU, Mario Fritz@MPI-INF,  
Margarita Osadchy@U.Haifa, Emily Fox@U.Washington

# Three Types of Learning

- Assume that we have a set of training inputs  $x_1, x_2, \dots, x_n$ 
  - **Supervised Learning** (what you guys have learnt most): we are also given the desired outputs  $y_1, y_2, \dots, y_n$  and the goal is to learn to produce the correct output given a new input



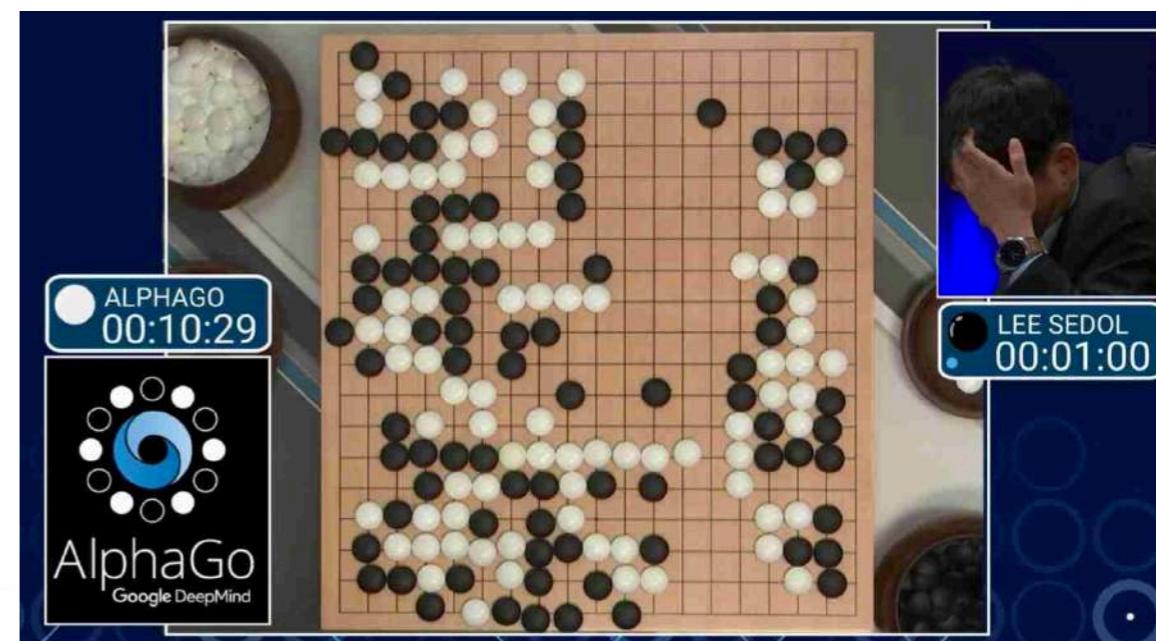
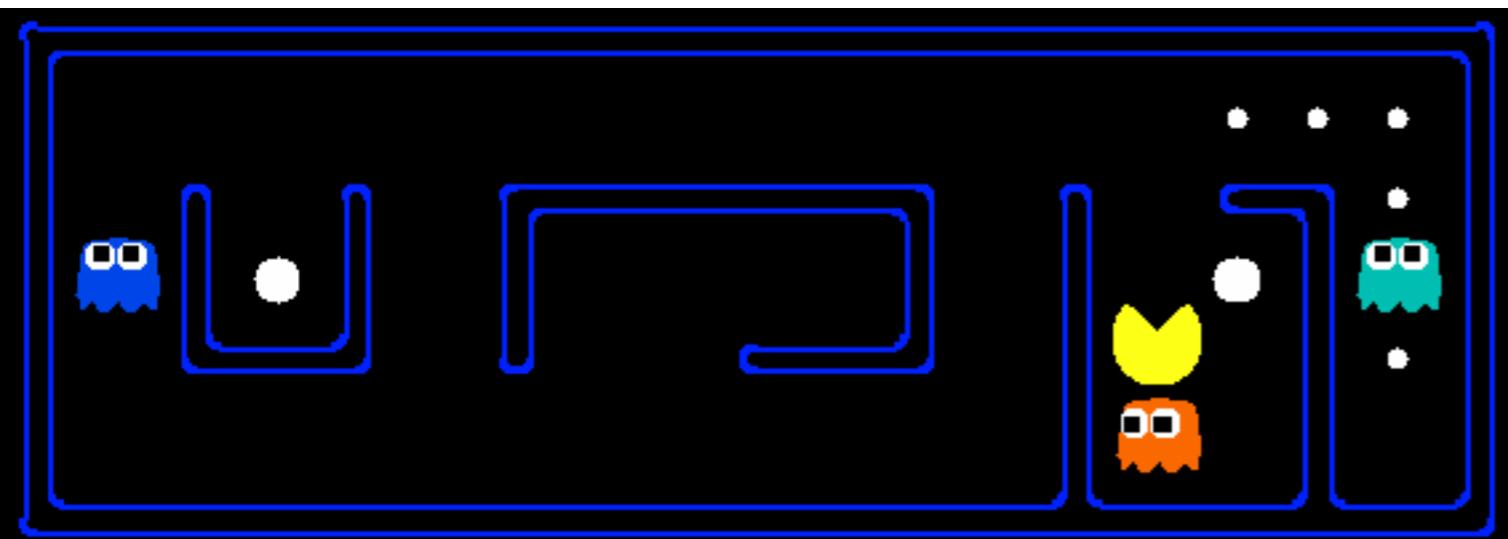
**regression**  
(continuous input-output mapping)



**classification**  
(output discrete variables  
learn a decision boundary)

# Three Types of Learning

- Assume a machine which has a set of inputs  $x_1, x_2, \dots, x_n$ 
  - **Supervised Learning** (what you guys have learnt most): the machine is also given the desired outputs  $y_1, y_2, \dots, y_n$  and the goal is to **produce the correct output** given a new input
  - **Reinforcement Learning**: the machine interacts with its environment by producing **actions**  $a_1, a_2, \dots$  which affect the state of the environment, and receive **rewards**  $r_1, r_2, \dots$ . Its goal is to **learn to act** (sometimes called **policy, decision making**) in a way that **maximize rewards** in the long term.



# Three Types of Learning

- Assume a machine which has a set of inputs  $x_1, x_2, \dots, x_n$ 
  - **Supervised Learning** (what you guys have learnt most): the machine is also given the desired outputs  $y_1, y_2, \dots, y_n$  and the goal is to learn to **produce the correct output** given a new input
  - **Reinforcement Learning**: the machine interacts with its environment by producing **actions**  $a_1, a_2, \dots$  which affect the state of the environment, and receive **rewards**  $r_1, r_2, \dots$ . Its goal is to **learn to act** (sometimes called **policy, decision making**) in a way that **maximize rewards** in the long term.
  - **Unsupervised Learning**: the machine only has input, no supervised output  $y$ , nor rewards from its environment



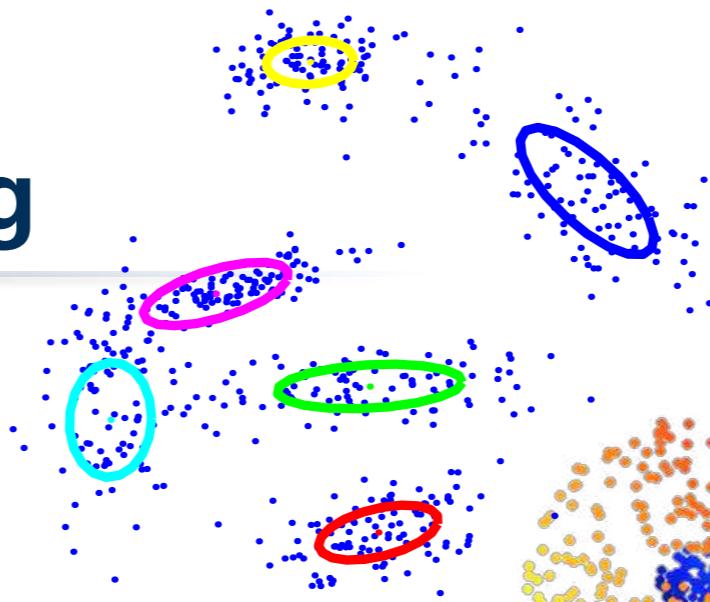
what do these mean?

# Three Types of Learning

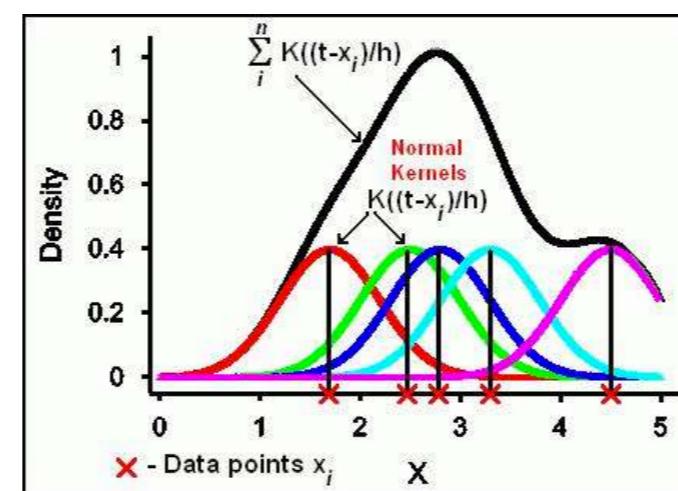
- Assume a machine which has a set of inputs  $x_1, x_2, \dots, x_n$ 
  - Supervised Learning (what you guys have learnt most): the machine is also given the desired outputs  $y_1, y_2, \dots, y_n$  and the goal is to produce the correct output given a new input
  - Reinforcement Learning: the machine interacts with its environment by producing actions  $a_1, a_2, \dots$  which affect the state of the environment, and receive rewards  $r_1, r_2, \dots$ . Its goal is to learn to act (sometimes called policy, decision making) in a way that maximizes rewards in the long term.
  - Unsupervised Learning: the machine only has input, no supervised output  $y$ , nor rewards from its environment
    - ⇒ Build a model of  $x$ ! Build representations of the input!
    - Finding patterns in the data!

# Unsupervised Learning

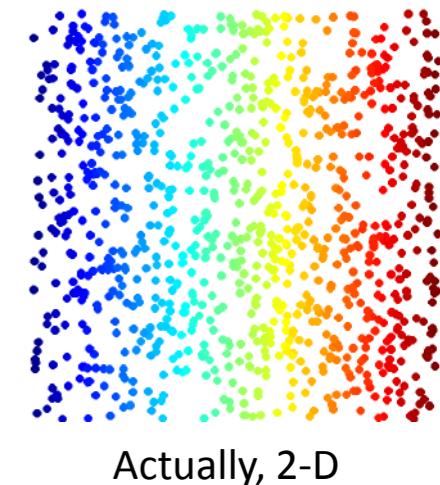
✓ clustering



✓ embedding

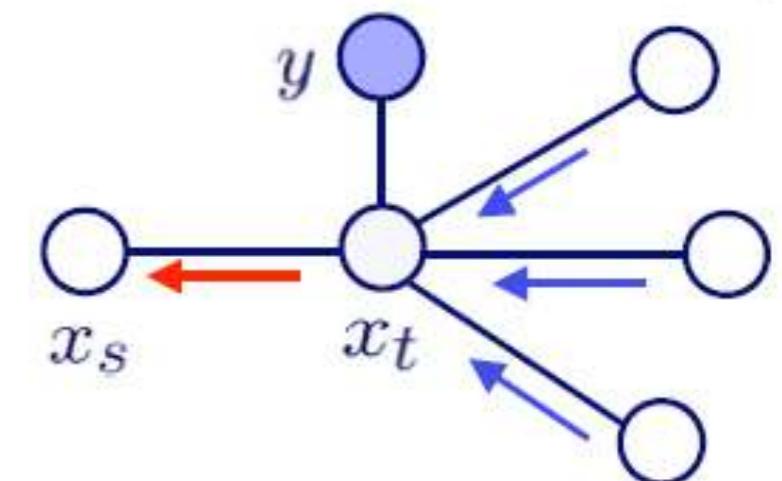


Looks like 3-D



✓ density estimation

✓ finding good explanations  
(hidden causes) of data



- ▶ **Unsupervised Learning: the machine only has input, no supervised output  $y$ , nor rewards from its environment**  
⇒ **Build a model of  $x$ ! Build representations of the input!**  
**Finding patterns in the data!**

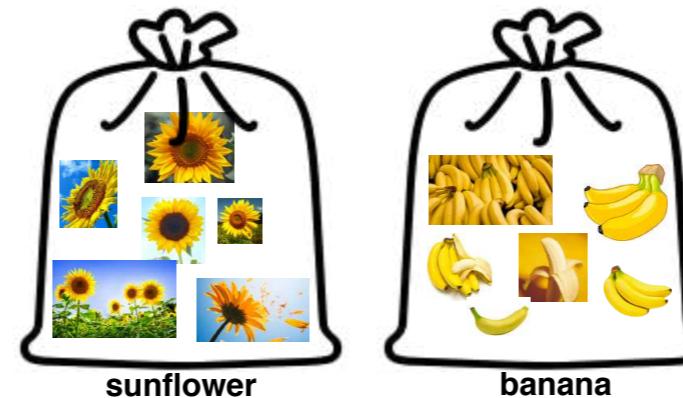
# Unsupervised Learning

- ✓ **finding good explanations  
(hidden causes) of data  
links to *generative model***
- **Unsupervised Learning: the machine only has input,  
no supervised output  $y$ , nor rewards from its environment**  
⇒ **Build a model of  $x$ ! Build representations of the input!  
Finding patterns in the data!**

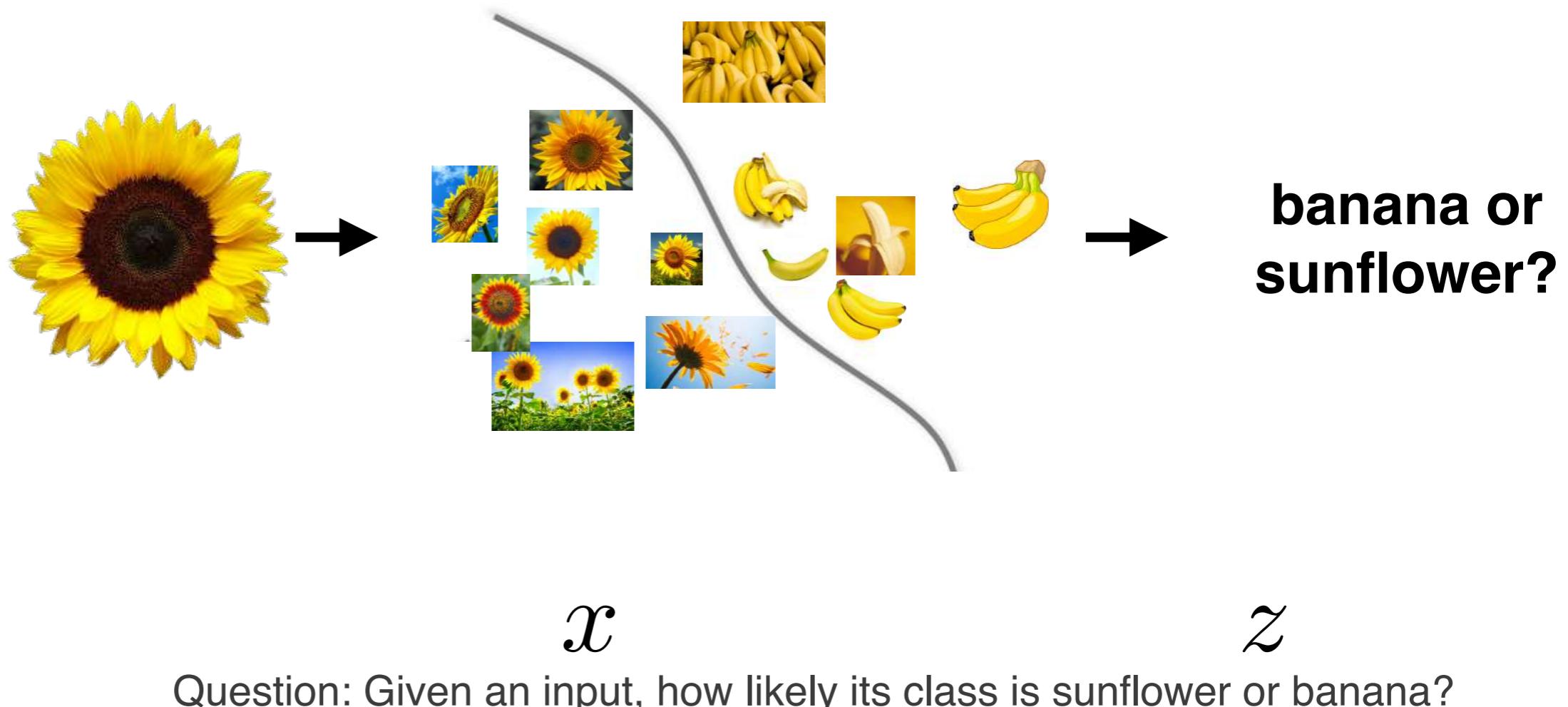
✓ Supervised  
Discriminative versus Generative Models

- **Discriminative**
  - model **posterior**

$$p(z|x)$$



learn a decision boundary  
in a specific feature space



✓ Supervised  
Discriminative versus Generative Models

- **Discriminative**

- model **posterior**

$$p(z|x)$$

- **Generative**

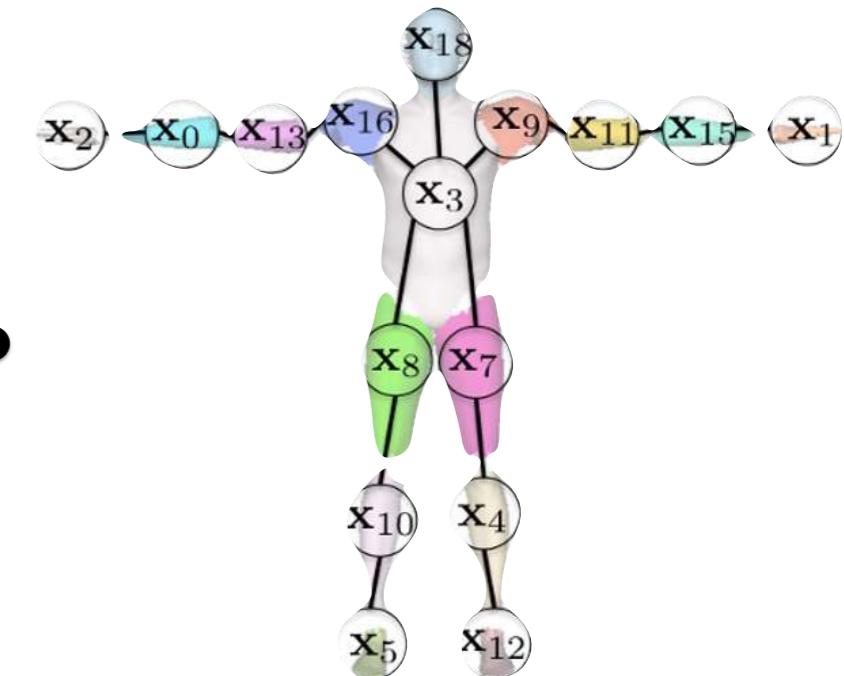
- model **likelihood w/ prior**

$$p(x|z) \cdot p(z)$$

✓ **finding good explanations (hidden causes) of data links to generative model**



$$x|z$$



$$z$$

Question: I know how human pose can be,  
given a pose, how well it fit the body of Mayor Ko?

# Why Generative Models?

- **Generative**
  - model **likelihood w/ prior** $p(x|z) \cdot p(z)$
  - Given data, train a model to generate samples like it

# Why Generative Models?

- **Generative**

- model **likelihood w/ prior**

$$p(x|z) \cdot p(z)$$

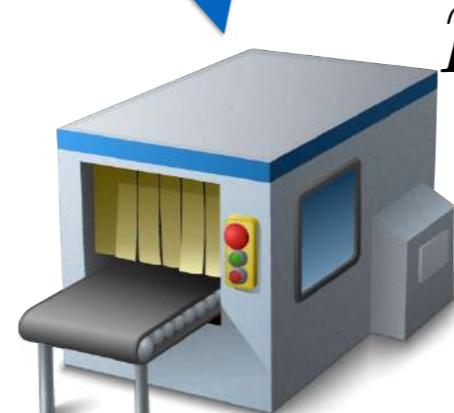
☞ Given data, train a model to generate samples like it

data: images of dog



$p(z)$  **latent variables:**  
**attributes of dogs**  
e.g. **color, species**

$p(x|z)$  **given a setting of attributes, how the dog will look like?**



model

☞ process of data formation



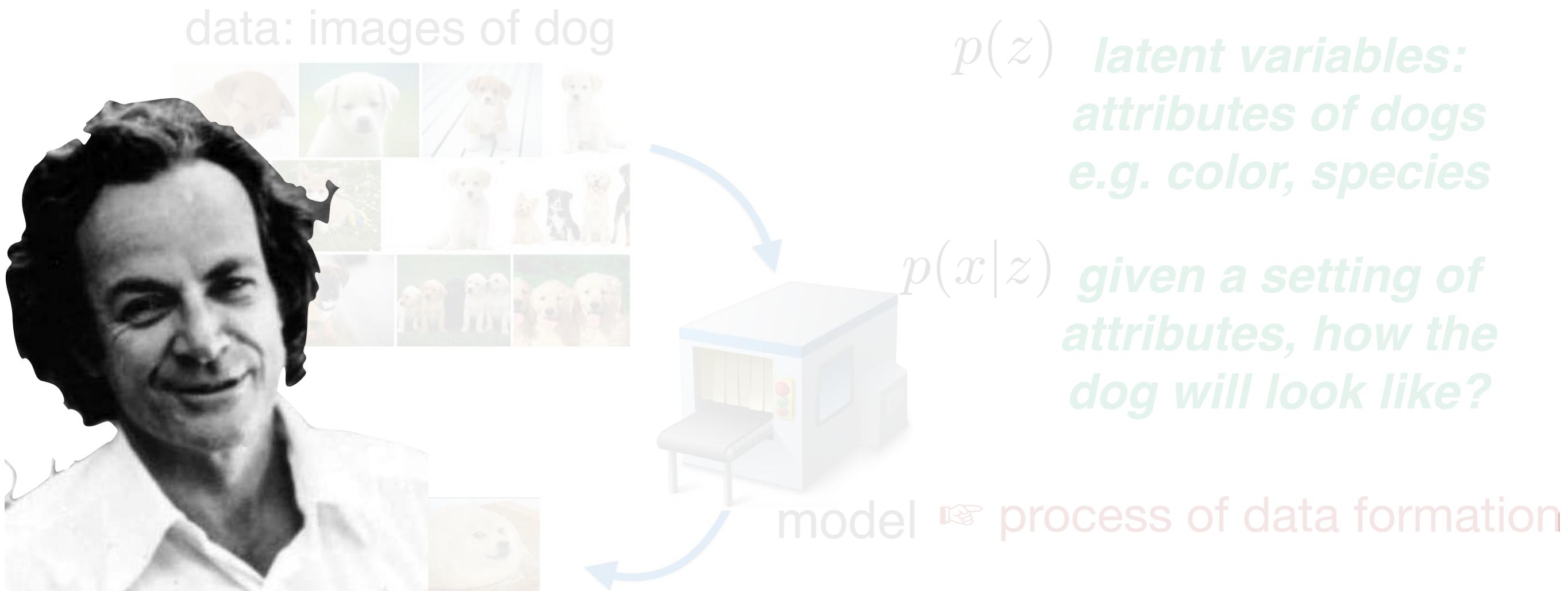
samples



# Why Generative Models?

- **Generative**
  - model **likelihood w/ prior** $p(x|z) \cdot p(z)$

☞ Given data, train a model to generate samples like it



Richard Feynman: “What I cannot create, I do not understand”

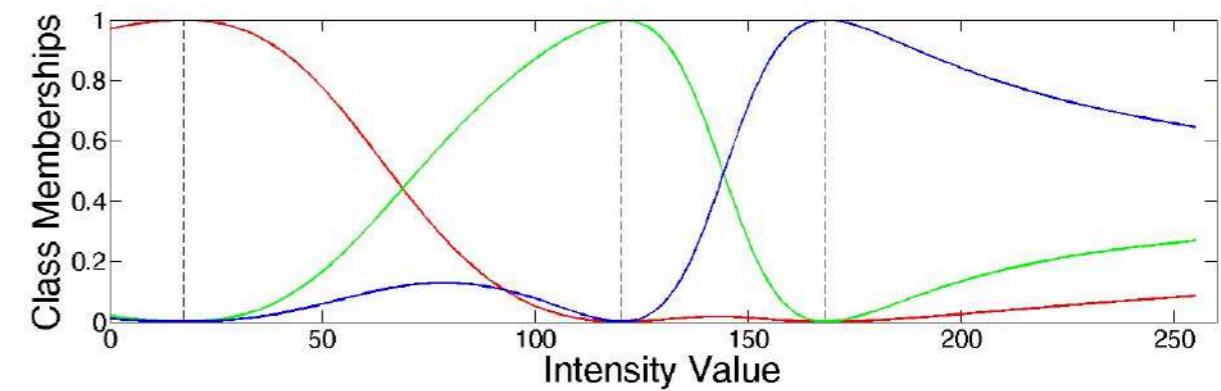
[Ref: <https://openai.com/blog/generative-models/>]

# Clustering

✓ clustering



- Goal: Partition a set  $V$  such that
  - elements in the same subset/cluster are similar
  - elements in distinct subsets are dissimilar



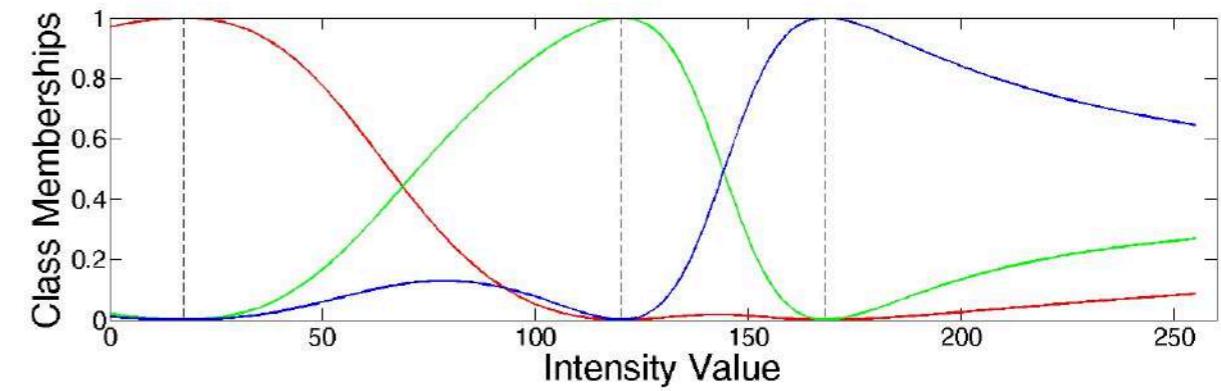
**reminds you anything? similarity? ... evil kernels!**

# Clustering

✓ clustering



- Goal: Partition a set  $V$  such that
  - elements in the same subset/cluster are similar
  - elements in distinct subsets are dissimilar
- Clustering is an optimization problem
  - feasible set: partitions of a set  $V$
  - objective function: depends on the applications
  - if you have no understanding on the objective function, directly use a randomly-chosen clustering method might cause you pain



**reminds you anything? similarity? ... evil kernels!**

**most frequent and open questions: how do you decide #cluster?**

# Most Popular Clustering: K-means

- Objective: denote by  $C_i$  the  $i$ -th cluster (set of points) which is represented by the prototype  $\mu_i$ , the center of the cluster

$$\arg \min_{(C_1, \mu_1), \dots, (C_k, \mu_k)} \sum_{i=1}^k \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

**euclidean norm**

- Aim to find an assignment of data points to clusters, as well as a set of vectors  $\{\mu_k\}$ , such that the sum of squares of the distances of each data point to its closest vector  $\mu_k$  is a minimum

# Most Popular Clustering: K-means

- 1-of- $K$  coding scheme:
  - For each data point  $x_n$ , we introduce a corresponding set of binary indicator variables  $r_{nk} \in \{0, 1\}$ , where  $k=1,\dots,K$
  - If the data point  $x_n$  is assigned to the  $k$ -th cluster, then  $r_{nk} = 1$

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - \mu_k\|^2 \quad \text{distortion measure}$$

# Most Popular Clustering: K-means

- **Lloyd's algorithm** for k-means clustering:
  - initialize centers  $\mu_k$  (e.g. randomly pick  $k$  data points as centers)
  - **do**
    - (1) classify all samples according to closet  $\mu_k$ ,  $k=1,\dots,K$   
**(E-step:** keep  $\mu_k$  fixed, minimize  $J$  with respect to  $r_{nk}$ )
    - (2) re-compute as the mean  $\mu_k$  of the points in cluster  $C_k$  for  $k=1,\dots,K$   
**(M-step:** keep  $r_{nk}$  fixed, minimize  $J$  with respect to  $\mu_k$ )
  - **while** no change in  $\mu_k$ ,  $k=1,\dots,K$
  - **return**  $\mu_1, \dots, \mu_k$
- K-means is a **combinatorial optimization problem**
  - simple iterative algorithm - converges fast but finds only local minimum (which depends on the initialization)

# Most Popular Clustering: K-means

- Determination of  $r_{nk}$ 
  - since  $J$  is a linear function of  $r_{nk}$ , we can optimize for each  $n$  separately

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|x_n - \mu_k\|^2$$

$$r_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_k \|x_n - \mu_k\| \\ 0 & \text{otherwise} \end{cases}$$

**E-step**

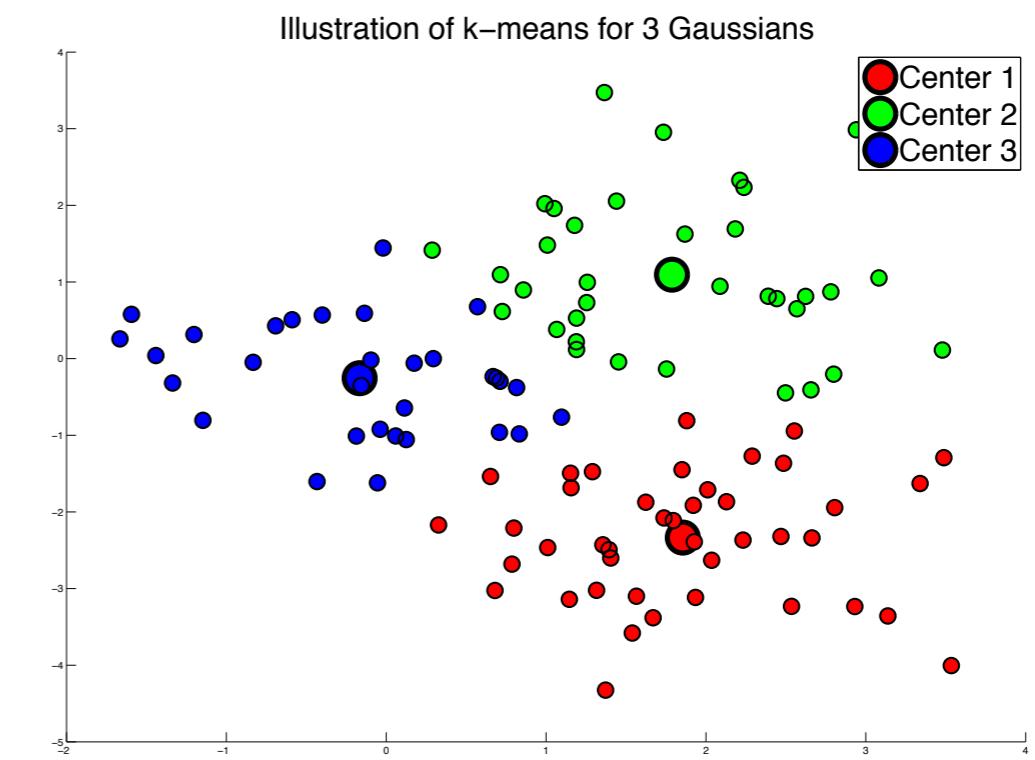
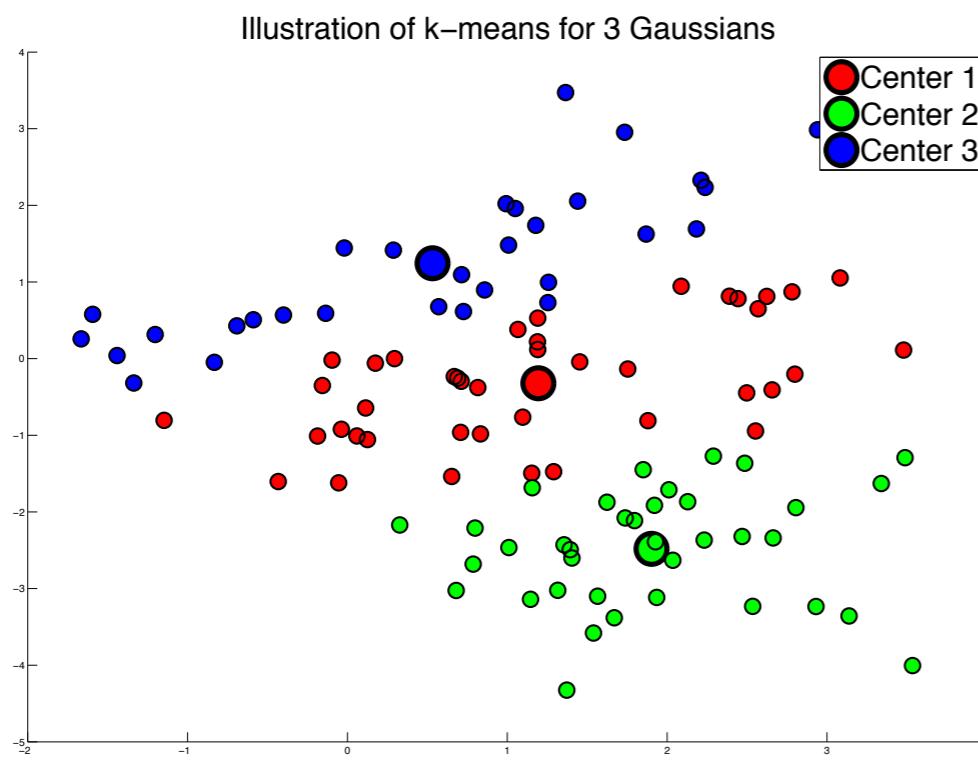
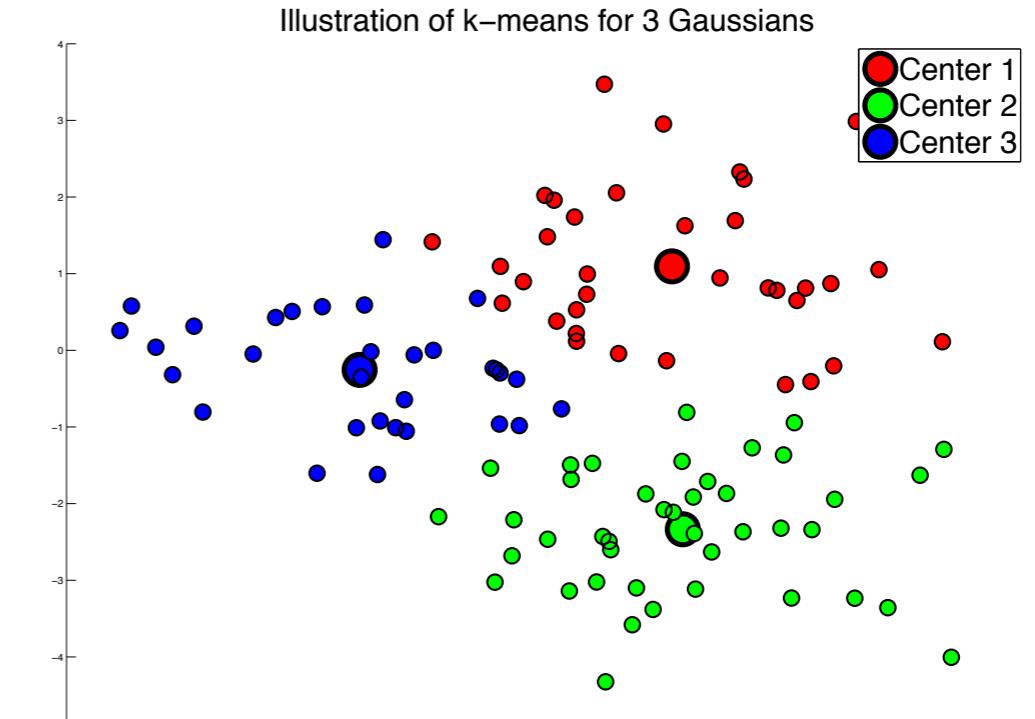
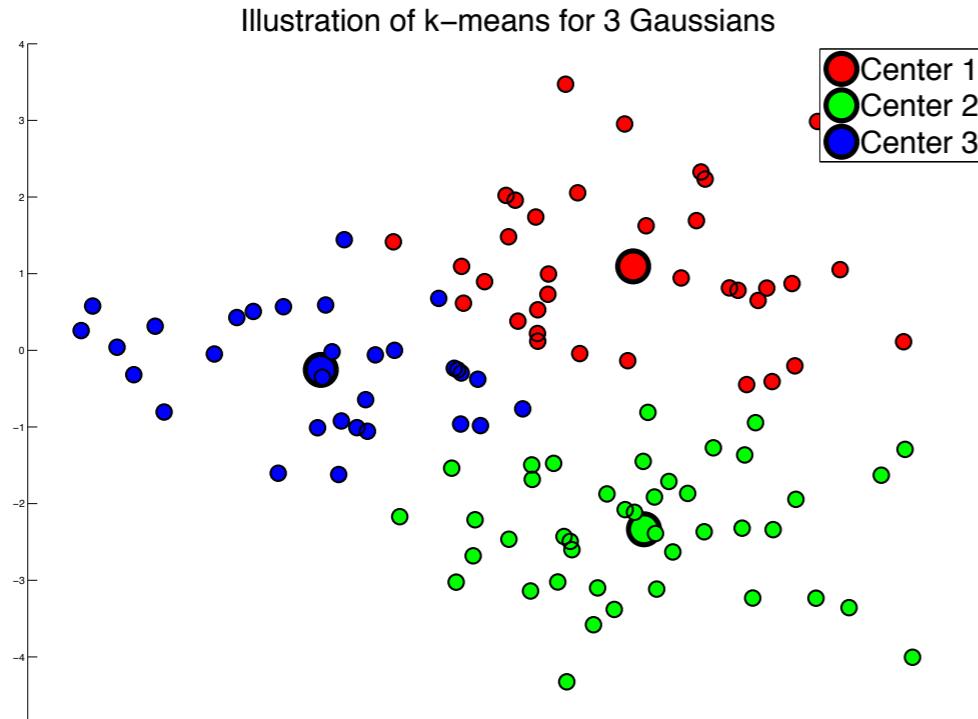
- Determination of  $\mu_k$

$$\frac{\partial J}{\partial \mu_k} = 0 \Rightarrow 2 \sum_{n=1}^N r_{nk} (x_n - \mu_k) = 0 \Rightarrow \mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

**M-step**

reminds you the Gaussian Mixture Model (GMM)?  
here analogously we have K Gaussians in GMM,  
but all these Gaussians are “isotropic”

# Most Popular Clustering: K-means



# Most Popular Clustering: K-means

---

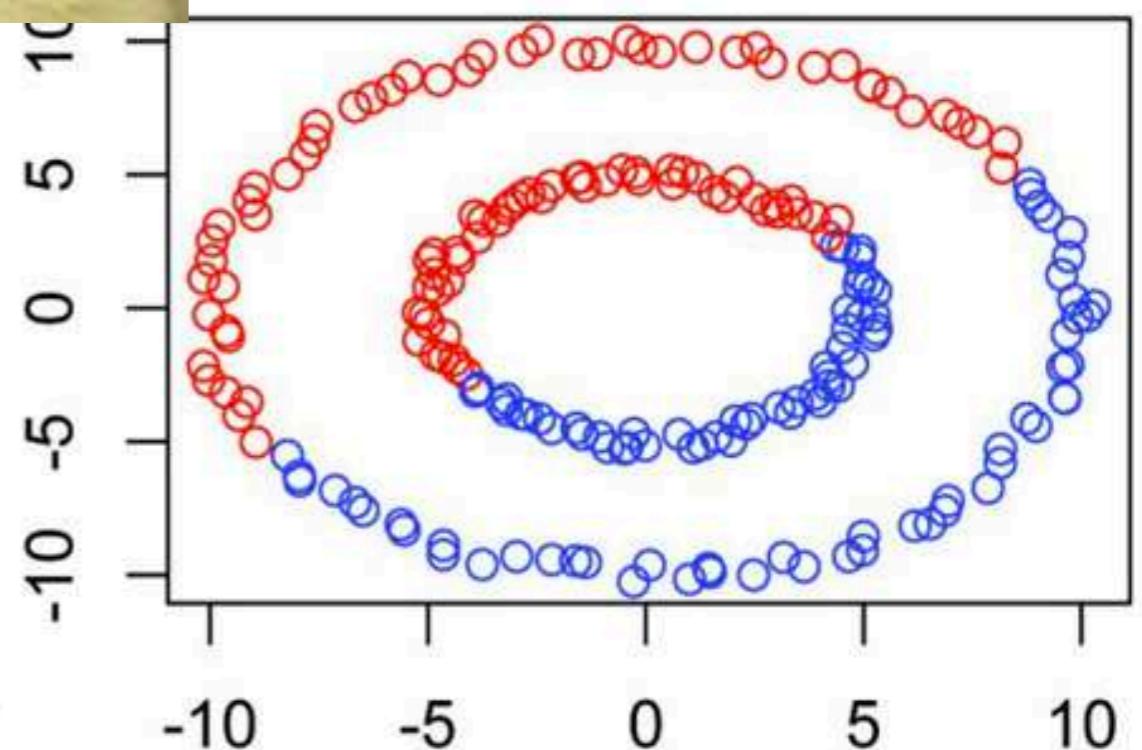
- Potential Problems:
  - Sub-optimality:  
converges to local minimum which depends on the initialization!
  - Choosing the metric (other than euclidean distance?)
  - Choosing  $K$  (the most frequent and open question)

# Most Popular Clustering: K-means

- Potential Problems:
  - Sub-optimality:  
converges to local minimum which depends on the initialization!
  - Choosing the metric (other than euclidean distance?)
  - Choosing  $K$  (the most important question)
  - sometimes problem is  
k-means only can detect clusters  
that are (roughly) linearly separable



k-means



# Kernel K-means

$$\arg \min_{(C_1, \mu_1), \dots, (C_k, \mu_k)} \sum_{i=1}^k \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$$

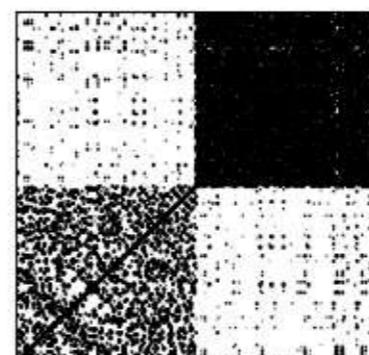
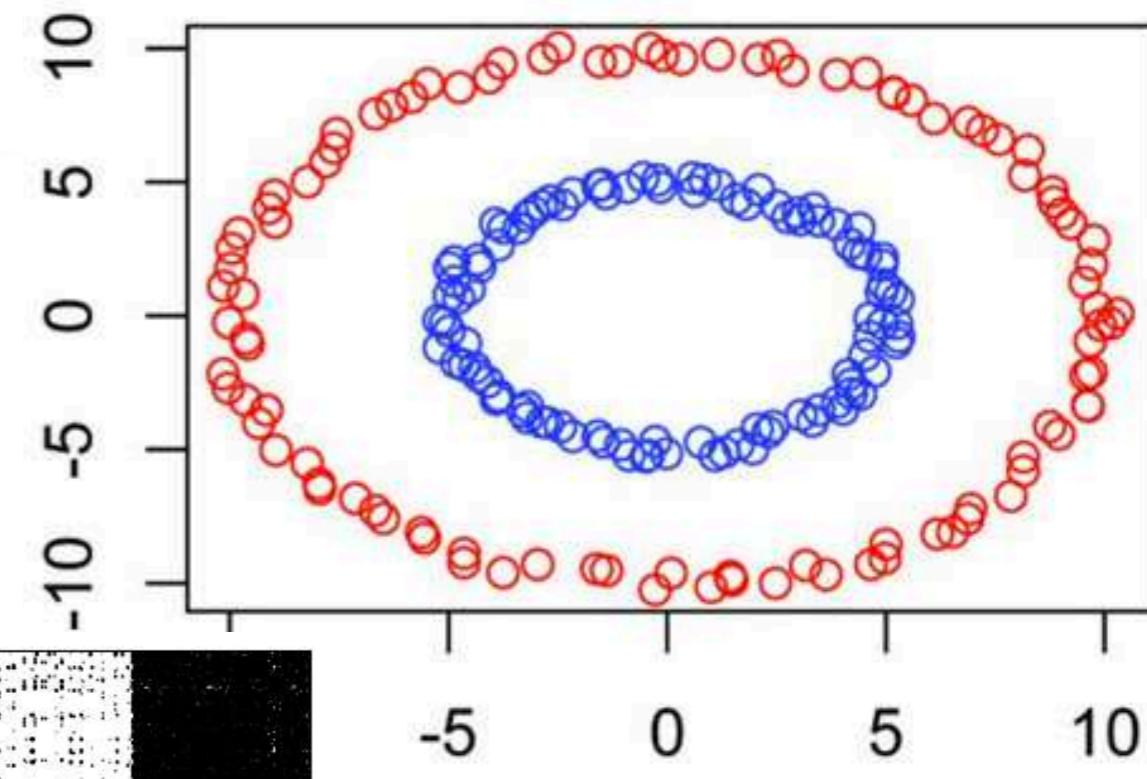
- write each center in kernel space:  
where if the data point  $x_n$  is assigned  
to the  $k$ -th cluster, then  $\alpha_{kn} = 1$
- Now

$$\begin{aligned}\|\phi(x_j) - \mu_k^\phi\| &= \left\| \phi(x_j) - \frac{1}{|C_k|} \sum_{n=1}^N \alpha_{kn} \phi(x_n) \right\| \\ &= \mathbf{k}(x_j, x_j) - \frac{2}{|C_k|} \sum_n \alpha_{kn} \mathbf{k}(x_j, x_n) + \frac{1}{|C_k|^2} \sum_p \sum_q \alpha_{kp} \alpha_{kq} \mathbf{k}(x_p, x_q)\end{aligned}$$

# Kernel K-means

You can first apply an initial clustering, then use its result to reorder your data.

By using the reorder data to compute the Gram matrix, ideally you can see the block diagonal structure if the clusters are uniformly dense which hence provide a good way to estimate the #cluster.



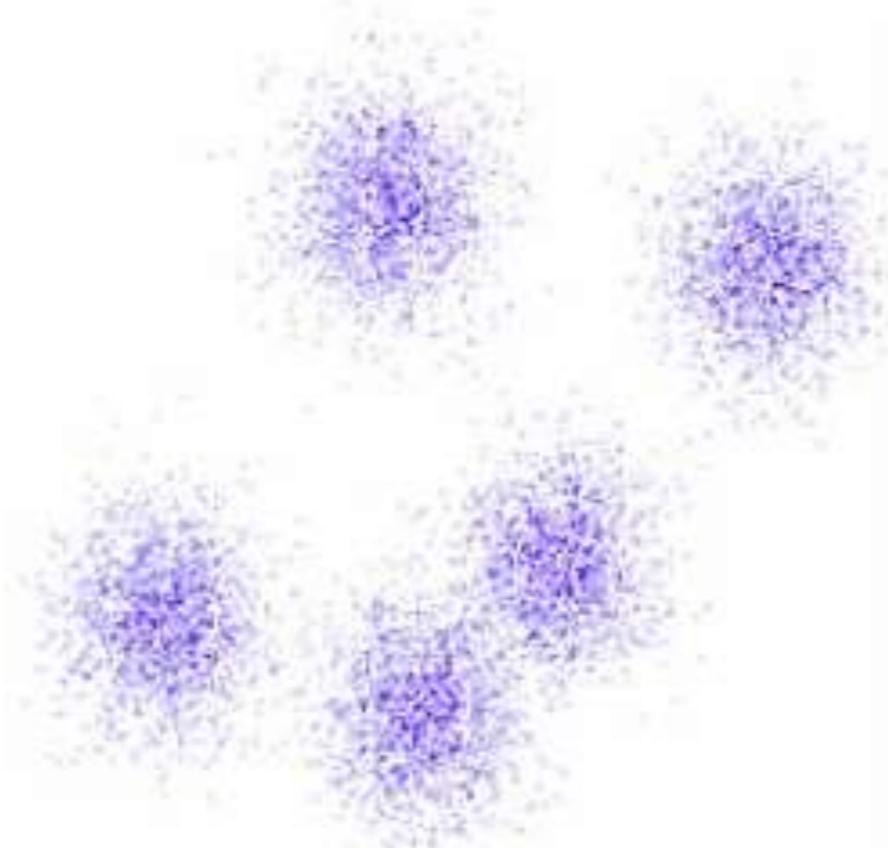
Gram Matrix  
After Reordering

$$\begin{aligned}
 \|\phi(x_j) - \mu_k^\phi\| &= \left\| \phi(x_j) - \sum_{n=1}^N \alpha_{kn} \phi(x_n) \right\| \\
 &= \mathbf{k}(x_j, x_j) - \frac{2}{|C_k|} \sum_n \alpha_{kn} \mathbf{k}(x_j, x_n) + \frac{1}{|C_k|^2} \sum_p \sum_q \alpha_{kp} \alpha_{kq} \underline{\mathbf{k}(x_p, x_q)}
 \end{aligned}$$

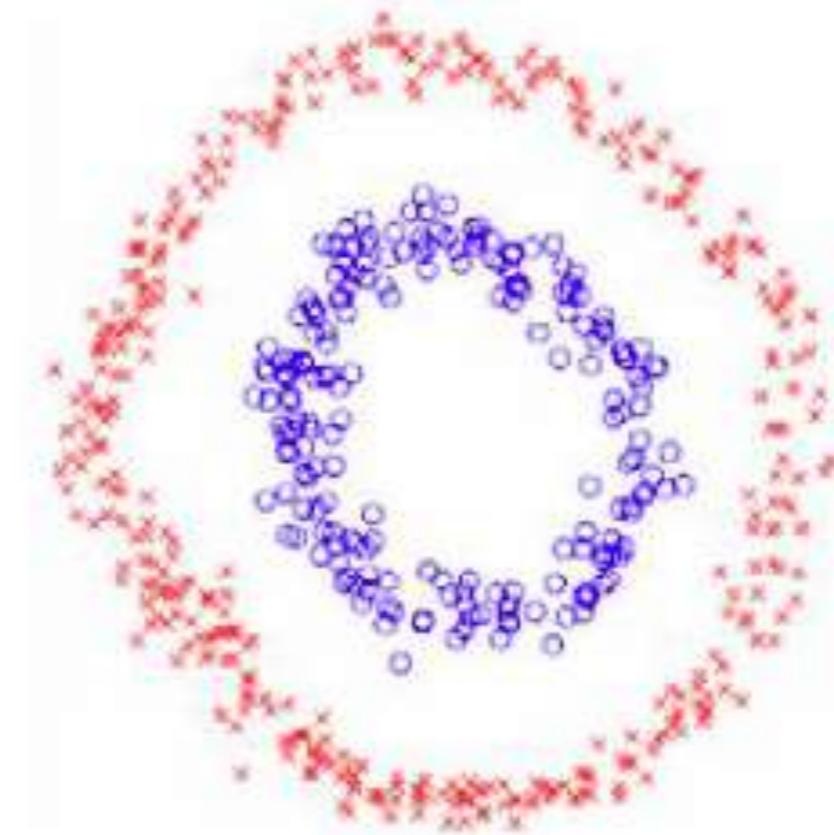
**Gram matrix!**

# Clustering

- Actually there are two main criteria we usually expect for
  - **compactness**: e.g. k-means, mixture models
  - **connectivity**, e.g. spectral clustering (which we are going into)



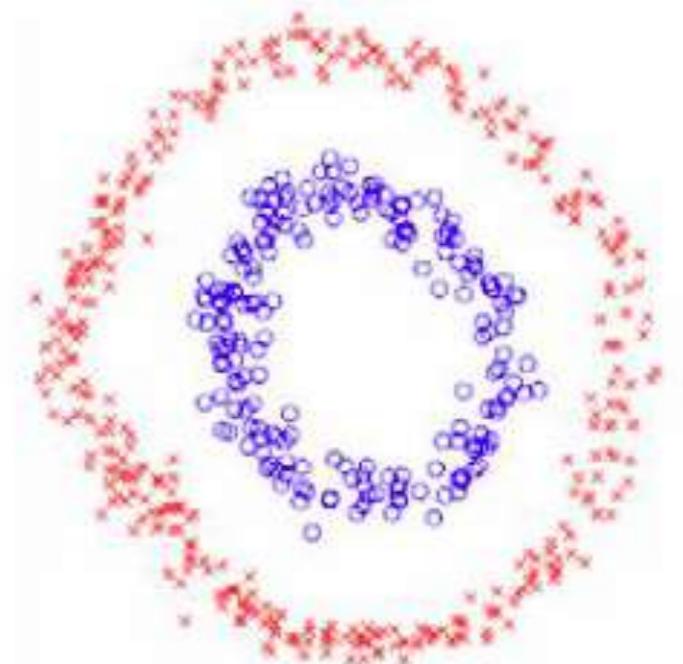
**Compactness**



**Connectivity**

# Spectral Clustering

- First attempts by Donath and Hoffman and Fiedler in 1973
- Popular as it can find clusters of almost arbitrary shape
- Rich theoretical background  
(eigenvectors of a graph Laplacian)

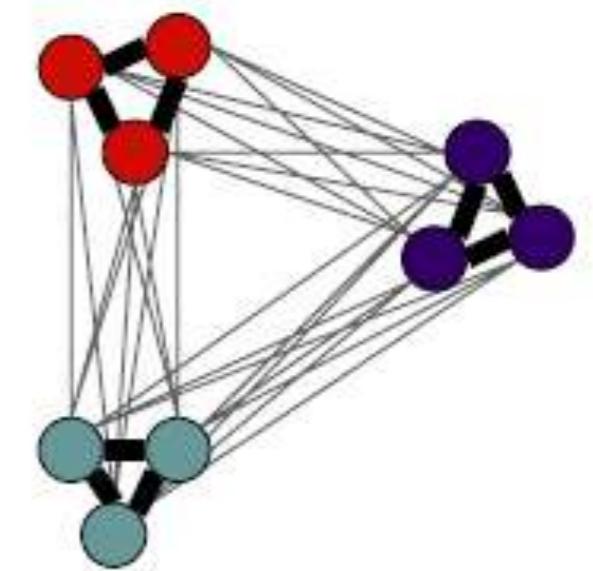
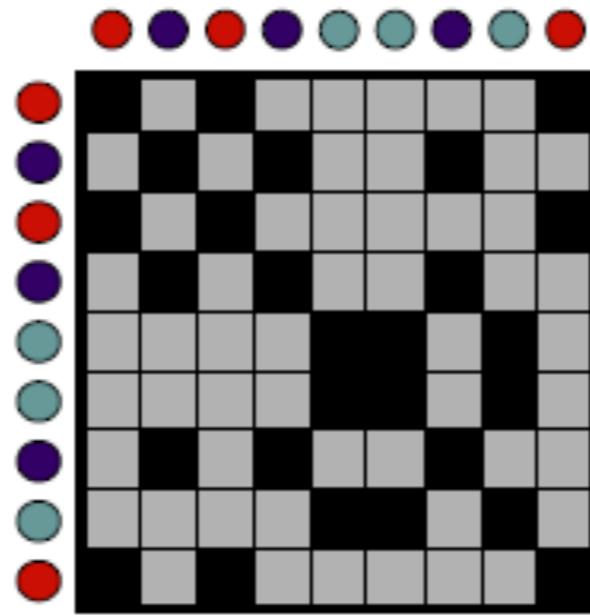
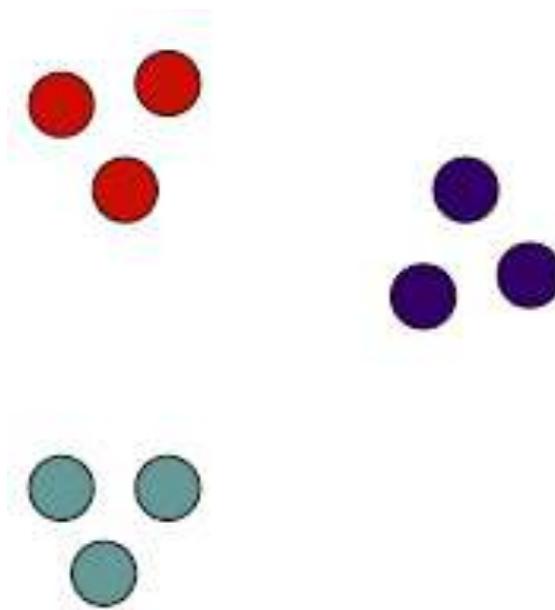


# Start with Graph Clustering

- Goal: Given a set  $V$  of data points  $x_1, \dots, x_n$  and their similarities  $w(x_i, x_j)$ , partition the data into groups so that points in a group are similar and points in different groups are dissimilar
  - edge if similarity > 0**
- We can build a similarity graph:  $G(V, E, W)$  or simply  $G(V, W)$

**vertices/nodes  
(data points)**

**edge weights  
(similarities)**



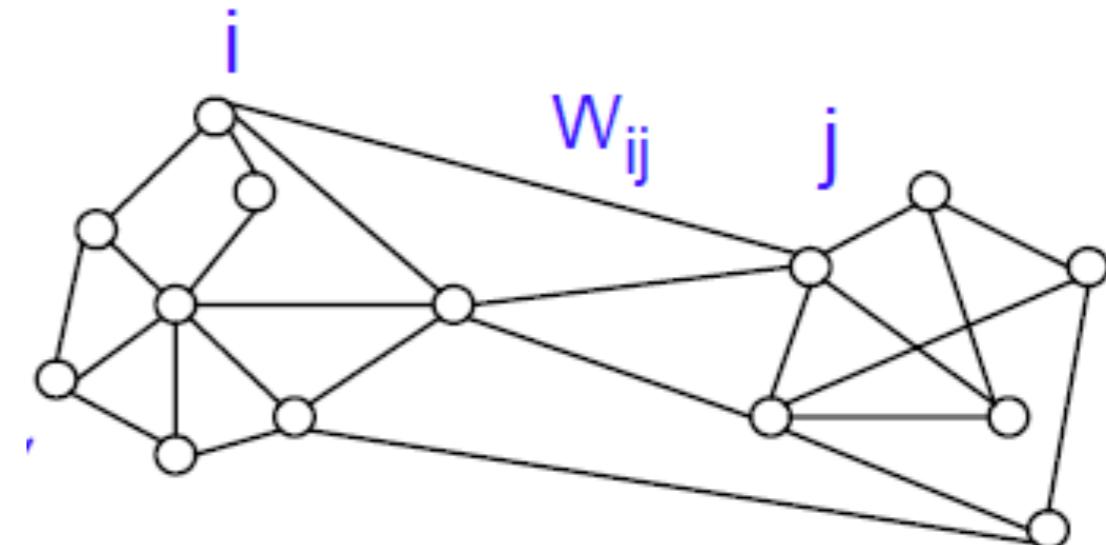
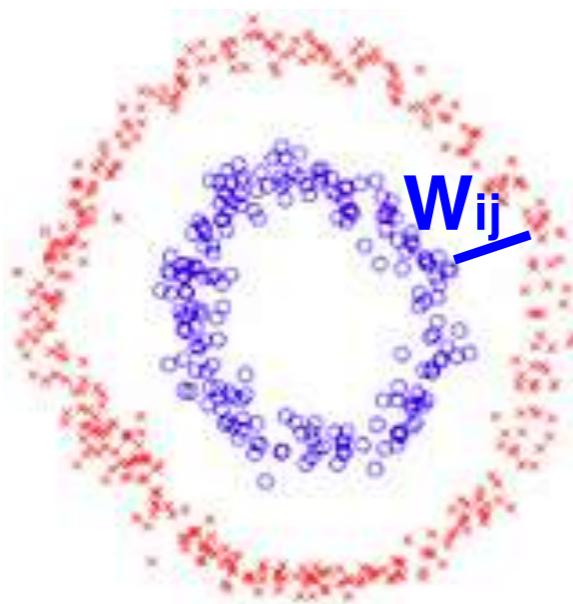
**partition the graph so that edges within a group have large weights  
and edges across groups have small weights**

# Start with Graph Clustering

- similarity graph construction:  
model local neighborhood relations between data points
- E.g. your favorite Gaussian kernel similarity function

$$W_{i,j} = e^{\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

control size of neighborhood

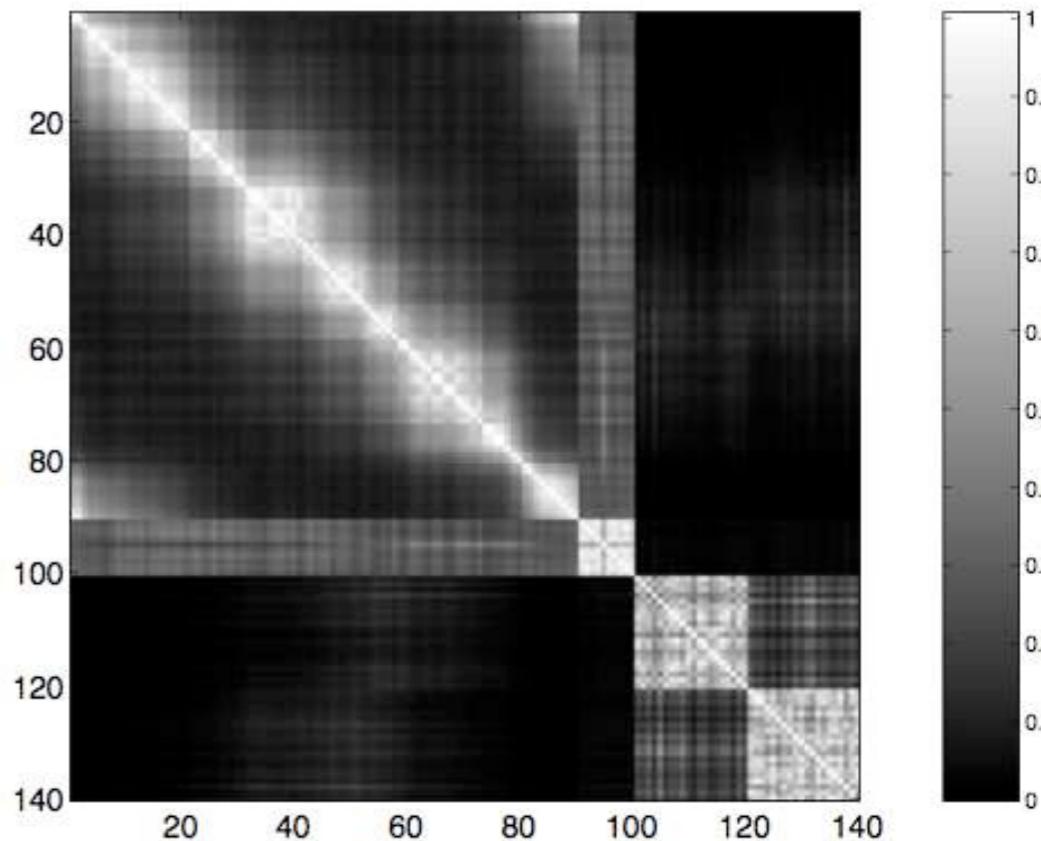
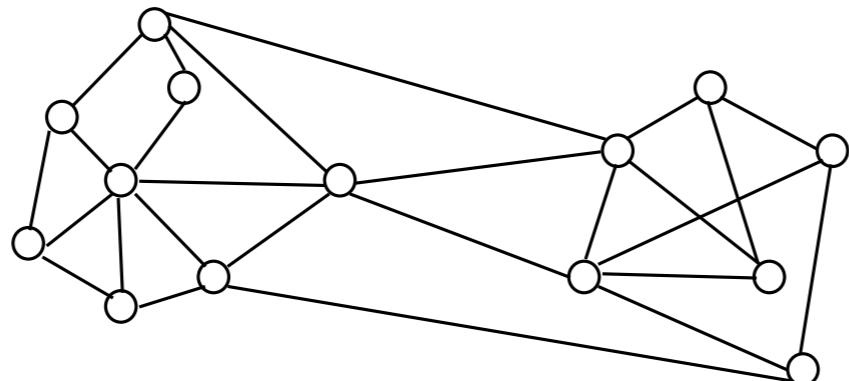


Data clustering

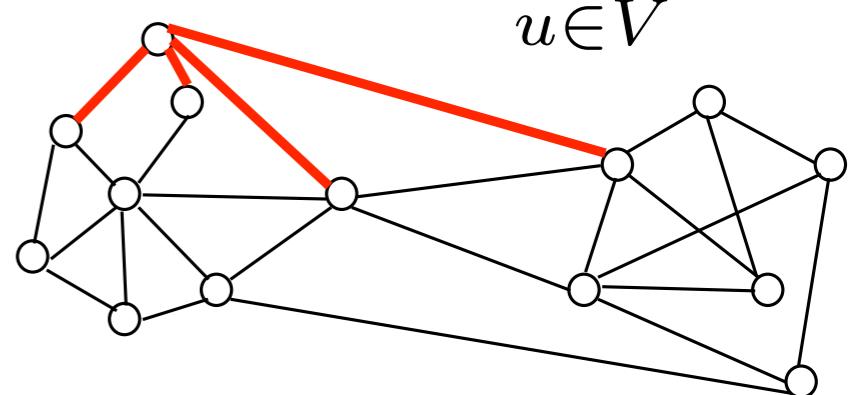
# Start with Graph Clustering

- Goal: Given a set  $V$  of data points  $x_1, \dots, x_n$  and their similarities  $w(x_i, x_j)$ , partition the data into groups so that points in a group are similar and points in different groups are dissimilar  
**symmetric matrix,  
all elements  $> 0$  if  
RBF kernel is used**
- We can build a similarity **graph**:  $G(V, W)$
- We also define for every node  $v \in V$ , its **degree**:  $d_v := \sum_{u \in V} W_{vu}$   
**summation of weights  
to all other nodes**
- The degree matrix  $D$  can be defined by:  
for any  $v, u \in V$      $D_{v,u} := d_v \delta_{vu}$

# Start with Graph Clustering



- We can build a similarity **graph**:  $G(V, W)$
- We also define for every node  $v \in V$ , its **degree**:  $d_v := \sum_{u \in V} W_{vu}$
- The degree matrix  $D$  can be defined by:  
for any  $v, u \in V$      $D_{v,u} := d_v \delta_{vu}$

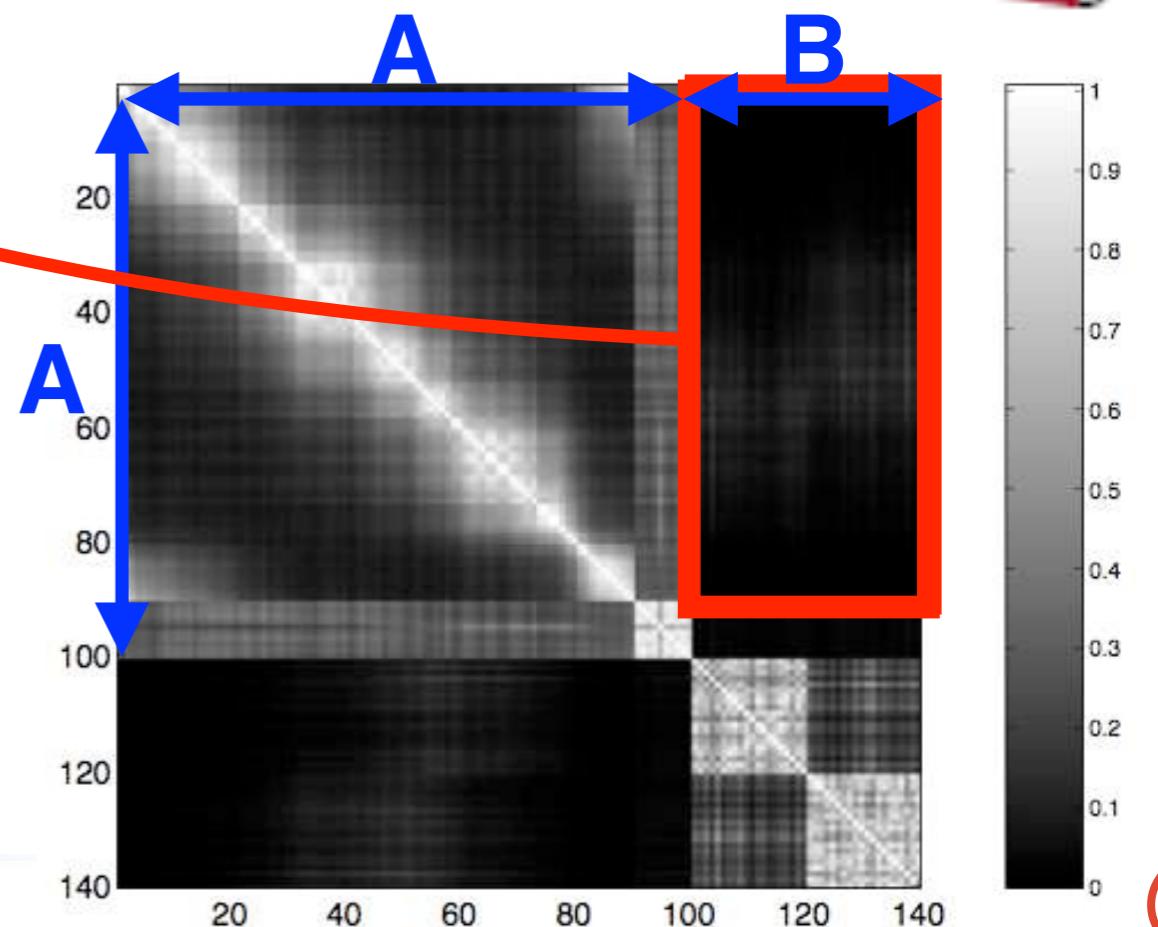
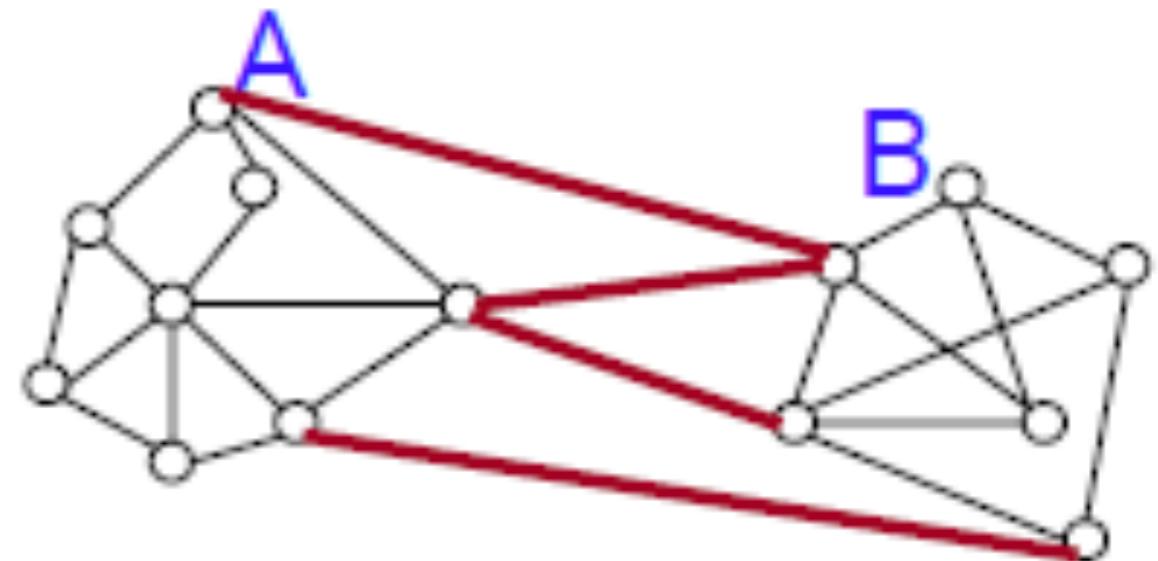


only count neighbors

# First Consider Simple Case

- Partition graph into two sets A and B such that
  - weights of edges connecting vertices in A to vertices in B is minimum

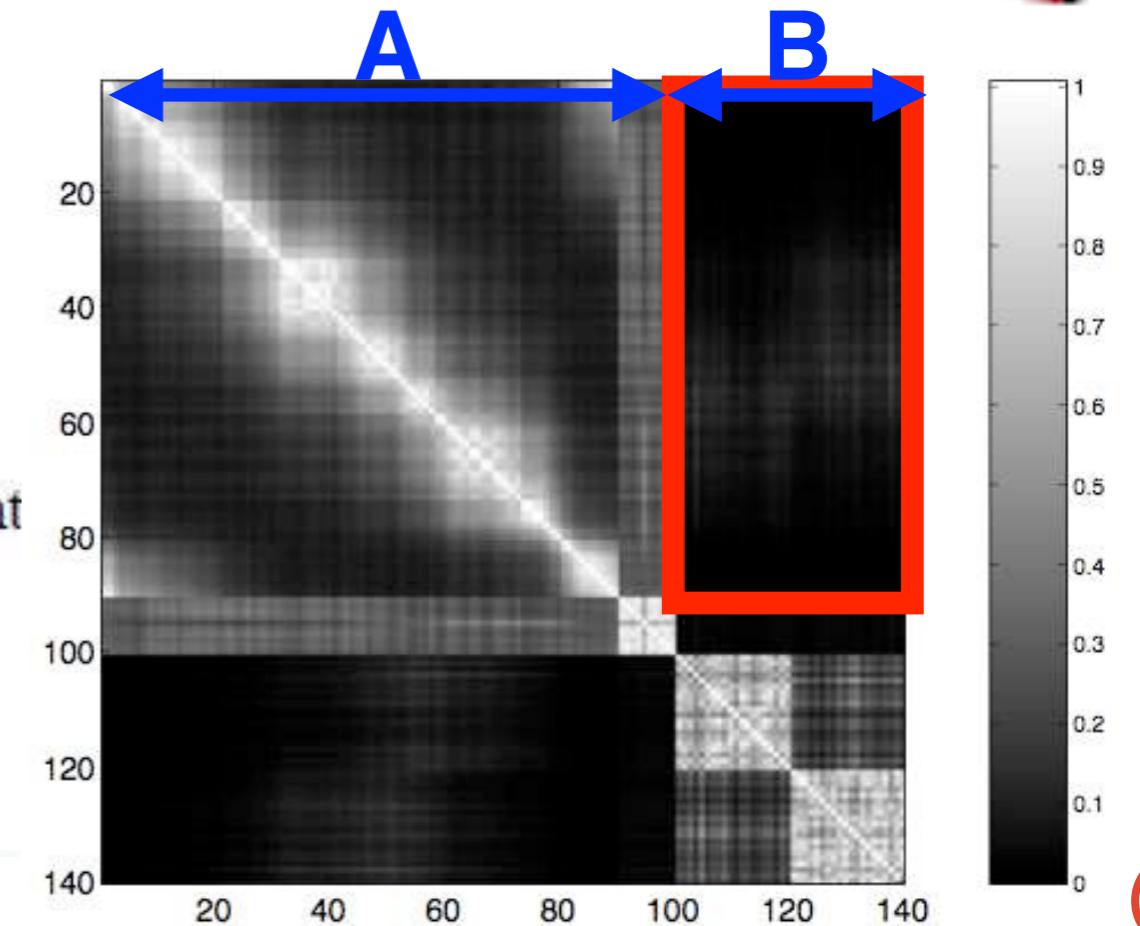
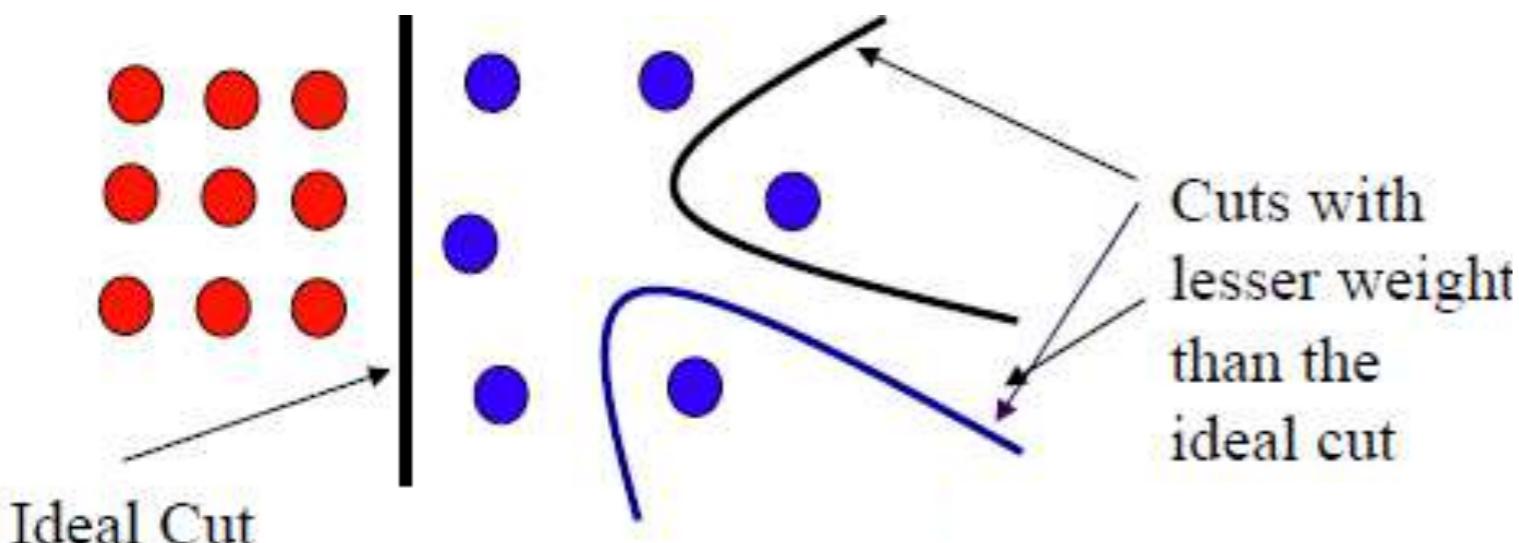
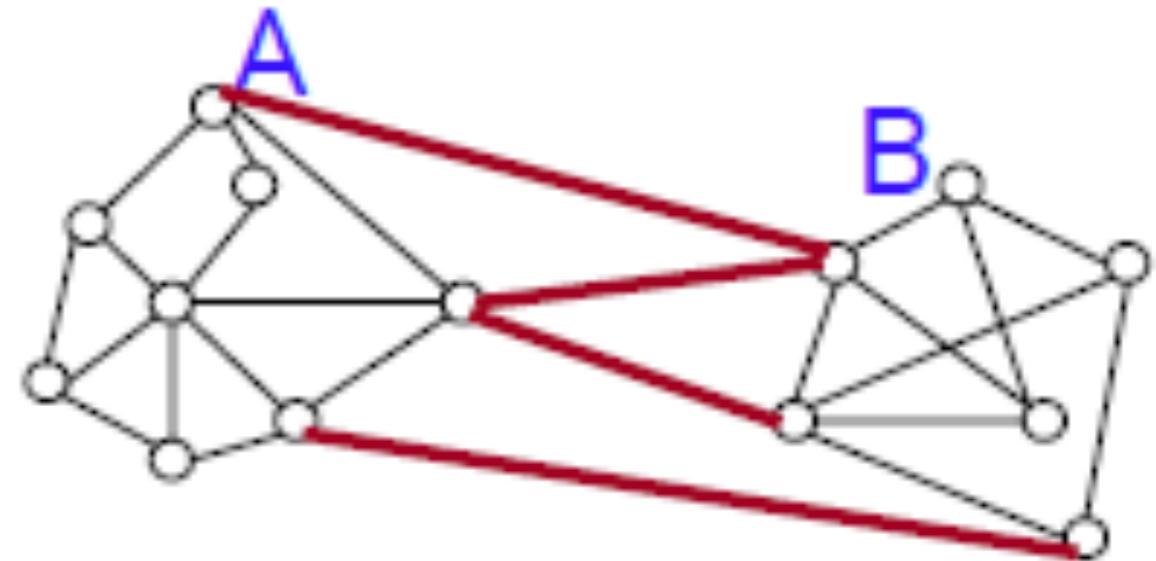
$$\text{cut}(A, B) := \sum_{i \in A, j \in B} W_{i,j}$$



# First Consider Simple Case

- Partition graph into two sets A and B such that
  - weights of edges connecting vertices in A to vertices in B is minimum

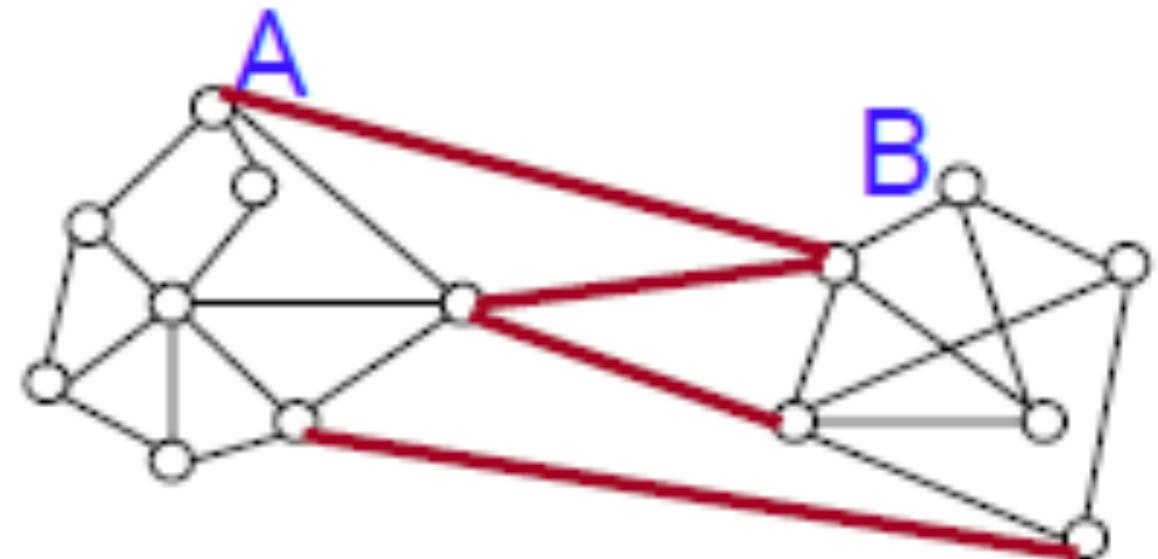
$$\text{cut}(A, B) := \sum_{i \in A, j \in B} W_{i,j}$$



# First Consider Simple Case

- Partition graph into two sets A and B such that
  - weights of edges connecting vertices in A to vertices in B is minimum
  - size of A and B are very similar**

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



- Normalized Cut:**

$$\text{Ncut}(A, B) := \text{cut}(A, B) \left( \frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)} \right) \quad \text{vol}(A) = \sum_{i \in A} d_i$$

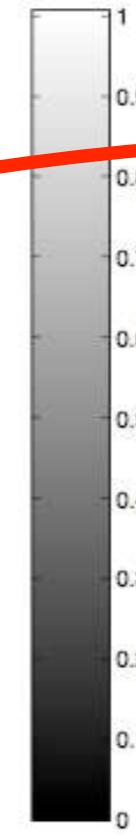
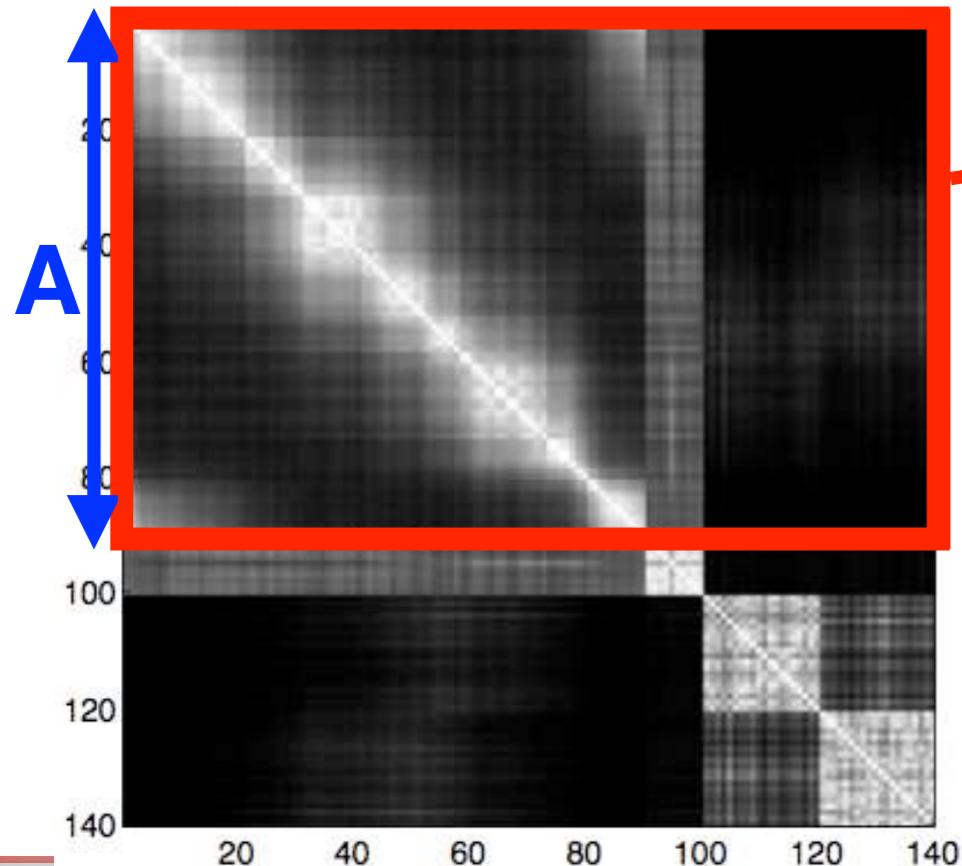
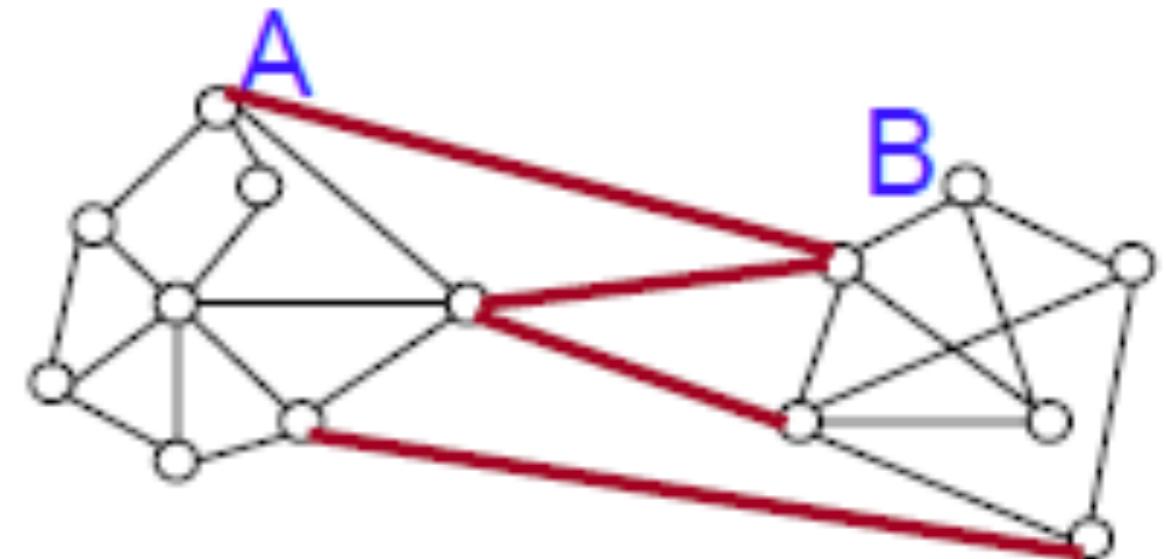
- Ratio Cut**

$$\text{Rcut}(A, B) := \text{cut}(A, B) \left( \frac{1}{|A|} + \frac{1}{|B|} \right) \quad |A| : \# \text{ of nodes in } A$$

# First Consider Simple Case

- Partition graph into two sets A and B such that
  - weights of edges connecting vertices in A to vertices in B is minimum
  - size of A and B are very similar**

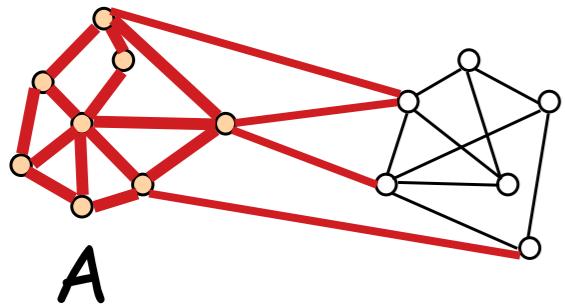
$$\text{cut}(A, B) := \sum_{i \in A, j \in B} W_{i,j}$$



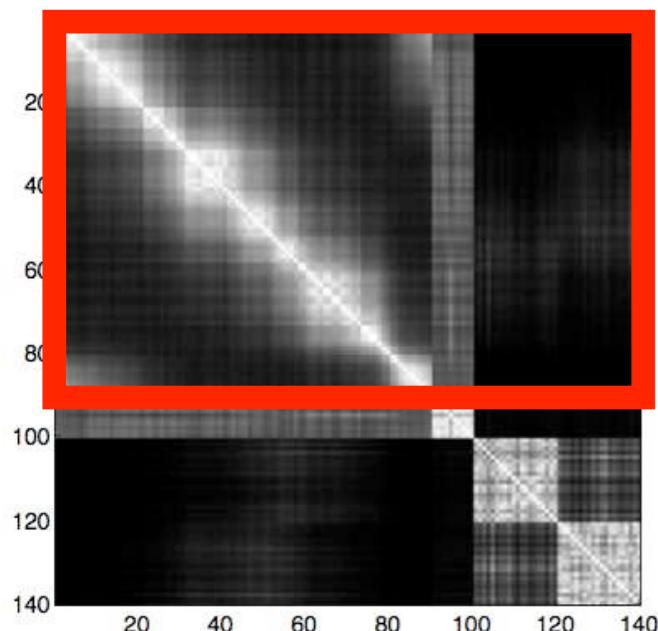
$$\text{vol}(A) = \sum_{i \in A} d_i$$

$|A|$  : # of nodes in A

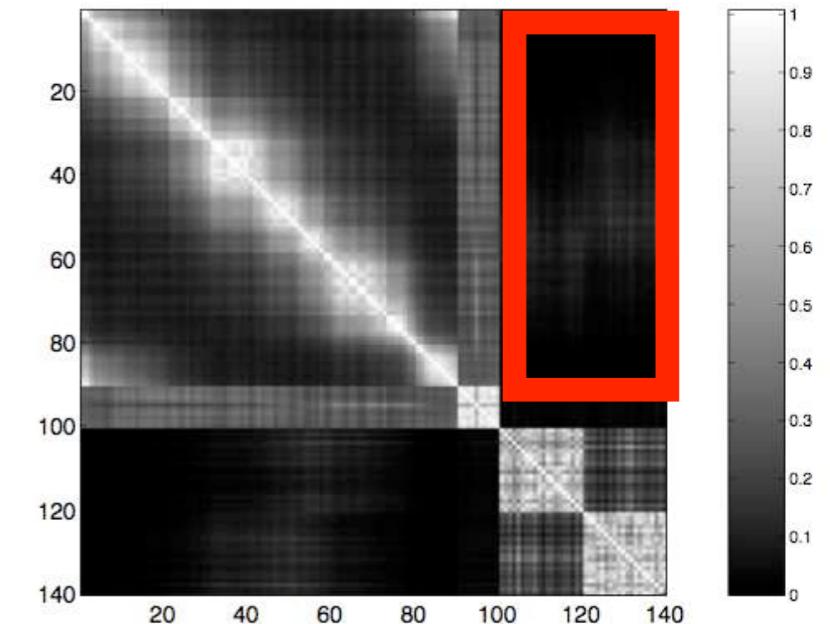
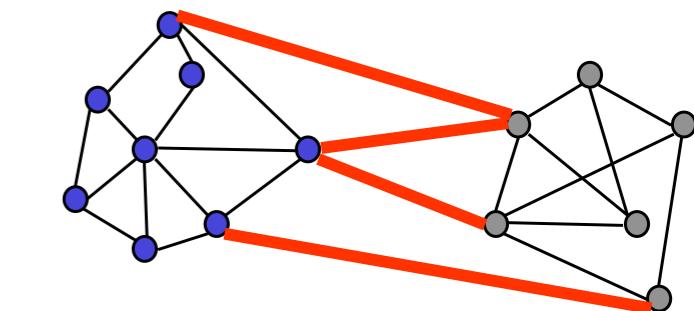
# Simple Visualization



A

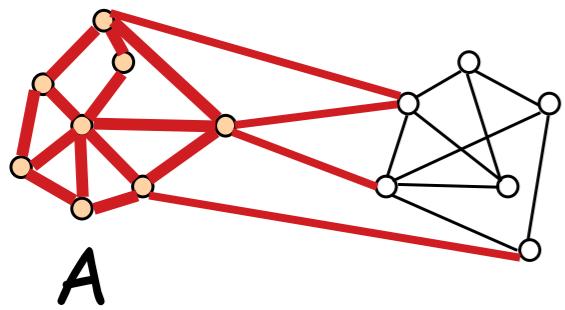


$\text{vol}(A)$

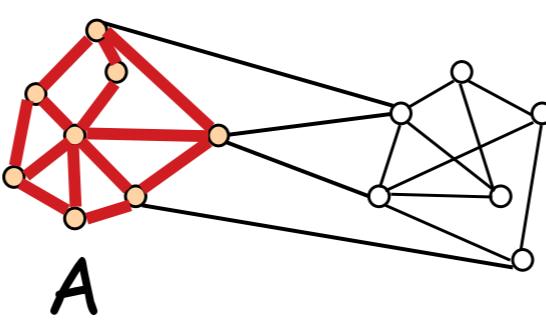


$\text{cut}(A, B)$   
**what you want  
to minimize**

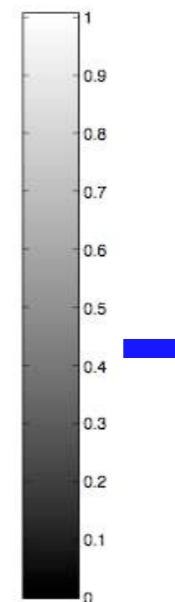
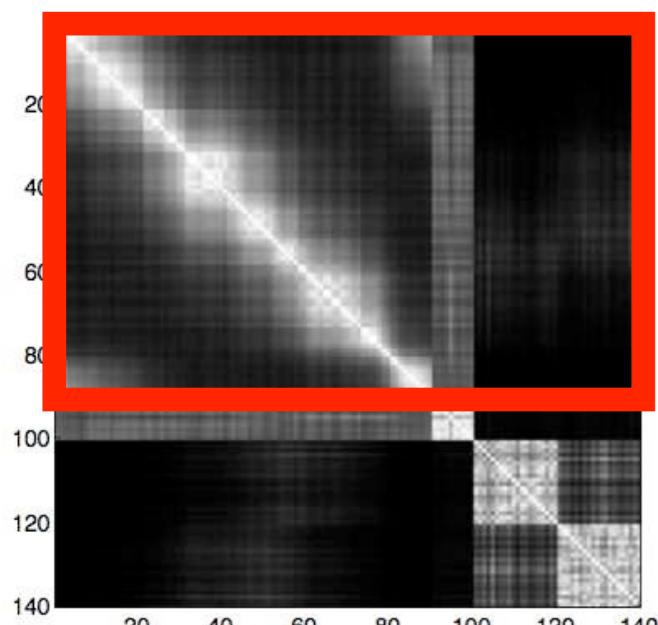
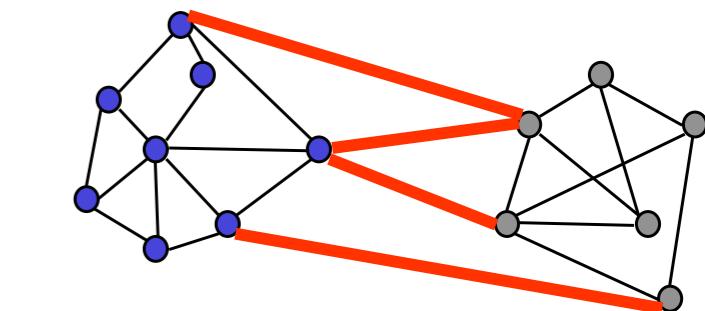
# Simple Visualization



-



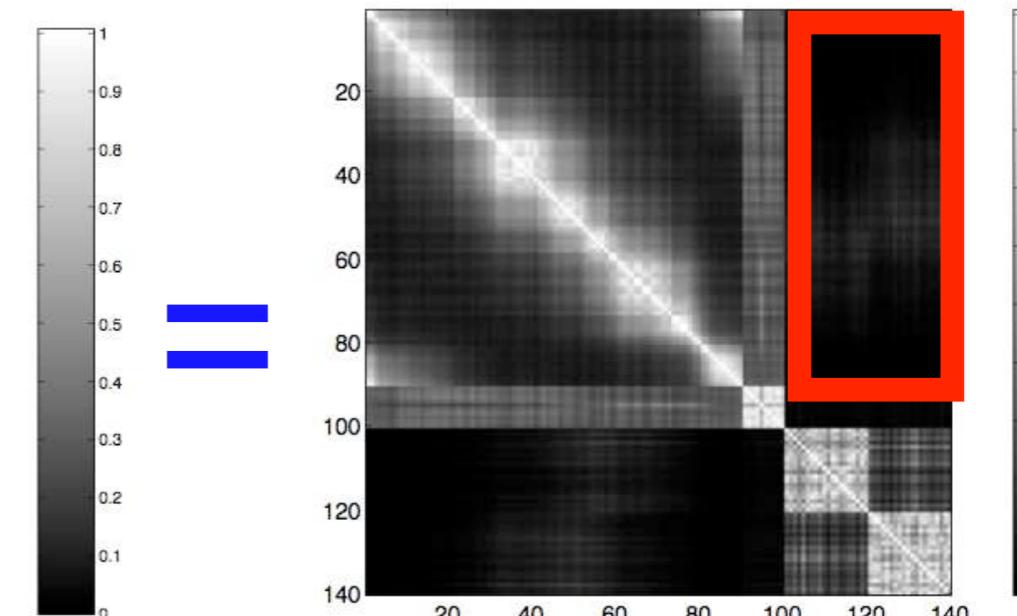
=



$\text{vol}(A)$

$\text{association}(A)$

$\text{cut}(A, B)$   
**what you want  
to minimize**



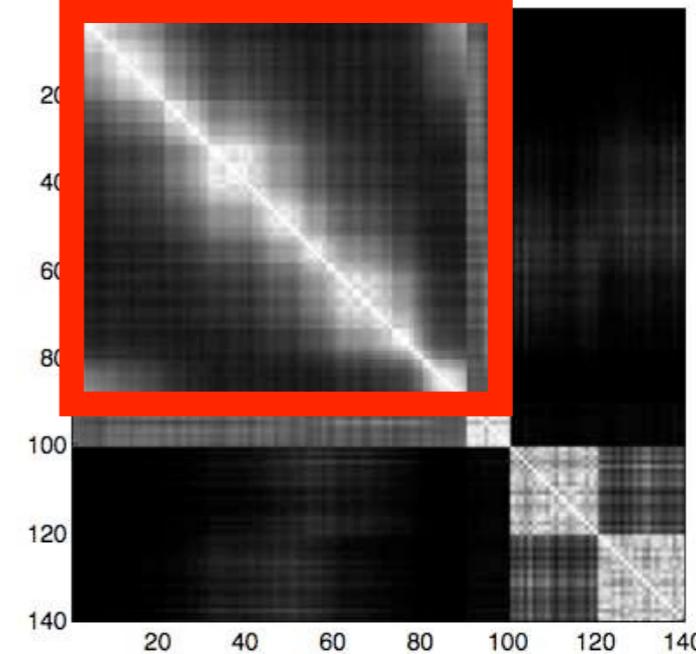
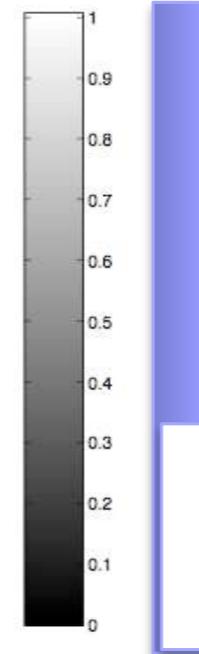
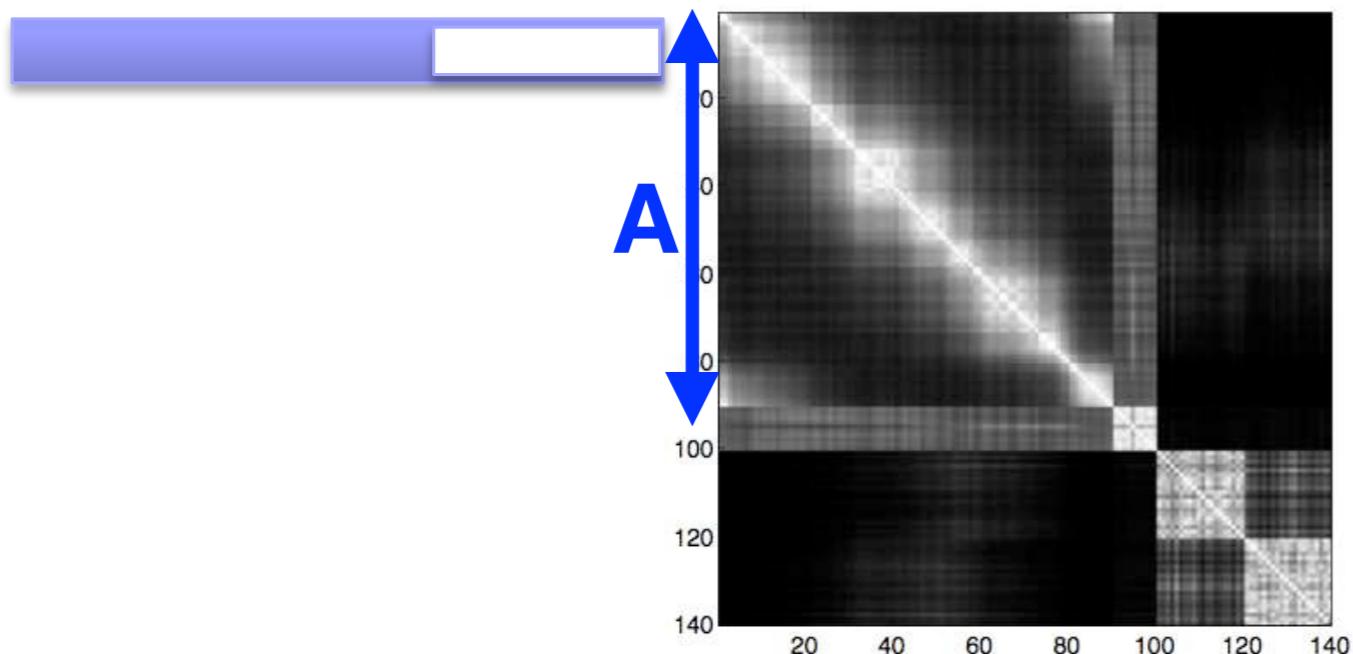
# Simple Visualization

binary vector that's  
an indicator on set A

$$\mathbf{f}^\top$$

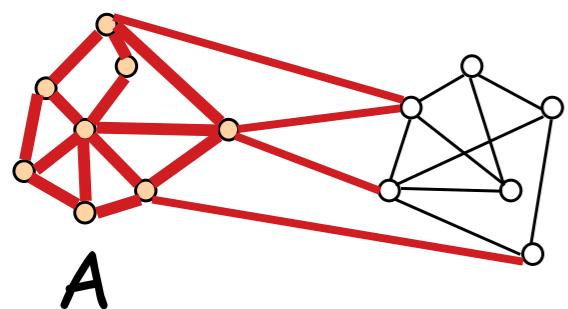
$$W$$

$$\mathbf{f}$$

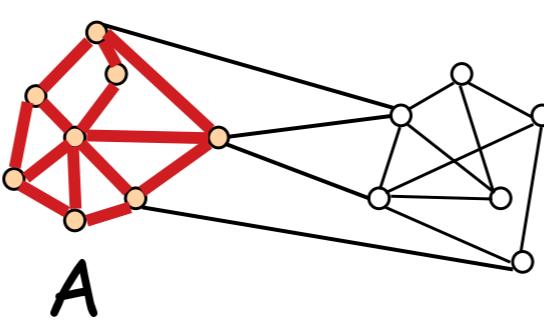


$$\text{association}(A)$$

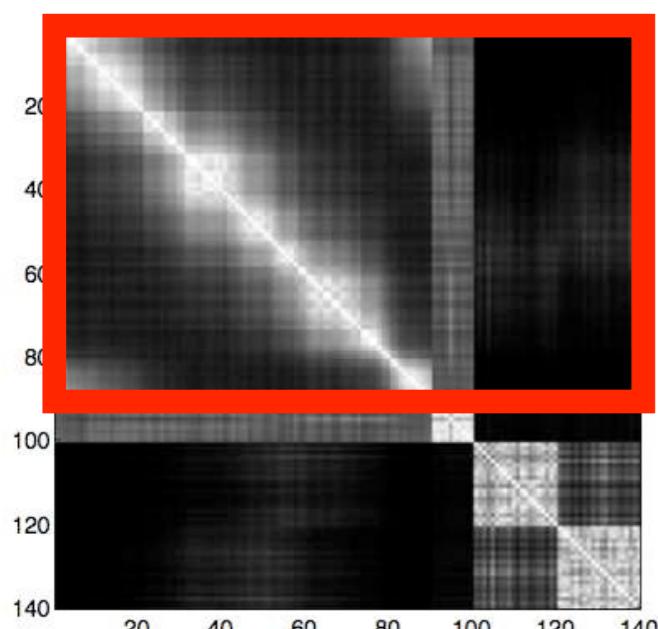
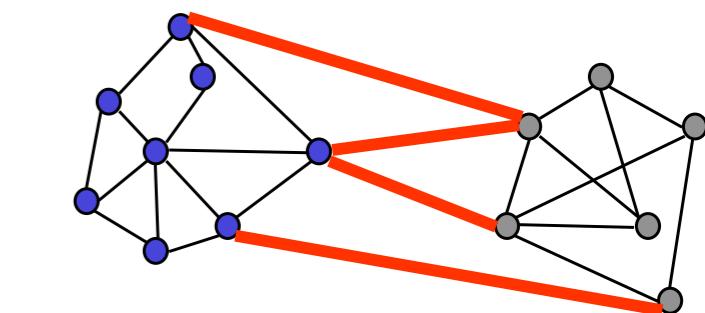
# Simple Visualization



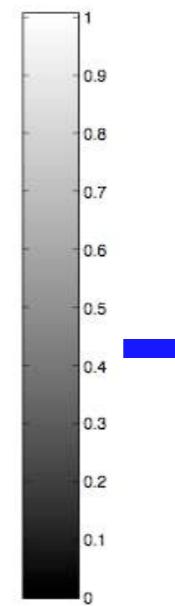
-



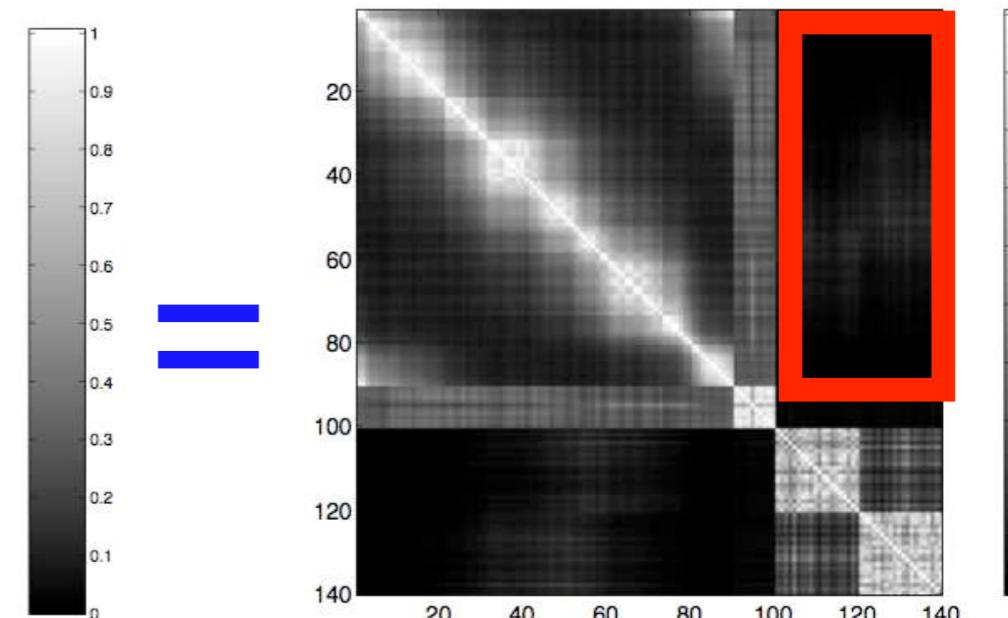
=



$$\text{vol}(A)$$
$$\mathbf{f}^\top D \mathbf{f}$$



$$\text{association}(A)$$
$$\mathbf{f}^\top W \mathbf{f}$$



$$\text{cut}(A, B)$$
$$\mathbf{f}^\top (D - W) \mathbf{f}$$

# Simple Visualization

define a special matrix:  $L = D - W$

**Graph Laplacian**

$$\text{vol}(A) \\ \mathbf{f}^\top D \mathbf{f}$$

$$\text{association}(A) \\ \mathbf{f}^\top W \mathbf{f}$$

$$\text{cut}(A, B) \\ \mathbf{f}^\top (D - W) \mathbf{f}$$

**Proposition 1 (Properties of  $L$ )** *The matrix  $L$  satisfies the following properties:*

1. *For every vector  $f \in \mathbb{R}^n$  we have*

$$f' L f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2.$$

2.  *$L$  is symmetric and positive semi-definite.*

3. *The smallest eigenvalue of  $L$  is 0, the corresponding eigenvector is the constant one vector  $\mathbf{1}$ .*

4.  *$L$  has  $n$  non-negative, real-valued eigenvalues  $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ .*

*Proof.*

Part (1): By the definition of  $d_i$ ,

$$\begin{aligned} f' L f &= f' D f - f' W f = \sum_{i=1}^n d_i f_i^2 - \sum_{i,j=1}^n f_i f_j w_{ij} \\ &= \frac{1}{2} \left( \sum_{i=1}^n d_i f_i^2 - 2 \sum_{i,j=1}^n f_i f_j w_{ij} + \sum_{j=1}^n d_j f_j^2 \right) = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2. \end{aligned}$$

Part (2): The symmetry of  $L$  follows directly from the symmetry of  $W$  and  $D$ . The positive semi-definiteness is a direct consequence of Part (1), which shows that  $f' L f \geq 0$  for all  $f \in \mathbb{R}^n$ .

Part (3): Obvious.

Part (4) is a direct consequence of Parts (1) - (3). □

# More Understanding on Graph Laplacian

- Number of connected components and the spectrum of  $L=D-W$
- First assume there is only one component (fully-connected)
  - $Lu = \lambda u$  eigen-problem
  - $L\mathbf{1}_n = \mathbf{0}$ ,  $\lambda_1 = 0$  is the smallest eigenvalue, where *one vector*  $\mathbf{1}_n = [1 \dots 1]^\top$  (remember  $D$  contains sum over rows?)
  - $0 = u^\top Lu = \sum w_{ij}(u(i)-u(j))^2$
  - if any two vertices are connected by a path ( $w_{ij} > 0$ ), then  $\mathbf{u} = [u(1) \dots u(n)]^\top$  needs to be constant at all vertices such that the quadratic form (related to  $u(i)-u(j)$  term) vanishes. Therefore, a graph with one connected component has the constant vector  $\mathbf{u}_1 = \mathbf{1}_n$  as the only eigenvector with eigenvalue 0

# More Understanding on Graph Laplacian

- Number of connected components and the spectrum of  $L=D-W$
- First assume there are  $k > 1$  connected component
  - Each connected component has an associated Laplacian. Therefore, we can write matrix  $L$  as a *block diagonal matrix*

$$L = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_k \end{bmatrix}$$

- Each block corresponds to a connected component, hence each matrix  $L_i$  has an eigenvalue 0 with multiplicity 1.
- The spectrum of  $L$  is given by the union of the spectra of  $L_i$
- The eigenvalue  $\lambda_1 = 0$  has multiplicity  $k$

# More Understanding on Graph Laplacian

- Number of connected components and the spectrum of  $L=D-W$
- First assume there are  $k > 1$  connected component
  - The eigenvalue  $\lambda_1 = 0$  has multiplicity  $k$
  - The eigenspace corresponding to  $\lambda_1 = \dots = \lambda_k = 0$  is spanned by the  $k$  mutually orthogonal vectors:

$$\mathbf{u}_1 = \mathbf{1}_{L_1}$$

...

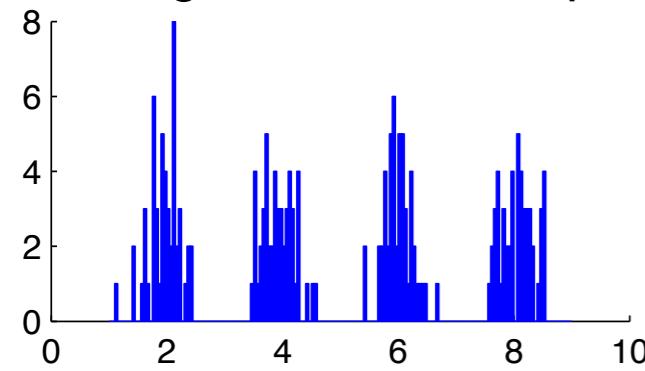
$$\mathbf{u}_k = \mathbf{1}_{L_k}$$

- with  $\mathbf{1}_{L_k} = (0000111110000)^\top \in \mathbb{R}^n$
- These vectors are indicator vectors of graph's connected components
- Note that  $\mathbf{1}_{L_1} + \dots + \mathbf{1}_{L_k} = \mathbf{1}_n$

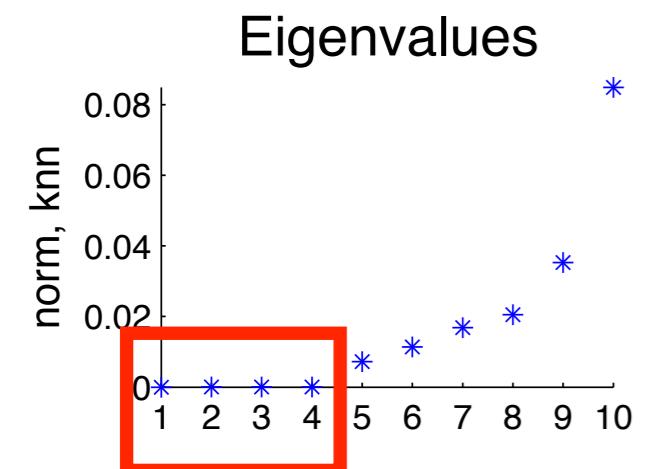
# Number of Connected Components

**Proposition 2 (Number of connected components and the spectrum of  $L$ )** Let  $G$  be an undirected graph with non-negative weights. Then the multiplicity  $k$  of the eigenvalue 0 of  $L$  equals the number of connected components  $A_1, \dots, A_k$  in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors  $\mathbb{1}_{A_1}, \dots, \mathbb{1}_{A_k}$  of those components.

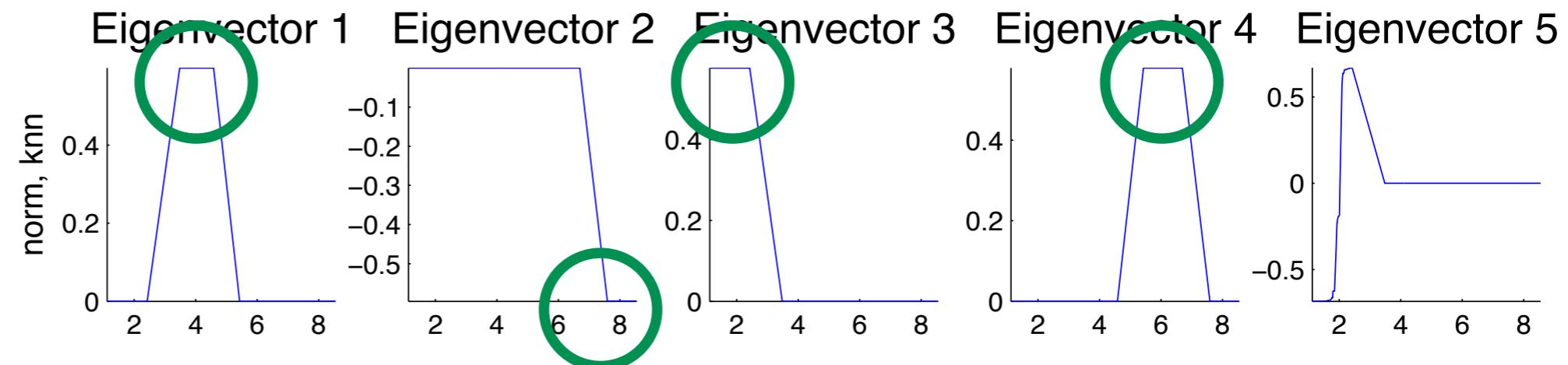
Histogram of the sample



**mixture of Gaussian with 4 clusters  
(now the graph is built by k-nn)**



**4 eigenvalues = 0  
which means there are  
4 connected components  
in the graph**



**indicator vectors on the partitions  
(they are eigenvectors of  $L$  with eigenvalue 0)**

# More Understanding on Graph Laplacian

- The Fiedler vector of the Graph Laplacian
  - The first non-null eigenvalue  $\lambda_{k+1}$  is called Fidler value
  - The corresponding eigenvector  $\mathbf{u}_{k+1}$  is called the Fiedler vector
  - Now again, for a fully-connected graph,  $\mathbf{u}_1 = \mathbf{1}_n$  and  $L\mathbf{1}_n = \mathbf{0}$
  - $\mathbf{u}_2$  is the the Fiedler vector with multiplicity 1
  - The eigenvectors form an orthonormal basis:  $\mathbf{u}_i^\top \mathbf{u}_j = \delta_{ij}$
  - For any eigenvector  $\mathbf{u}_i = [\mathbf{u}_i(v_1) \dots \mathbf{u}_i(v_n)]^\top$ ,  $2 \leq i \leq n$   
(orthogonal between eigenvectors, where the 1st eigenvector is  $\mathbf{1}_n$ )

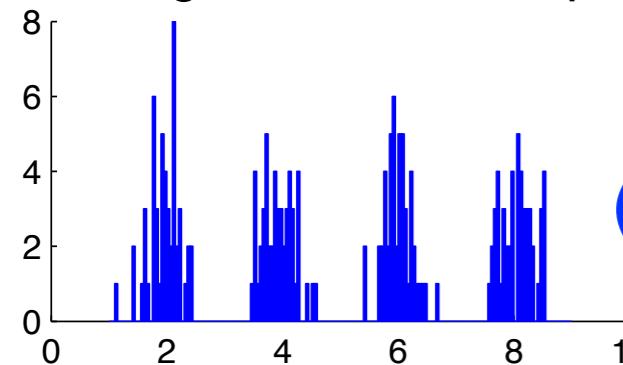
$$\mathbf{u}_i^\top \mathbf{1}_n = 0$$

- Hence the components of  $\mathbf{u}_i$ ,  $2 \leq i \leq n$  satisfy:

$$\sum_{j=1}^n \mathbf{u}_i(v_j) = 0, \text{ where } -1 < \mathbf{u}_i(v_j) < 1 \text{ due to orthonormality}$$

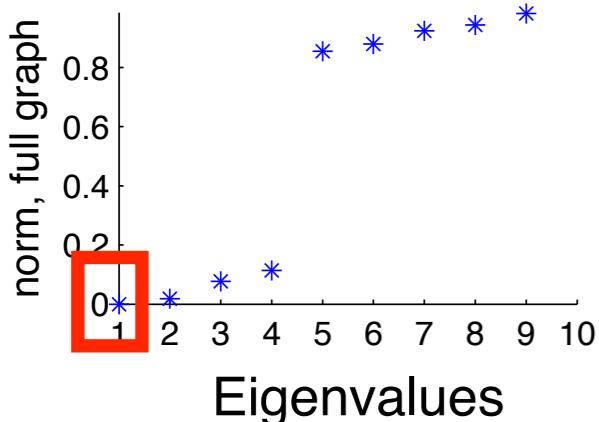
# How about Fully Connected Graph?

Histogram of the sample

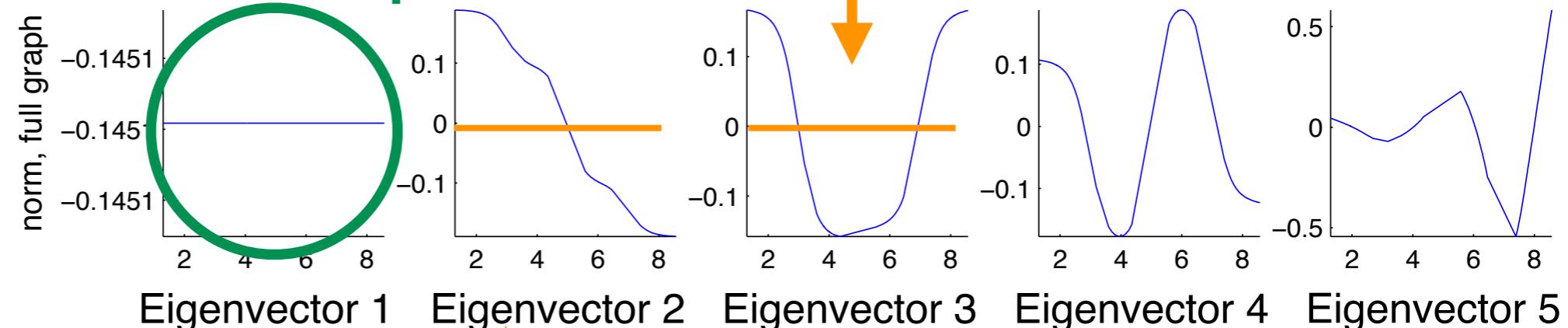


**mixture of Gaussian with 4 clusters  
(now the graph is built by RBF kernel)**

**indicates now all  
nodes in a component**



**now fully connected graph  
only 1 eigenvalue = 0**



**thresholding on 0 can separate  
cluster {#1, #2} from {#3, #4}  
still like indicator vectors!**

# More Understanding on Graph Laplacian

- Now again, assume a fully connected Graph (e.g. built by RBF)
  - sometimes we call this graph as **weighted graph**, since every edge is weighted by the similarities between vertices
  - Map a weighted graph onto a line (will come back to this later)**  
such that connected vertices stay as close as possible, i.e., minimize

$$\sum_{i,j=1}^n w_{ij}(f(v_i) - f(v_j))^2, \text{ or}$$

$$\operatorname{argmin}_{\mathbf{f}} \mathbf{f}^\top L \mathbf{f} \text{ subject to } \underline{\mathbf{f}^\top \mathbf{f} = 1} \text{ and } \underline{\mathbf{f}^\top \mathbf{1} = 0}$$

**orthonormality    w.r.t. 1st eigenvector**

- Lets first introduce something related to this optimization problem but more general: **Rayleigh quotient** (雷利 · 商)

# More Understanding on Graph Laplacian

- **Rayleigh quotient** (雷利 · 商)

$$\text{objective function: } J(\mathbf{f}) = \frac{\mathbf{f}^\top L \mathbf{f}}{\mathbf{f}^\top D \mathbf{f}}$$

where  $L$  and  $D$  are symmetric and positive semidefinite

- please note here  $L$  and  $D$  are not limited to graph Laplacian or Degree
- if you scale  $\mathbf{f}$  then you get the same value of  $J(\mathbf{f})$ , let's constraint it!

optimizing  $\mathbf{f}^\top L \mathbf{f}$  subject to  $\mathbf{f}^\top D \mathbf{f} = C$

- reminds you anything? Lagrange Multiplier!

$$J(\mathbf{f}) = \mathbf{f}^\top L \mathbf{f} - \lambda(\mathbf{f}^\top D \mathbf{f} - C)$$
$$\frac{\partial J}{\partial \mathbf{f}} = 2(L - \lambda D)\mathbf{f} = 0 \Rightarrow L\mathbf{f} = \lambda D\mathbf{f}$$

**generalized eigenvalue problem**

# More Understanding on Graph Laplacian

- **Rayleigh quotient** (雷利 · 商)

- We want to

optimizing  $\mathbf{f}^\top \mathbf{L} \mathbf{f}$  subject to  $\mathbf{f}^\top \mathbf{D} \mathbf{f} = C$

where the optimal  $\mathbf{f}^*$  satisfies  $\mathbf{L}\mathbf{f} = \lambda\mathbf{D}\mathbf{f}$

- Hence we get  $(\mathbf{f}^*)^\top \mathbf{L} \mathbf{f}^* = \lambda(\mathbf{f}^*)^\top \mathbf{D} (\mathbf{f}^*)^\top = \lambda C$
  - since  $C$  is a constant, the maximum of  $(\mathbf{f}^*)^\top \mathbf{L} \mathbf{f}^*$  can be obtained by the largest eigenvalue of  $\mathbf{L}$  and  $\mathbf{D}$  (generalized eigenvalue problem); and the minimum obtained by the smallest eigenvalue

# More Understanding on Graph Laplacian

- Map a weighted graph onto a line (will come back to this later)  
such that connected vertices stay as close as possible, i.e., minimize

$$\sum_{i,j=1}^n w_{ij}(f(v_i) - f(v_j))^2, \text{ or}$$

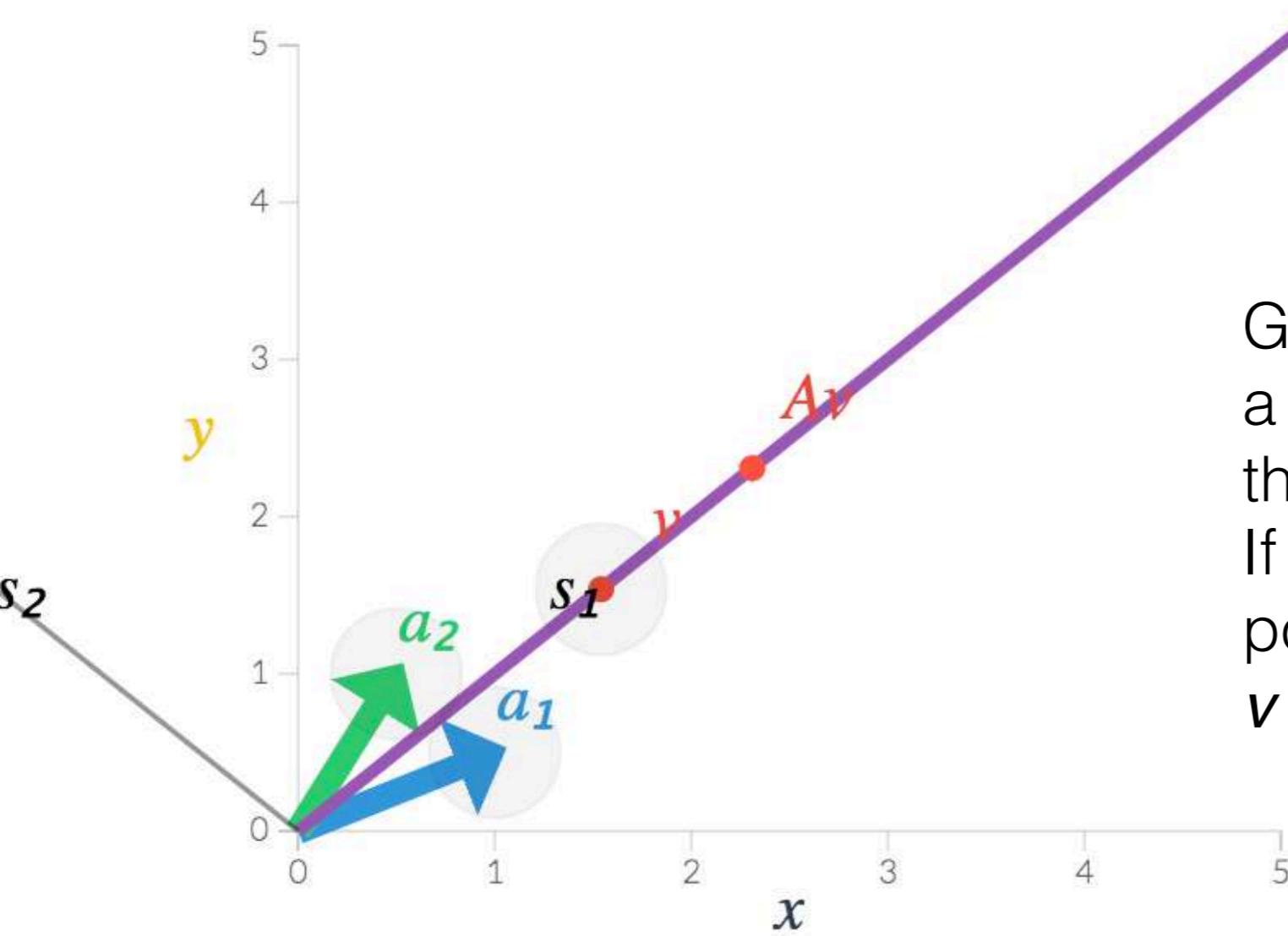
$$\operatorname{argmin}_{\mathbf{f}} \mathbf{f}^\top L \mathbf{f} \text{ subject to } \underline{\mathbf{f}^\top \mathbf{f} = 1} \text{ and } \underline{\mathbf{f}^\top \mathbf{1} = 0}$$

**orthonormality    w.r.t. 1st eigenvector**

- Analogous to simpler Rayleigh quotient formulation  
and eigenvalue problem  $L\mathbf{f} = \lambda\mathbf{f}$
- Minimum obtained by **2nd smallest eigenvalue** since we have  $\mathbf{f}^\top \mathbf{1} = 0$

# More Understanding on Graph Laplacian

- Map a weighted graph onto a line (will come back to this later)

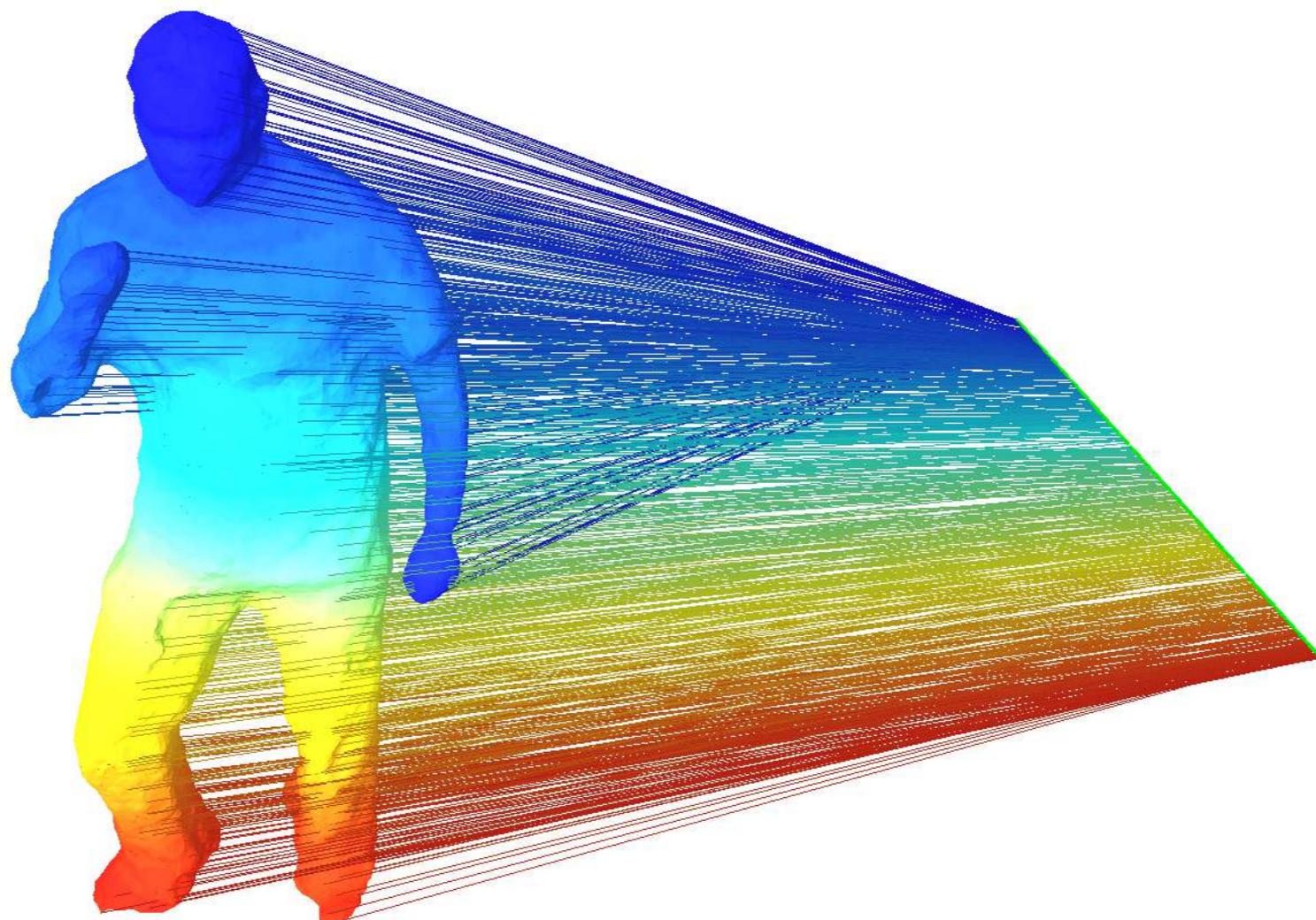


$$A = \begin{bmatrix} a_{1,x} & a_{2,x} \\ a_{1,y} & a_{2,y} \end{bmatrix}$$

Given an matrix  $A$ , if we multiply a vector  $v$  (shown as a point) by  $A$ , then  $A$  sends  $v$  to a new vector  $Av$ . If we can draw a **line** through three points:  $(0,0)$ ,  $v$ , and  $Av$ , then  $Av$  is just  $v$  multiplied by a number  $\lambda$ , so  $Av=\lambda v$

# More Understanding on Graph Laplacian

- ▶ Map a weighted graph onto a line:  
*Fiedler vector* related to first non-null eigenvalue (2nd smallest)



# More Understanding on Graph Laplacian

- Now not only map a weighted graph onto a line, but embed it into a  **$k$ -dimensional Euclidean space**. The embedding is given by the  $n \times k$  matrix  $\mathbf{F} = [f_1 \ f_2 \ \dots \ f_k]$  where the  $i$ -th row of this matrix corresponds to the Euclidean coordinates of the  $i$ -th node  $v_i$ 
  - We need to minimize (Belkin & Niyogi '03):

$$\arg \min_{\mathbf{f}_1 \dots \mathbf{f}_k} \sum_{i,j=1}^n w_{ij} \left\| \mathbf{f}^{(i)} - \mathbf{f}^{(j)} \right\|^2 \text{ with } \mathbf{F}^\top \mathbf{F} = \mathbf{I}$$

- The solution is provided by the matrix of eigenvectors corresponding to the  $k$  lowest nonzero eigenvalues of the eigenvalue problem  $L\mathbf{f} = \lambda\mathbf{f}$

# More Understanding on Graph Laplacian

- Now not only map a weighted graph onto a line, but embed it into a  **$k$ -dimensional Euclidean space**

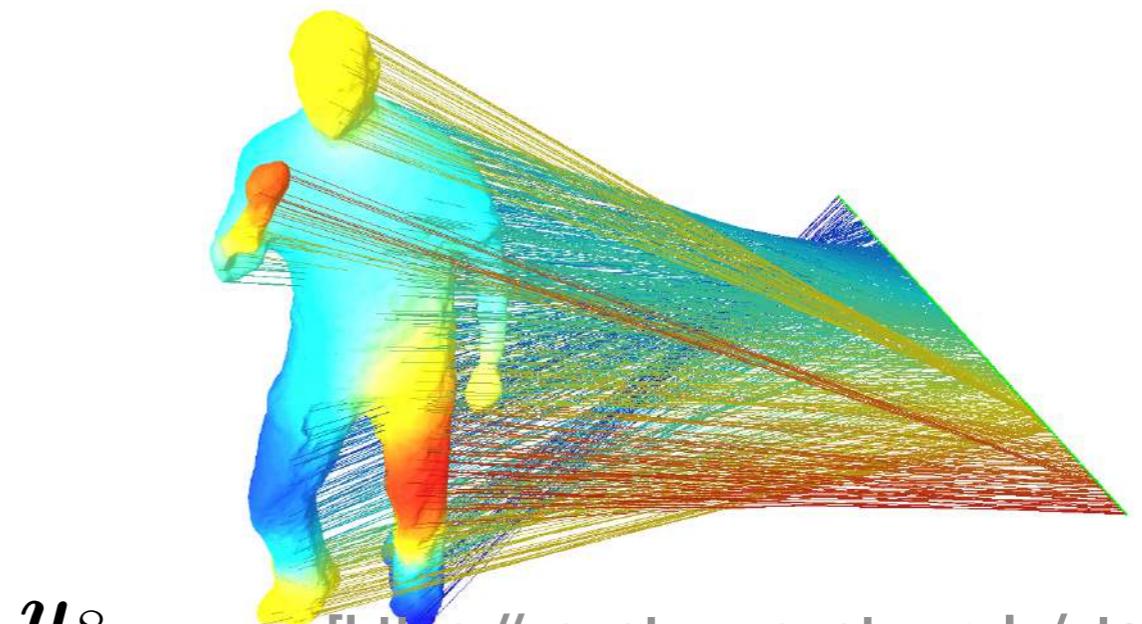
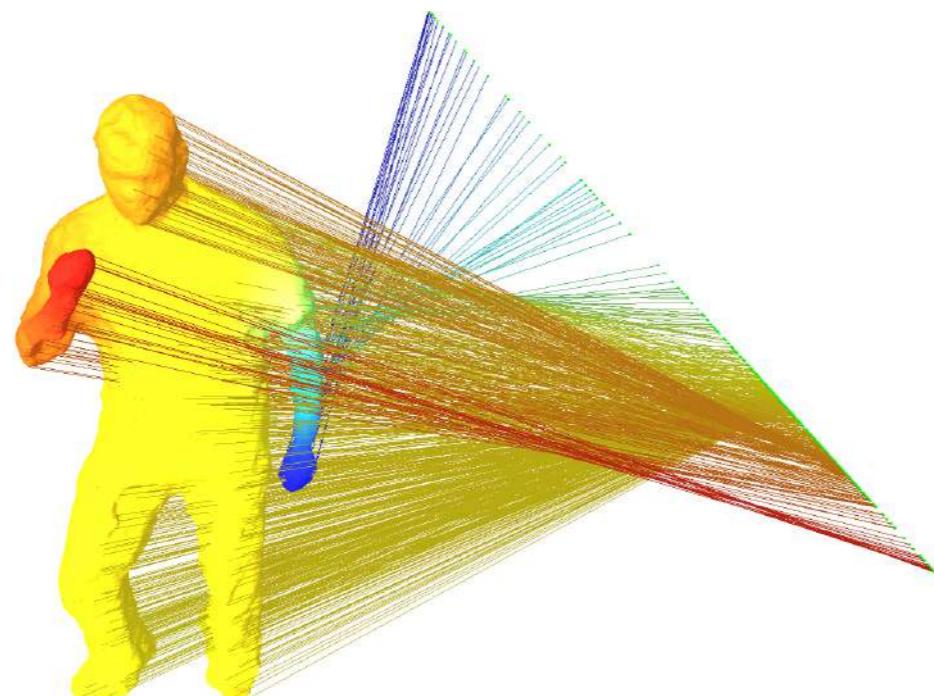
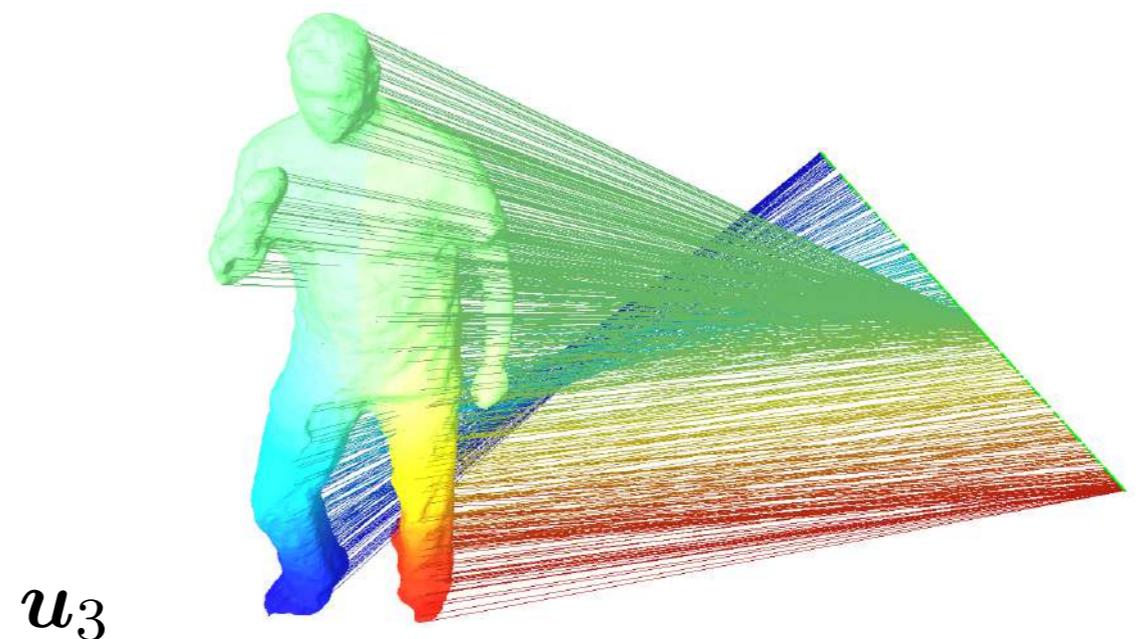
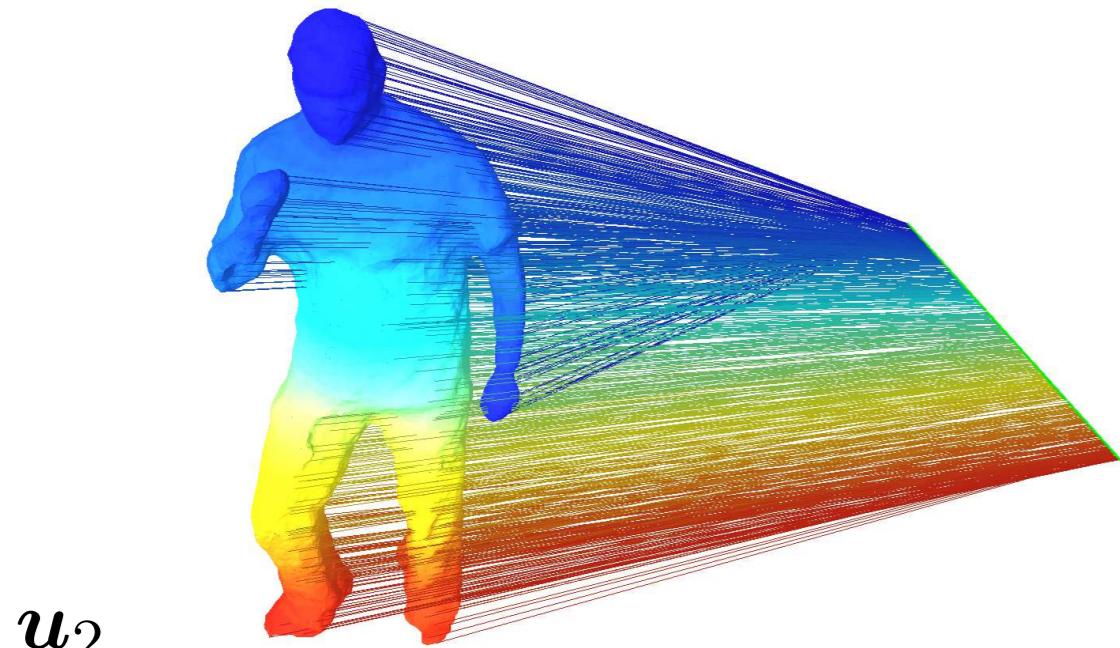
- Compute the eigendecomposition  $L = D - W$
- Select the  $k$  smallest non-null eigenvalues  $\lambda_2 \leq \dots \leq \lambda_{k+1}$
- $\lambda_{k+2} - \lambda_{k+1}$  = **eigengap**
- We obtain the  $n \times k$  matrix  $\mathbf{U} = [\mathbf{u}_2 \dots \mathbf{u}_{k+1}]$

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_2(v_1) & \cdots & \mathbf{u}_{k+1}(v_1) \\ \vdots & & \vdots \\ \mathbf{u}_2(v_n) & \cdots & \mathbf{u}_{k+1}(v_n) \end{bmatrix}$$

- $\mathbf{u}_i^\top \mathbf{u}_j = \delta_{ij}$  (orthonormal vectors), hence  $\mathbf{U}^\top \mathbf{U} = \mathbf{I}_k$
- Column  $i$  ( $2 \leq i \leq k+1$ ) of  $\mathbf{U}$  is a mapping on the eigenvector  $u_i$**
- These eigenvectors are related to indicator vectors**

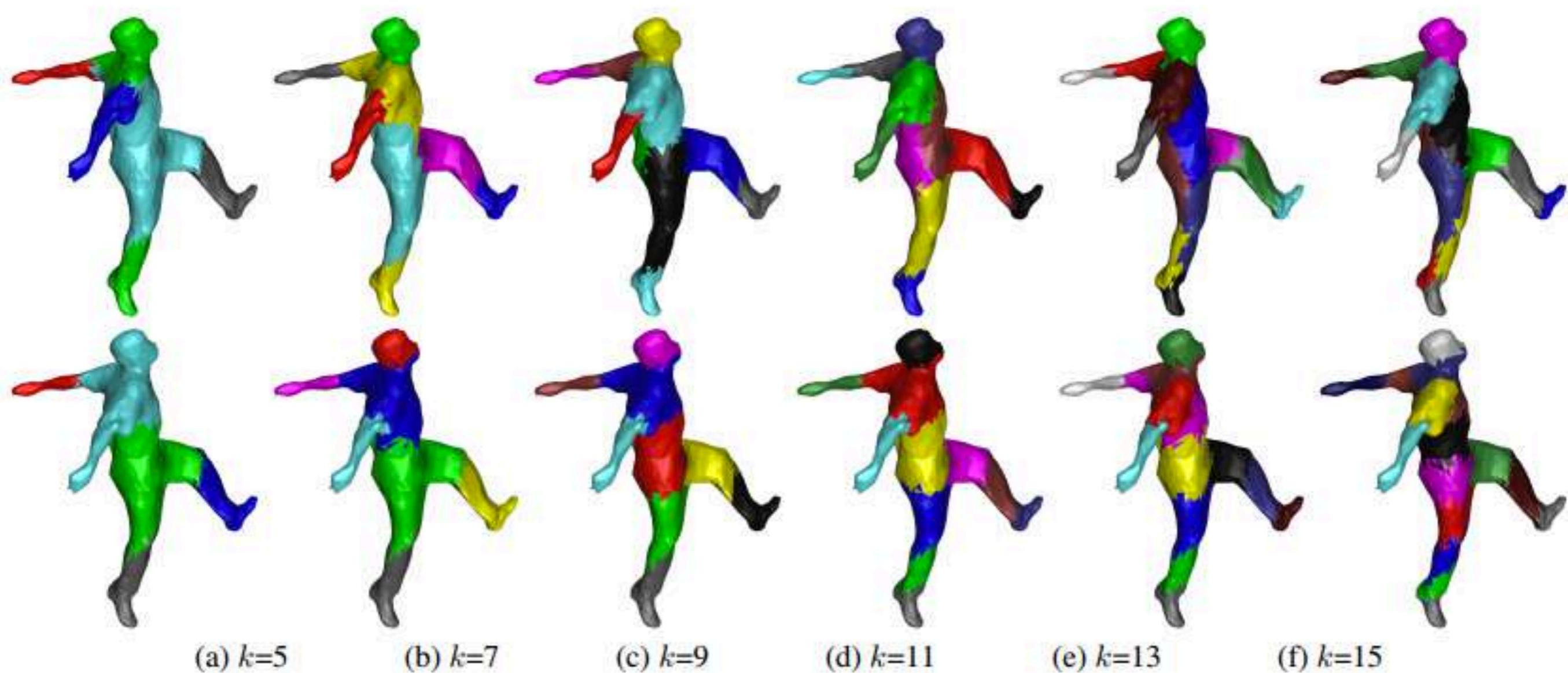
# More Understanding on Graph Laplacian

- Now not only map a weighted graph onto a line, but embed it into a  **$k$ -dimensional Euclidean space**



# More Understanding on Graph Laplacian

- Now not only map a weighted graph onto a line, but embed it into a  **$k$ -dimensional Euclidean space**



# Spectral Clustering

## Unnormalized spectral clustering

Input: Similarity matrix  $S \in \mathbb{R}^{n \times n}$ , number  $k$  of clusters to construct.

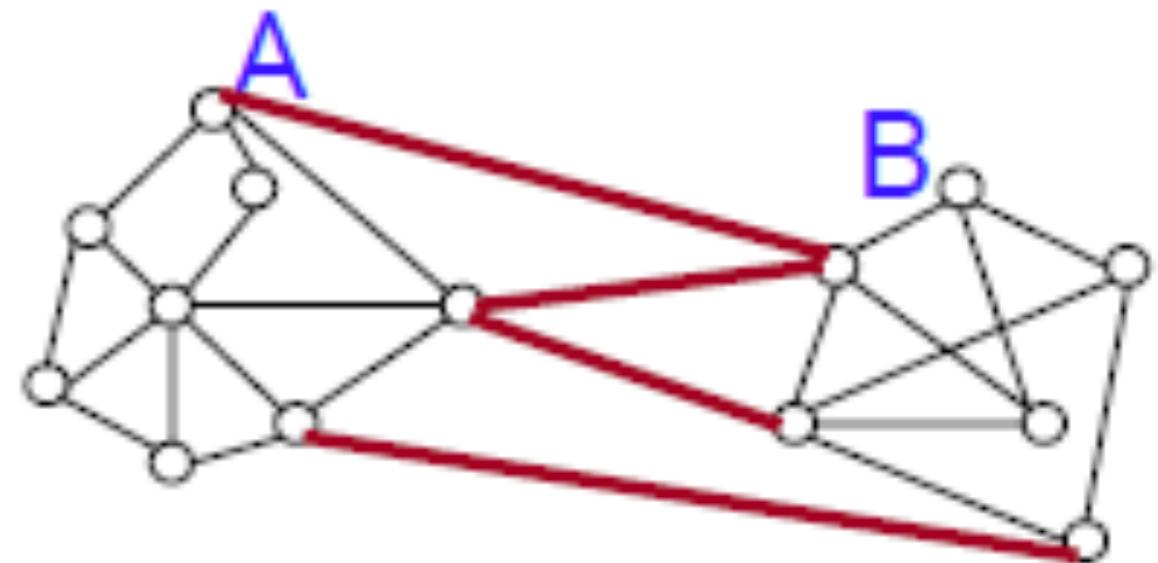
- Construct a similarity graph by one of the ways described in Section 2. Let  $W$  be its weighted adjacency matrix.
- Compute the unnormalized Laplacian  $L$ .
- **Compute the first  $k$  eigenvectors  $u_1, \dots, u_k$  of  $L$ .**
- Let  $U \in \mathbb{R}^{n \times k}$  be the matrix containing the vectors  $u_1, \dots, u_k$  as columns.
- For  $i = 1, \dots, n$ , let  $y_i \in \mathbb{R}^k$  be the vector corresponding to the  $i$ -th row of  $U$ .
- Cluster the points  $(y_i)_{i=1,\dots,n}$  in  $\mathbb{R}^k$  with the  $k$ -means algorithm into clusters  $C_1, \dots, C_k$ .

Output: Clusters  $A_1, \dots, A_k$  with  $A_i = \{j \mid y_j \in C_i\}$ .

# Now back to Ratio Cut and Normalized Cut

- Partition graph into two sets A and B such that
  - weights of edges connecting vertices in A to vertices in B is minimum
  - size of A and B are very similar**

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



- Normalized Cut:**

$$\text{Ncut}(A, B) := \text{cut}(A, B) \left( \frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)} \right) \quad \text{vol}(A) = \sum_{i \in A} d_i$$

- Ratio Cut**

**additional constraints**

$$\text{Rcut}(A, B) := \text{cut}(A, B) \left( \frac{1}{|A|} + \frac{1}{|B|} \right) \quad |A| : \# \text{ of nodes in } A$$

# Graph Laplacian and Ratio Cuts

- Ratio cuts for  $k = 2 \quad \min_{A \subset V} \text{RatioCut}(A, \bar{A}).$

- Define cluster indicator variables

**we wish  $f$  with form  
similar to indicator**

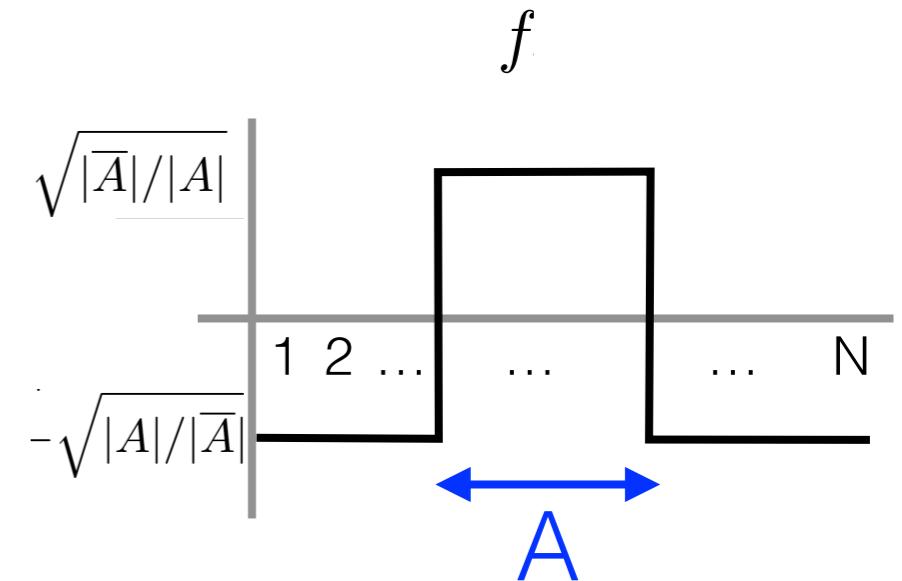
$$f_i = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } v_i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } v_i \in \bar{A}. \end{cases}$$

- Properties

$$\sum_{i=1}^n f_i = \sum_{i \in A} \sqrt{\frac{|\bar{A}|}{|A|}} - \sum_{i \in \bar{A}} \sqrt{\frac{|A|}{|\bar{A}|}} = |A| \sqrt{\frac{|\bar{A}|}{|A|}} - |\bar{A}| \sqrt{\frac{|A|}{|\bar{A}|}} = 0$$

$$\|f\|^2 = \sum_{i=1}^n f_i^2 = |A| \frac{|\bar{A}|}{|A|} + |\bar{A}| \frac{|A|}{|\bar{A}|} = |\bar{A}| + |A| = n.$$

👉  $f \perp \mathbf{1}, \|f\| = \sqrt{n}.$



# Graph Laplacian and Ratio Cuts

- RatioCut

$$f' L f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2$$

$$\begin{aligned} &= \frac{1}{2} \sum_{\substack{i \in A, j \in \bar{A}}} w_{ij} \left( \sqrt{\frac{|\bar{A}|}{|A|}} + \sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 + \frac{1}{2} \sum_{\substack{i \in \bar{A}, j \in A}} w_{ij} \left( -\sqrt{\frac{|\bar{A}|}{|A|}} - \sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 \\ &= \underline{\text{cut}(A, \bar{A})} \left( \frac{|\bar{A}|}{|A|} + \frac{|A|}{|\bar{A}|} + 2 \right) \end{aligned}$$

$$= \boxed{\text{cut}(A, \bar{A})} \left( \frac{|A| + |\bar{A}|}{|A|} + \frac{|A| + |\bar{A}|}{|\bar{A}|} \right)$$

$$= |V| \cdot \boxed{\text{RatioCut}(A, \bar{A})}$$

$$\text{Rcut}(A, B) := \text{cut}(A, B) \left( \frac{1}{|A|} + \frac{1}{|B|} \right)$$

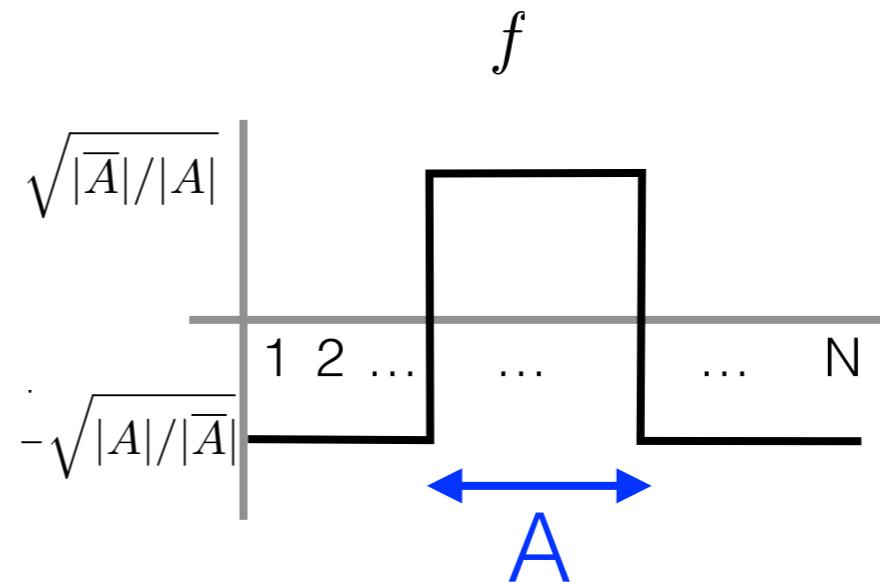


$$\text{RatioCut}(A, \bar{A}) = \frac{f' L f}{|V|}$$

ah-ha!

# Graph Laplacian and Ratio Cuts

- RatioCut



*f* are in “discrete” set

$$\min_{A \subset V} f' L f \text{ subject to } f \perp \mathbf{1}, \quad f_i = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } v_i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } v_i \in \bar{A}. \end{cases}, \quad \|f\| = \sqrt{n}.$$

↑

$$\text{RatioCut}(A, \bar{A}) = \frac{f' L f}{|V|}$$

# Graph Laplacian and Ratio Cuts

- RatioCut

1st eigenvector of  $L$

$$\min_{f \in \mathbb{R}^n} f' L f \text{ subject to } f \perp \mathbf{1}, \|f\| = \sqrt{n}.$$

constant

 **relaxation by allowing  $f$  continuous  
 $f$  are in “discrete” set**

$$\min_{A \subset V} f' L f \text{ subject to } f \perp \mathbf{1}, f_i = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } v_i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } v_i \in \bar{A}. \end{cases}, \|f\| = \sqrt{n}.$$



$$\text{RatioCut}(A, \bar{A}) = \frac{f' L f}{|V|}$$

# Graph Laplacian and Ratio Cuts

1st eigenvector of  $L$

$$\min_{f \in \mathbb{R}^n} f' L f \text{ subject to } f \perp \mathbf{1}, \|f\| = \sqrt{n}.$$

constant

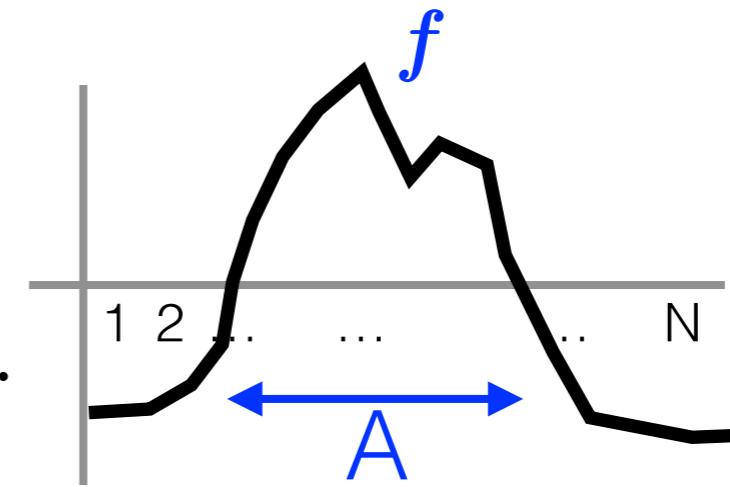
↑ relaxation by allowing  $f$  continuous  
 $f$  are in “discrete” set

- Rayleigh-Ritz Theorem
  - Which vector minimizes objective subject to constraint that the vector is orthogonal to the first eigenvector and has bounded norm?

**$f$  = eigenvector associated with 2nd smallest eigenvalue**

# Graph Laplacian and Ratio Cuts

$$\begin{cases} v_i \in A & \text{if } f_i \geq 0 \\ v_i \in \bar{A} & \text{if } f_i < 0. \end{cases}$$



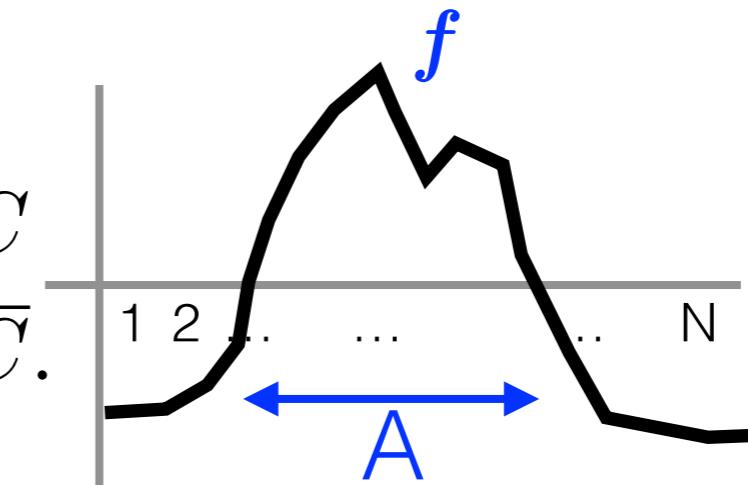
- Rayleigh-Ritz Theorem
  - Which vector maximizes objective subject to constraint that the vector is orthogonal to the first eigenvector and has bounded norm?

**$f$  = eigenvector associated with 2nd smallest eigenvalue**

**but now  $f$  is continuous, need to transform it to a discrete indicator**

# Graph Laplacian and Ratio Cuts

$$\begin{cases} v_i \in A & \text{if } f_i \in C \\ v_i \in \bar{A} & \text{if } f_i \in \bar{C}. \end{cases}$$



- Rayleigh-Ritz Theorem
  - Which vector maximizes objective subject to constraint that the vector is orthogonal to the first eigenvector and has bounded norm?

**$f$  = eigenvector associated with 2nd smallest eigenvalue**

**but now  $f$  is continuous, need to transform it to a discrete indicator**

**Commonly used: cluster coordinates by k-means!**

# Graph Laplacian and Ratio Cuts, $k$ Cases

- Indicator variables

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & \text{if } v_i \in A_j \\ 0 & \text{otherwise} \end{cases} \quad (i = 1, \dots, n; j = 1, \dots, k).$$

$$H'H = I$$

- RatioCut

$$\text{RatioCut}(A_1, \dots, A_k) = \sum_{i=1}^k h'_i L h_i = \sum_{i=1}^k (H'LH)_{ii} = \text{Tr}(H'LH)$$



$$\min_{A_1, \dots, A_k} \text{Tr}(H'LH) \text{ subject to } H'H = I$$



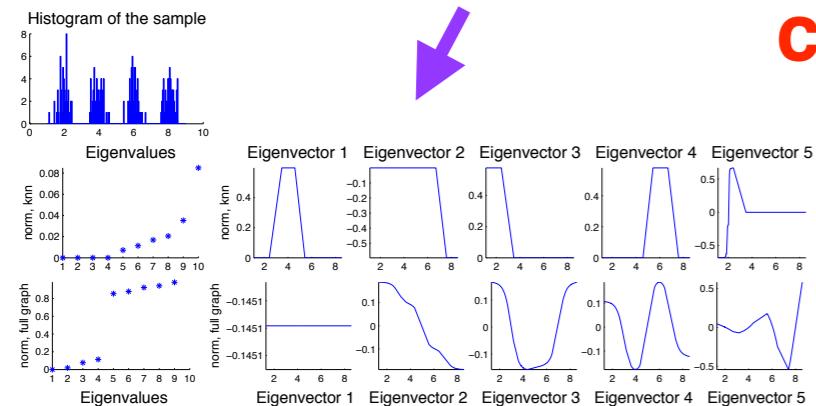
$$\min_{H \in \mathbb{R}^{n \times k}} \text{Tr}(H'LH) \text{ subject to } H'H = I$$

relaxation by allowing  $h$  continuous

# Graph Laplacian and Ratio Cuts, k Cases

if point  $i$  is in a cluster with point  $j$ , then the rows are the same?  
refer to story shown before, # connected components (by kNN or RBF)

cluster coordinates by k-means!



$$H =$$

$$\begin{bmatrix} & & & \\ u_1 & \cdots & u_k & \\ & & & \end{bmatrix}$$

# clusters

# data points

standard trace minimization problem:  
choose  $H$  containing first  $k$  eigenvectors of  $L$

$$\min_{H \in \mathbb{R}^{n \times k}} \text{Tr}(H'LH) \text{ subject to } H'H = I$$

# Graph Laplacian and Normalized Cuts

- Normalized cuts for  $k = 2$
- Define cluster indicator variables

**we wish  $f$  with form  
similar to indicator**

$$f_i = \begin{cases} \sqrt{\frac{\text{vol}(A)}{\text{vol } A}} & \text{if } v_i \in A \\ -\sqrt{\frac{\text{vol}(A)}{\text{vol } (\bar{A})}} & \text{if } v_i \in \bar{A}. \end{cases}$$

- Properties

👉  $(Df)' \mathbf{1} = 0$ ,  $f' Df = \text{vol}(V)$ , and  $f' L f = \text{vol}(V) \text{Ncut}(A, \bar{A})$

$$\min_A f' L f \text{ subject to } f \text{ as } f_i = \begin{cases} \sqrt{\frac{\text{vol}(\bar{A})}{\text{vol } A}} & \text{if } v_i \in A \\ -\sqrt{\frac{\text{vol}(A)}{\text{vol } (\bar{A})}} & \text{if } v_i \in \bar{A}. \end{cases}$$
$$Df \perp \mathbf{1}, f' Df = \text{vol}(V)$$

# Graph Laplacian and Normalized Cuts

Re-substitute  $f = D^{-1/2}g$  to get continuous indicator vectors then kmeans

Rayleigh-Ritz Theorem again:  $g$  = eigenvector w.r.t. 2nd smallest eigenvalue

$$\min_{g \in \mathbb{R}^n} g' \underline{D^{-1/2} L D^{-1/2}} g \text{ subject to } g \perp D^{1/2} \mathbf{1}, \|g\|^2 = \text{vol}(V)$$

normalized Laplacian

↑ substitution  $\underline{g := D^{1/2} f}$  normalized

$$\min_{f \in \mathbb{R}^n} f' L f \text{ subject to } Df \perp \mathbf{1}, f'Df = \text{vol}(V)$$

↑ relaxation by allowing  $f$  continuous  
 $f$  are in “discrete” set

$$\min_A f' L f \text{ subject to } f \text{ as } f_i = \begin{cases} \sqrt{\frac{\text{vol}(\bar{A})}{\text{vol } A}} & \text{if } v_i \in A \\ -\sqrt{\frac{\text{vol}(A)}{\text{vol}(\bar{A})}} & \text{if } v_i \in \bar{A}. \end{cases}$$
$$Df \perp \mathbf{1}, f'Df = \text{vol}(V)$$

# Graph Laplacian and Normalized Cuts, $k$ Cases

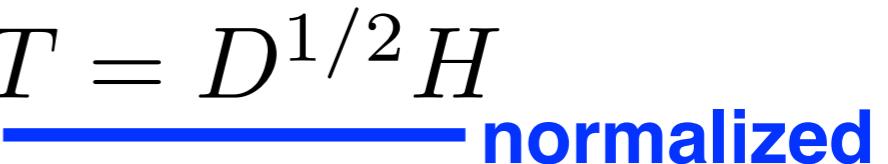
- Indicator variables

$$h_{i,j} = \begin{cases} 1/\sqrt{\text{vol}(A_j)} & \text{if } v_i \in A_j \\ 0 & \text{otherwise} \end{cases} \quad (i = 1, \dots, n; j = 1, \dots, k).$$

$$H'DH = I$$

- Normalized Cut

$$\min_{A_1, \dots, A_k} \text{Tr}(H'LH) \text{ subject to } H'DH = I$$

 **relaxation by allowing  $h$  continuous substitution**  $T = D^{1/2}H$   **normalized**

$$\min_{T \in \mathbb{R}^{n \times k}} \text{Tr}(T'D^{-1/2}LD^{-1/2}T) \text{ subject to } T'T = I.$$

 **normalized Laplacian**

# Graph Laplacian and Normalized Cuts, k Cases

cluster coordinates by k-means!

$$H = \begin{bmatrix} | & & | \\ u_1 & \cdots & u_k \\ | & & | \end{bmatrix}$$

# clusters

# data points

choose  $T$  containing first  $k$  eigenvectors of normalized Laplacian

Re-substitute  $H = D^{-1/2} T$  to get  $H$

$$\min_{T \in \mathbb{R}^{n \times k}} \text{Tr}(T' D^{-1/2} L D^{-1/2} T) \text{ subject to } T'T = I.$$

# Different Laplacian and Relations to Different Cut

- Unnormalized Laplacian  $L = D - W$  serve in the approximation of the minimization of RatioCut
- Normalized Laplacian  $D^{-1/2} L D^{-1/2}$  serve in the approximation of the minimization of NormalizedCut.



ah-ha!

$$D^{-\frac{1}{2}} L D^{-\frac{1}{2}} u = \lambda u$$

**eigenproblem of normalized Laplacian**

$$D^{-\frac{1}{2}} L u' = \lambda D^{\frac{1}{2}} u'$$

**a little trick**  $u = D^{\frac{1}{2}} u'$

$$D^{\frac{1}{2}} D^{-\frac{1}{2}} L u' = \lambda D^{\frac{1}{2}} D^{\frac{1}{2}} u'$$

$$L u' = \lambda D u'$$

**generalized eigenvalue problem!**

$$D^{-1} L u' = \lambda u'$$

$D^{-1} L$  **another normalized Laplacian**  
**(closely related to random-walk)**

# Normalized Laplacian

$$L_{\text{sym}} := D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$

$$L_{\text{rw}} := D^{-1} L = I - D^{-1} W.$$

**Proposition 3 (Properties of  $L_{\text{sym}}$  and  $L_{\text{rw}}$ )** *The normalized Laplacians satisfy the following properties:*

1. For every  $f \in \mathbb{R}^n$  we have

$$f' L_{\text{sym}} f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} \left( \frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2.$$

2.  $\lambda$  is an eigenvalue of  $L_{\text{rw}}$  with eigenvector  $u$  if and only if  $\lambda$  is an eigenvalue of  $L_{\text{sym}}$  with eigenvector  $w = D^{1/2}u$ .
3.  $\lambda$  is an eigenvalue of  $L_{\text{rw}}$  with eigenvector  $u$  if and only if  $\lambda$  and  $u$  solve the generalized eigenproblem  $Lu = \lambda Du$ .
4. 0 is an eigenvalue of  $L_{\text{rw}}$  with the constant one vector  $\mathbf{1}$  as eigenvector. 0 is an eigenvalue of  $L_{\text{sym}}$  with eigenvector  $D^{1/2}\mathbf{1}$ .
5.  $L_{\text{sym}}$  and  $L_{\text{rw}}$  are positive semi-definite and have  $n$  non-negative real-valued eigenvalues  $0 = \lambda_1 \leq \dots \leq \lambda_n$ .

# Normalized Spectral Clustering

Normalized spectral clustering according to Ng, Jordan, and Weiss (2002)

Input: Similarity matrix  $S \in \mathbb{R}^{n \times n}$ , number  $k$  of clusters to construct.

- Construct a similarity graph by one of the ways described in Section 2. Let  $W$  be its weighted adjacency matrix.
- Compute the normalized Laplacian  $L_{\text{sym}} = D^{-1/2} L D^{-1/2}$
- Compute the first  $k$  eigenvectors  $u_1, \dots, u_k$  of  $L_{\text{sym}}$ .
- Let  $U \in \mathbb{R}^{n \times k}$  be the matrix containing the vectors  $u_1, \dots, u_k$  as columns.
- Form the matrix  $T \in \mathbb{R}^{n \times k}$  from  $U$  by normalizing the rows to norm 1, that is set  $t_{ij} = u_{ij}/(\sum_k u_{ik}^2)^{1/2}$ .
- For  $i = 1, \dots, n$ , let  $y_i \in \mathbb{R}^k$  be the vector corresponding to the  $i$ -th row of  $T$ .
- Cluster the points  $(y_i)_{i=1,\dots,n}$  with the  $k$ -means algorithm into clusters  $C_1, \dots, C_k$ .

Output: Clusters  $A_1, \dots, A_k$  with  $A_i = \{j \mid y_j \in C_i\}$ .

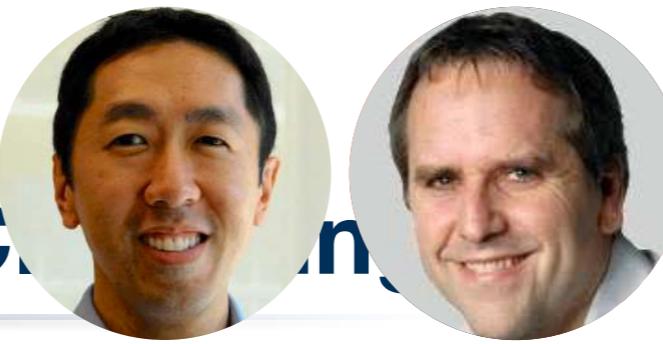


- Compute the unnormalized Laplacian  $L$ .
- Compute the first  $k$  generalized eigenvectors  $u_1, \dots, u_k$  of the generalized eigenproblem  $Lu = \lambda Du$
- Let  $U \in \mathbb{R}^{n \times k}$  be the matrix containing the vectors  $u_1, \dots, u_k$  as columns.
- For  $i = 1, \dots, n$ , let  $y_i \in \mathbb{R}^k$  be the vector corresponding to the  $i$ -th row of  $U$ .
- Cluster the points  $(y_i)_{i=1,\dots,n}$  in  $\mathbb{R}^k$  with the  $k$ -means algorithm into clusters  $C_1, \dots, C_k$ .

Output: Clusters  $A_1, \dots, A_k$  with  $A_i = \{j \mid y_j \in C_i\}$ .

# Normalized Spectral Clustering

coursera



Normalized spectral clustering according to **Ng** **Jordan** and Weiss (2002)

Input: Similarity matrix  $S \in \mathbb{R}^{n \times n}$ , number  $k$  of clusters to construct.

- Construct a similarity graph by one of the ways described in Section 2. Let  $W$  be its weighted adjacency matrix.
- Compute the normalized Laplacian  $L_{\text{sym}} = D^{-1/2} L D^{-1/2}$
- Compute the first  $k$  eigenvectors  $u_1, \dots, u_k$  of  $L_{\text{sym}}$ .
- Let  $U \in \mathbb{R}^{n \times k}$  be the matrix containing the vectors  $u_1, \dots, u_k$  as columns.
- Form the matrix  $T \in \mathbb{R}^{n \times k}$  from  $U$  by normalizing the rows to norm 1, that is set  $t_{ij} = u_{ij}/(\sum_k u_{ik}^2)^{1/2}$ .
- For  $i = 1, \dots, n$ , let  $y_i \in \mathbb{R}^k$  be the vector corresponding to the  $i$ -th row of  $T$ .
- Cluster the points  $(y_i)_{i=1,\dots,n}$  with the  $k$ -means algorithm into clusters  $C_1, \dots, C_k$ .

Output: Clusters  $A_1, \dots, A_k$  with  $A_i = \{j \mid y_j \in C_i\}$ .



- Compute the unnormalized Laplacian  $L$ .
  - Compute the first  $k$  generalized eigenvectors  $u_1, \dots, u_k$  of the generalized eigenproblem  $Lu = \lambda Du$
  - Let  $U \in \mathbb{R}^{n \times k}$  be the matrix containing the vectors  $u_1, \dots, u_k$  as columns.
  - For  $i = 1, \dots, n$ , let  $y_i \in \mathbb{R}^k$  be the vector corresponding to the  $i$ -th row of  $U$ .
  - Cluster the points  $(y_i)_{i=1,\dots,n}$  in  $\mathbb{R}^k$  with the  $k$ -means algorithm into clusters  $C_1, \dots, C_k$ .
- Output: Clusters  $A_1, \dots, A_k$  with  $A_i = \{j \mid y_j \in C_i\}$ .

# We Have Learnt Spectral Clustering, Go Back to “kernel kmeans” many slides ago...

- Make ***kernel kmeans*** great again!
  - Now with **weight** on each data point
  - known as ***weighted kernel kmeans***



$\pi$  defines a clustering

$$\text{distortion measure } \mathcal{D}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^k \sum_{\mathbf{a} \in \pi_j} w(\mathbf{a}) \|\phi(\mathbf{a}) - \mathbf{m}_j\|^2$$

**$k$  clusters**

$$\text{cluster mean } \mathbf{m}_j = \frac{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}) \phi(\mathbf{b})}{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b})}$$

**$w(\mathbf{b})$  weight for data point**

$$\text{feature matrix for cluster } \pi_j = \Phi_j \frac{W_j \mathbf{e}}{s_j}$$

**$\Phi_j$  feature matrix for cluster  $\pi_j$**

**$W_j$  weight matrix for cluster  $\pi_j$**

**$\mathbf{e}$  1-vector with proper size**

$$s_j = \sum_{\mathbf{a} \in \pi_j} w(\mathbf{a})$$

**distance from a to cluster  $j$**

$$\phi(\mathbf{a}) \cdot \phi(\mathbf{a}) - \frac{2 \sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}) \phi(\mathbf{a}) \cdot \phi(\mathbf{b})}{\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b})} + \frac{\sum_{\mathbf{b}, \mathbf{d} \in \pi_j} w(\mathbf{b}) w(\mathbf{d}) \phi(\mathbf{b}) \cdot \phi(\mathbf{d})}{(\sum_{\mathbf{b} \in \pi_j} w(\mathbf{b}))^2}$$

# We Have Learnt Spectral Clustering, Go Back to “kernel kmeans” many slides ago...

- Make ***kernel kmeans*** great again!
  - Now with **weight** on each data point
  - known as ***weighted kernel kmeans***

**distortion measure**  $\mathcal{D}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^k \sum_{\mathbf{a} \in \pi_j} w(\mathbf{a}) \|\phi(\mathbf{a}) - \mathbf{m}_j\|^2$

If we represent the full matrix of points as  $\Phi = [\Phi_1, \Phi_2, \dots, \Phi_k]$ , then we have that

$$\mathcal{D}(\{\pi_j\}_{j=1}^k) = \underbrace{\text{trace}(W^{1/2} \Phi^T \Phi W^{1/2})}_{\text{constant}} - \text{trace}(Y^T W^{1/2} \Phi^T \Phi W^{1/2} Y),$$

where

$$Y = \begin{bmatrix} \frac{W_1^{1/2} \mathbf{e}}{\sqrt{s_1}} & & & \\ & \frac{W_2^{1/2} \mathbf{e}}{\sqrt{s_2}} & & \\ & & \ddots & \\ & & & \frac{W_k^{1/2} \mathbf{e}}{\sqrt{s_k}} \end{bmatrix}.$$

**W matrix:**  
**diagonal elements**  
**are weights for**  
**each data point**  
**0 on off-diagonal**

Note that  $Y$  is an  $n \times k$  orthonormal matrix, i.e.,  $Y^T Y = I$ .

**don't mess up with the variables we used in spectral clustering**

# We Have Learnt Spectral Clustering, Go Back to “kernel kmeans” many slides ago.



- Make ***kernel kmeans*** great again!
  - Now with **weight** on each data point
  - known as ***weighted kernel kmeans***

ah-ha!  
reminds you  
anything?

distortion measure  $\mathcal{D}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^k \sum_{l=1}^n (\pi_j \|\cdot\|_Y)^2$

relaxation by allowing  $Y$  continuous  
optimal  $Y$  is obtained by taking  
top  $k$  eigenvectors of  $W^{1/2} K W^{1/2}$

$$\mathcal{D}(\{\pi_j\}_{j=1}^k) = \text{trace}(W^{1/2} Y^T \Phi Y W^{1/2})$$

constant

maximize  $\text{trace}(Y^T W^{1/2} \Phi^T \Phi W^{1/2} Y)$

where

$$Y = \begin{bmatrix} \frac{W_1^{1/2} e}{\sqrt{s_1}} & & & \\ & \frac{W_2^{1/2} e}{\sqrt{s_2}} & & \\ & & \ddots & \\ & & & \frac{W_k^{1/2} e}{\sqrt{s_k}} \end{bmatrix}.$$

***W matrix:***  
**diagonal elements**  
**are weights for**  
**each data point**  
**0 on off-diagonal**

Note that  $Y$  is an  $n \times k$  orthonormal matrix, i.e.,  $Y^T Y = I$ .

don't mess up with the variables we used in spectral clustering

# We Have Learnt Spectral Clustering, Go Back to “kernel kmeans” many slides ago...

- Make ***kernel kmeans*** great again!
  - Now with **weight** on each data point
  - known as ***weighted kernel kmeans***



Similar story holds for RatioCut.

- ☞ Spectral methods are special case of kernel kmeans! OMG!
- ☞ refer to paper “A unified view of kernel kmeans, spectral clustering and graph cuts”

Inderjit S. et al. KDD’04.

relaxation by allowing  $Y$  continuous  
optimal  $Y$  is obtained by taking

top  $k$  eigenvectors of  $W^{1/2} K W^{1/2}$

maximize  $\text{trace}(Y^T W^{1/2} \Phi^T \Phi W^{1/2} Y)$

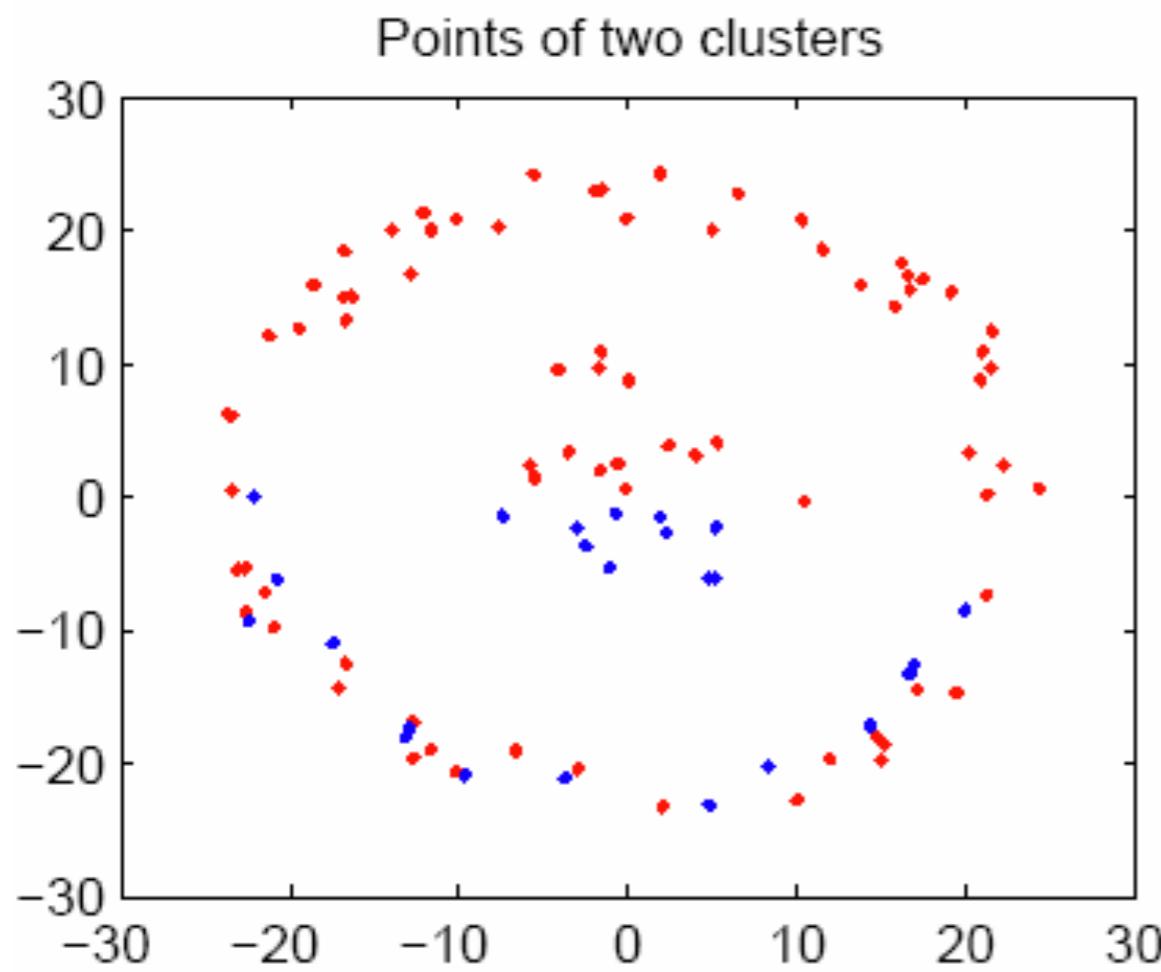
if we assign weights for each data points by its degree used in spectral clustering,  $W=D$ , and replace  $K$  by  $D^{-1} W D^{-1}$ , where  $W$  is similarity matrix in spectral clustering,

☞ maximizing  $\text{trace}(Y D^{-1/2} W D^{-1/2} Y)$

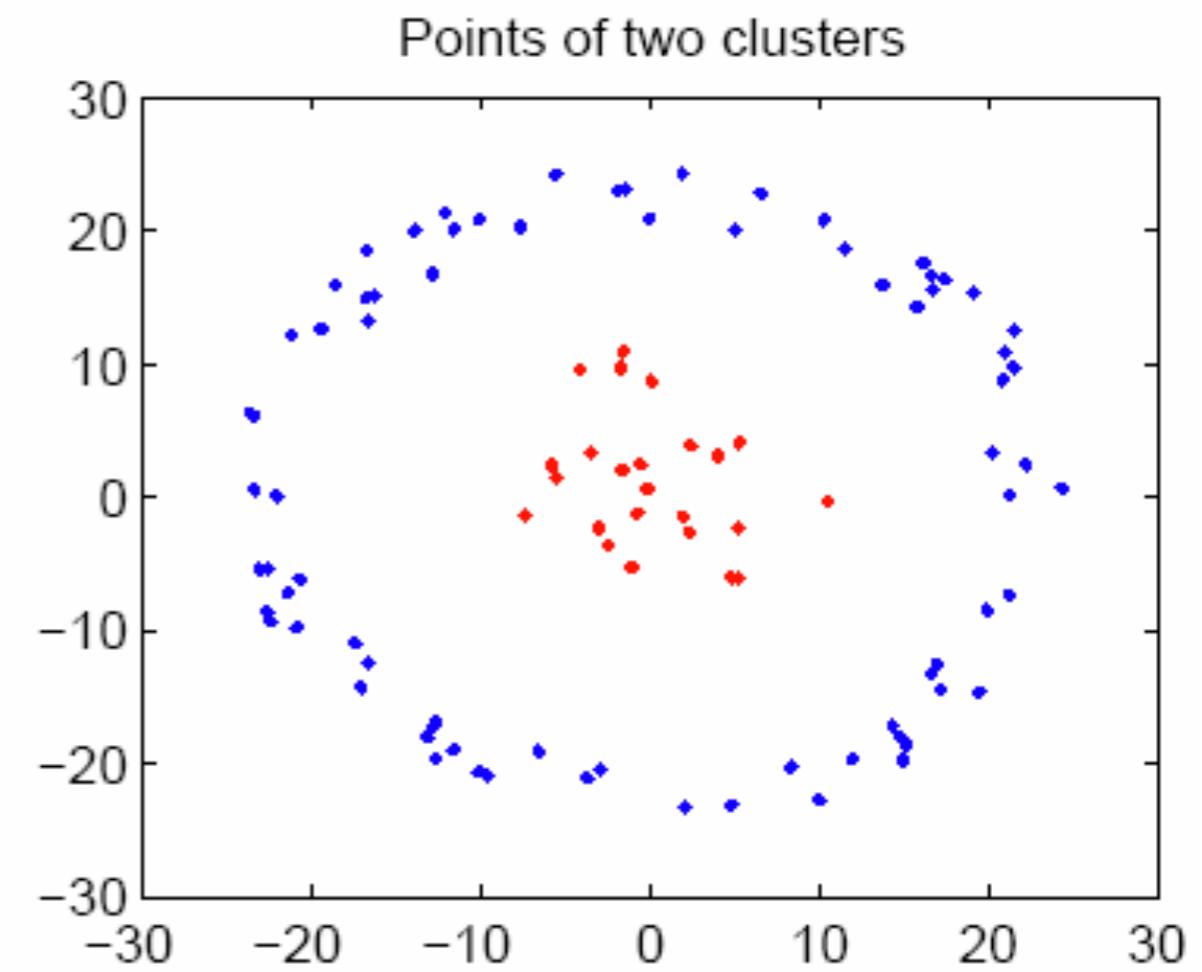
☞ minimizing  $\text{trace}(Y D^{-1/2} L D^{-1/2} Y)$   
Normalized Cut!!!

# kmeans versus Spectral Clustering

- Applying kmeans to Laplacian eigenvectors allows us to **find cluster with non-convex boundaries, as in kernel kmeans**



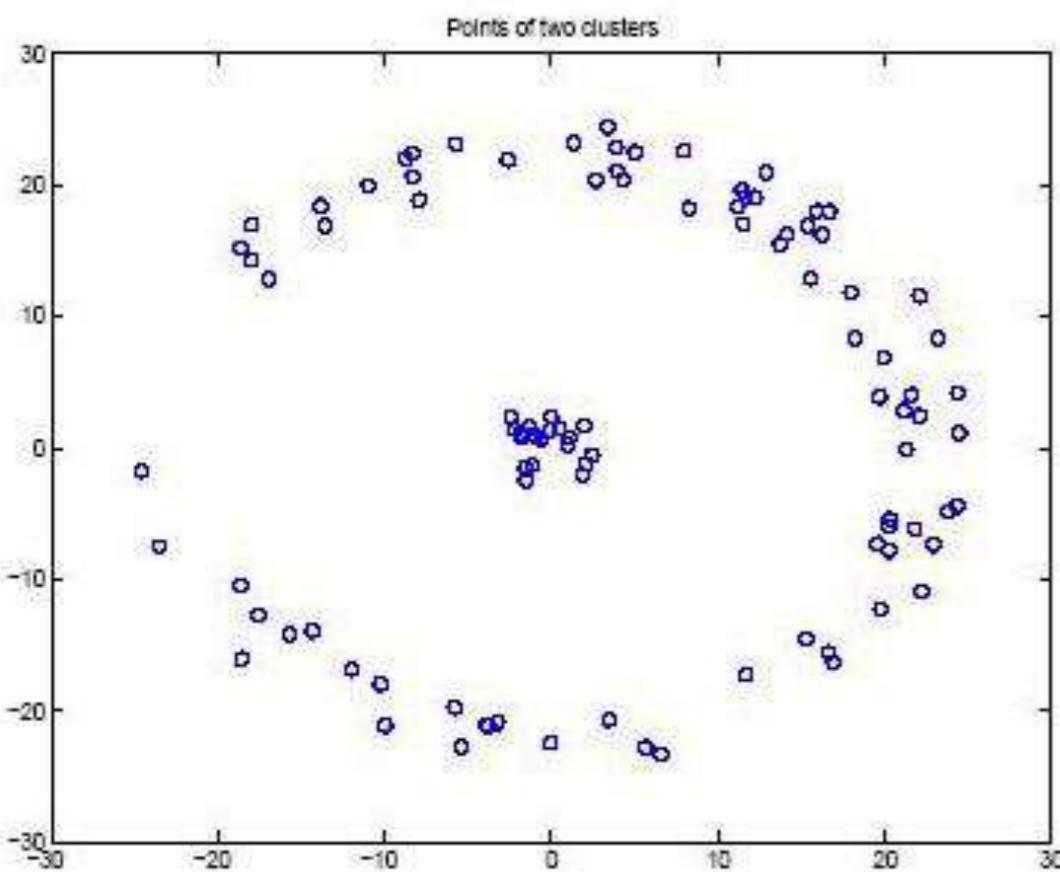
k-means output



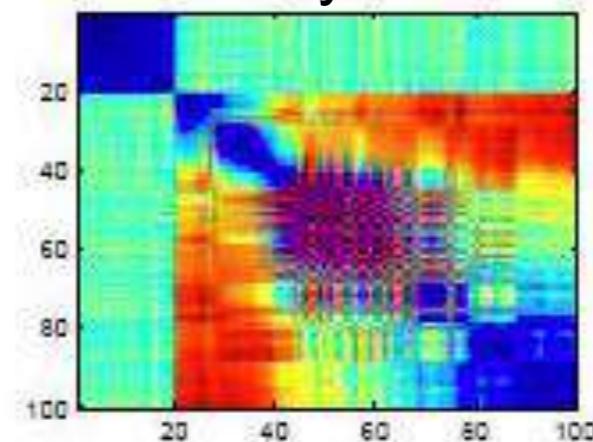
Spectral clustering output

# kmeans versus Spectral Clustering

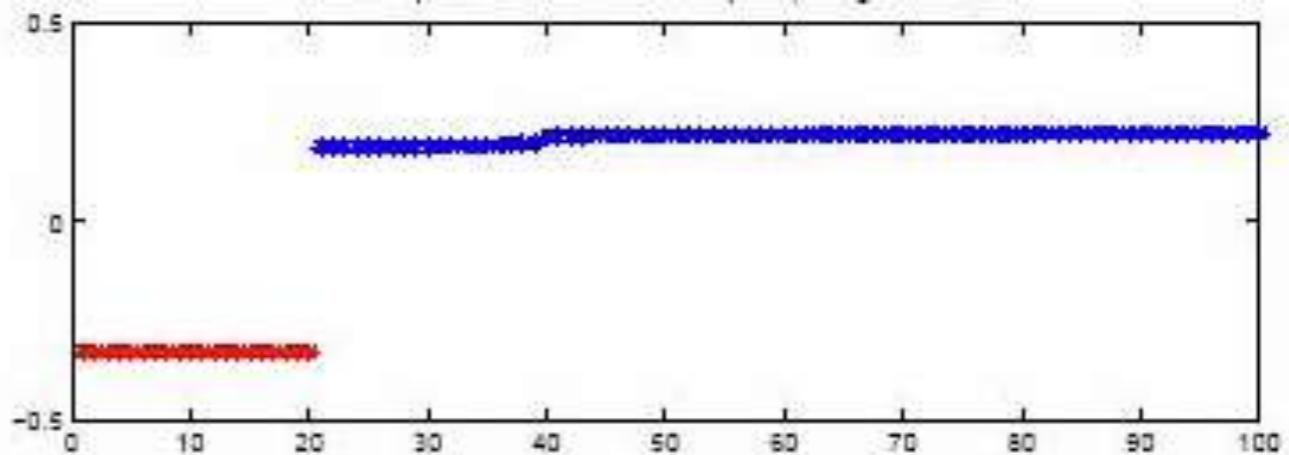
- Applying kmeans to Laplacian eigenvectors allows us to **find cluster with non-convex boundaries**, as in kernel kmeans



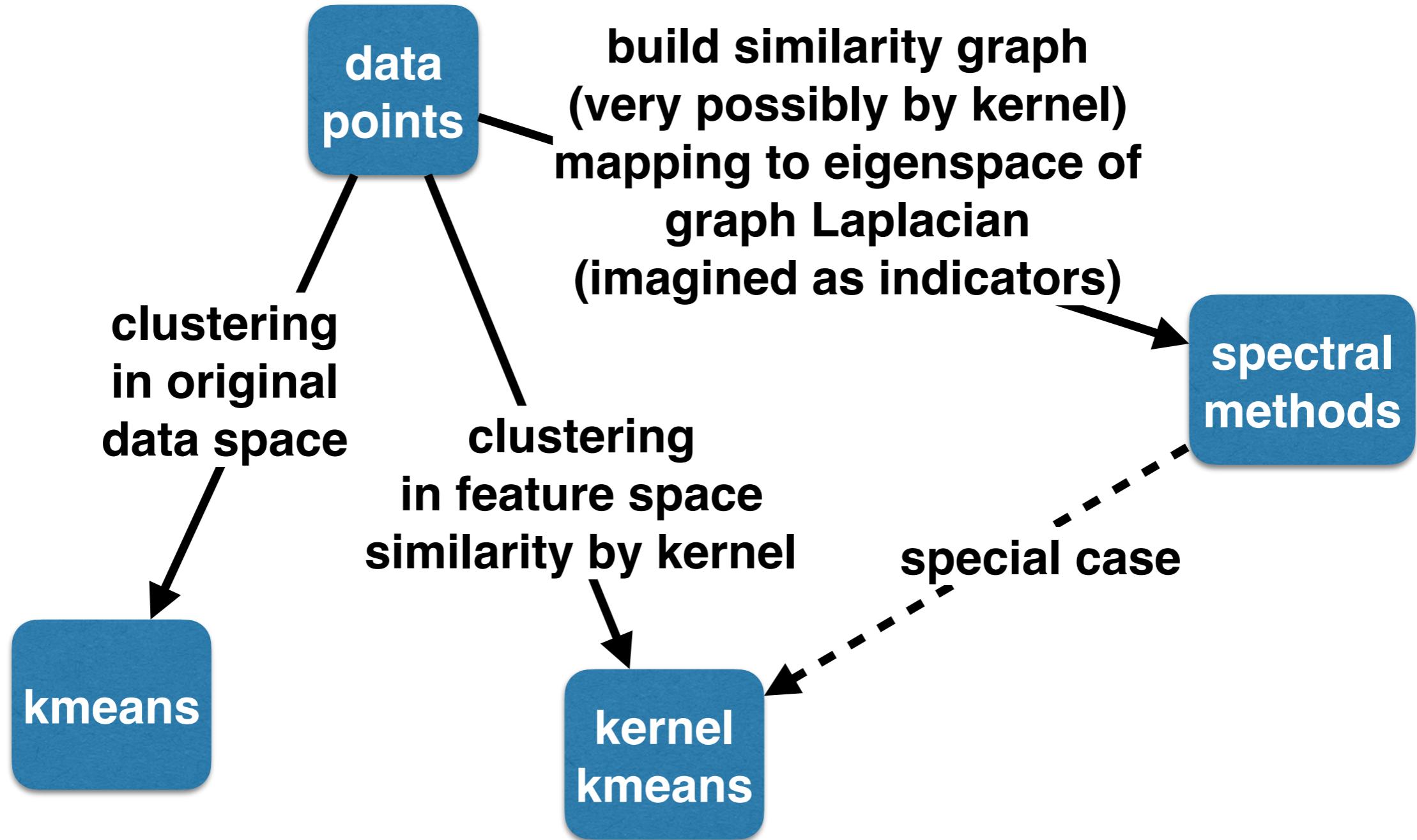
Similarity matrix



Second eigenvector of graph Laplacian



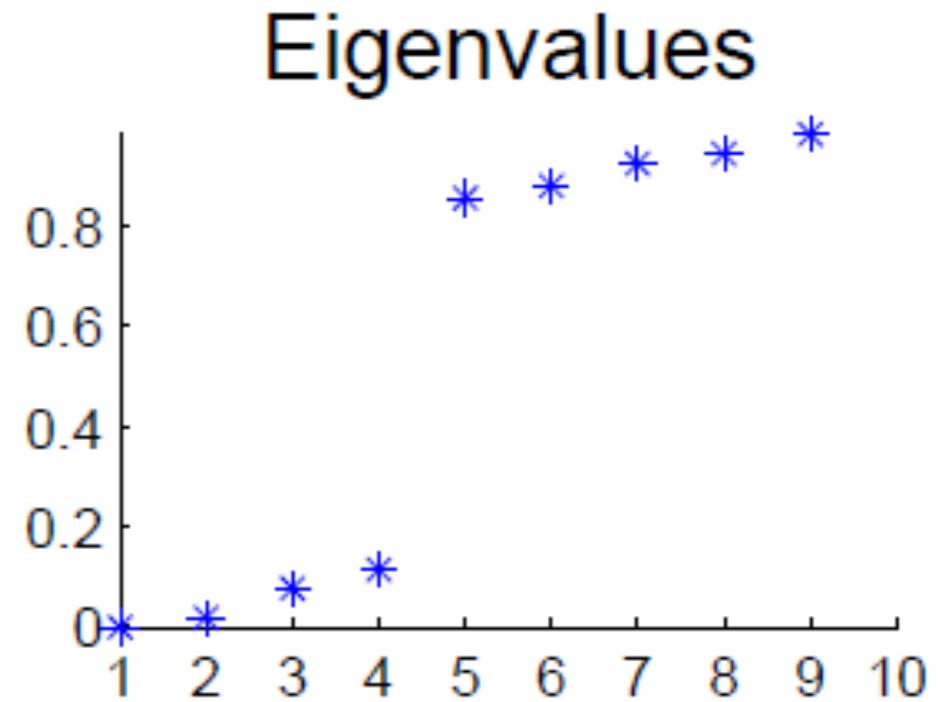
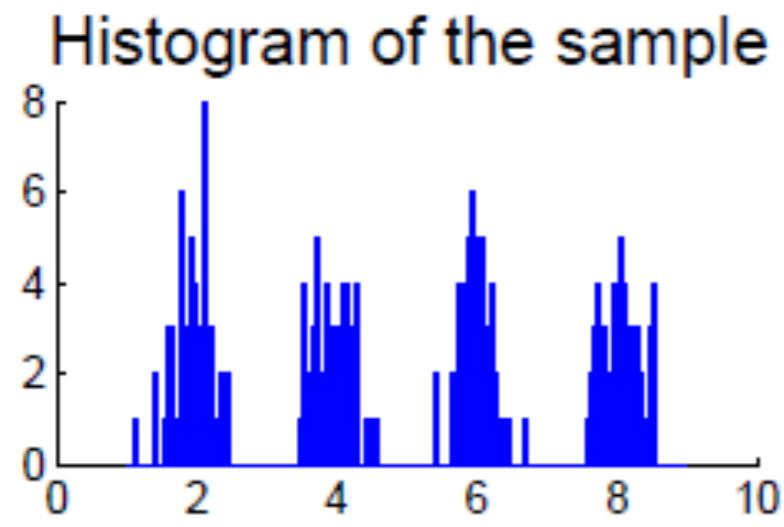
# Unprofessional Summary



# Some Issues Still

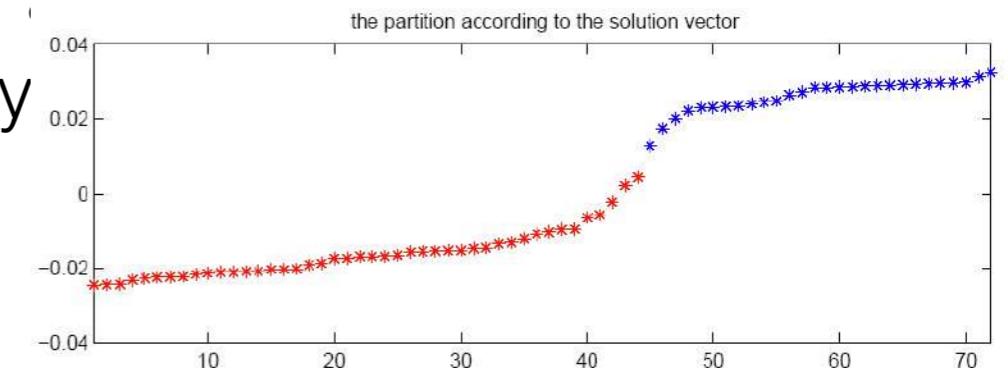
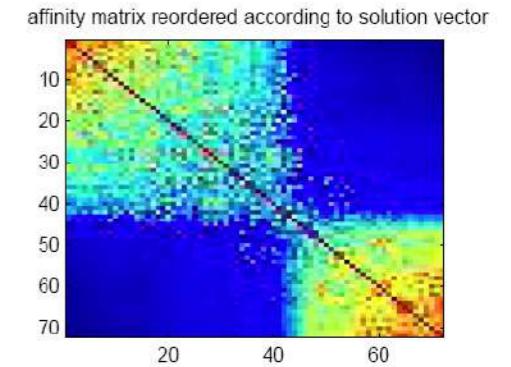
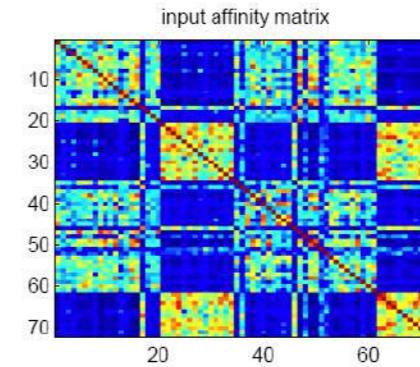
- Choice of number of cluster  $k$ 
  - Most stable clustering is usually given by the  $k$  that maximizes the **eigengap** (gap b/w consecutive eigenvalues)

$$\Delta_k = |\lambda_k - \lambda_{k-1}|$$

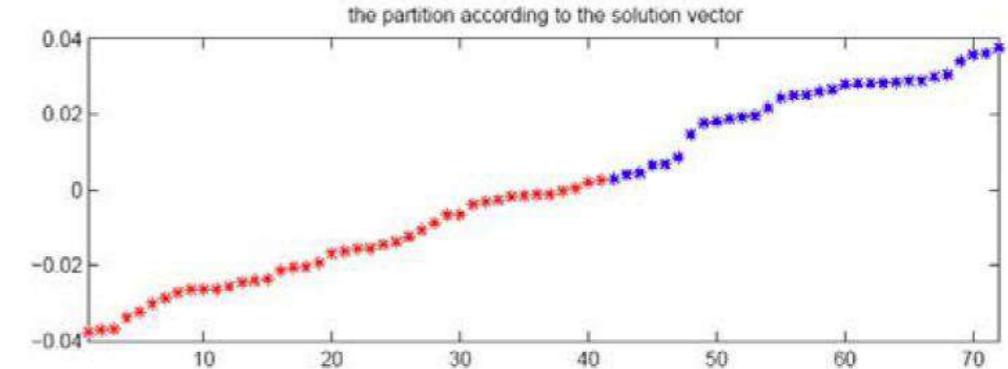
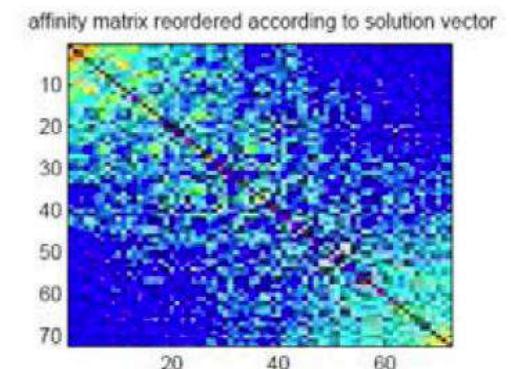
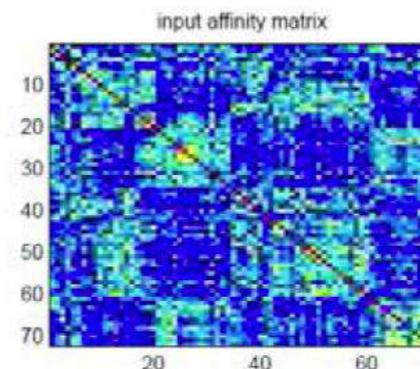


# Some Issues Still

- Choice of number of cluster  $k$   
(most common problem for clustering)
  - Most stable clustering is usually given by the  $k$  that maximizes the **eigengap**
- Choice of similarity  
(common problem for kernel methods)
  - choice of kernel function
  - for Gaussian kernels, choice of  $\sigma$



Good similarity measure



Poor similarity measure

# Determine the Parameters of Kernel Functions

- Yes, in **supervised** setting!
- Still remember our memory-based method? GPR and SVM?
  - **GPR**: can be estimated by **ARD** (Automatic Relevance Determination) for points  $x_n$  and  $x_m$  that are similar, the corresponding values  $y(x_n)$  and  $y(x_m)$  will be more strongly correlated than for dissimilar points.
  - **SVM**: usually estimated by **cross-validation** (please check **libsvm**)
  - If we know the **ground-truth clustering** (source of supervision!), you can also **learn the similarity function** (built by kernel most of time)
    - ☞ a classical paper:  
Bach & Jordan, “Learning Spectral Clustering”, NIPS’03





# Learning Spectral Clustering



- Cost for Normalized Cut

$$C(E, W) = \sum_{r=1}^R \frac{e_r^\top (D - W)e_r}{e_r^\top De_r}$$

- $E = (e_1 \ e_2 \dots \ e_R)$ ,  $e_r$  is indicator vector (length: #data  $P$ ) for  $r$ -th cluster
- Define a objective function  $J$

$$J(W, E) = R - \sum_r \frac{e_r^\top D^{1/2} U(W) U(W)^\top D^{1/2} e_r}{e_r^\top De_r}$$

- $U \in \mathbb{R}^{P \times R}$  : any orthonormal basis of  
 $R$ -th principal subspace of  $D^{-1/2} L D^{-1/2}$

Please refer to paper for more details on derivation if you are interested



# Learning Spectral Clustering



- Define a objective function  $J$

$$J(W, E) = R - \sum_r \frac{e_r^\top D^{1/2} U(W) U(W)^\top D^{1/2} e_r}{e_r^\top D e_r}$$

- optimizing w.r.t.  $E$  gives a weighted Kmeans algorithm!**

**Input:** Similarity matrix  $W \in \mathbb{R}^{P \times P}$ .

**Algorithm:**

1. Compute first  $R$  eigenvectors  $U$  of  $D^{-1/2} W D^{-1/2}$  where  $D = \text{diag}(W\mathbf{1})$ .
2. Let  $U = (u_1, \dots, u_P)^\top \in \mathbb{R}^{P \times R}$  and  $d_p = D_{pp}$ .
3. Initialize partition  $A$ . remind you kernel kmeans?
4. Weighted  $K$ -means: While partition  $A$  is not stationary,
  - a. For all  $r$ ,  $\mu_r = \sum_{p \in A_r} d_p^{1/2} u_p / \sum_{p \in A_r} d_p$
  - b. For all  $p$ , assign  $p$  to  $A_r$  where  $r = \arg \min_{r'} \|u_p d_p^{-1/2} - \mu_{r'}\|^2$

**Output:** partition  $A$ , distortion measure  $\sum_r \sum_{p \in A_r} d_p \|u_p d_p^{-1/2} - \mu_r\|^2$

Please refer to paper for more details on derivation if you are interested



# Learning Spectral Clustering



- Define a objective function  $J$

$$J(W, E) = R - \sum_r \frac{e_r^\top D^{1/2} U(W) U(W)^\top D^{1/2} e_r}{e_r^\top D e_r}$$

- **optimizing w.r.t.  $W$  : learning similarity matrix**

- Given  $N$  datasets  $D_n$ , each  $D_n$  is composed of  $P_n$  points
- Each dataset is clustered, known partition  $E_n$
- The cost function is

**details in the paper, compare subspaces  
built by NCut & by groundtruth partition**

$$H(\alpha) = \frac{1}{N} \sum_n \underline{F(W_n(\alpha), \Pi_0(e_n, \alpha))} + \underline{C \|\alpha\|_1} \quad \begin{matrix} \text{feature} \\ \text{selection} \end{matrix}$$

$$k_\alpha(x, y) = \exp(-(x - y)^\top \text{diag}(\alpha)(x - y)) \quad \begin{matrix} \text{can be any} \\ \text{other kernels} \end{matrix}$$

Please refer to paper for more details on derivation if you are interested



# Learning Spectral Clustering

- Define a objective function  $J$

**differentiable NCut cost,  
differentiable feature representation  
end-to-end optimization**

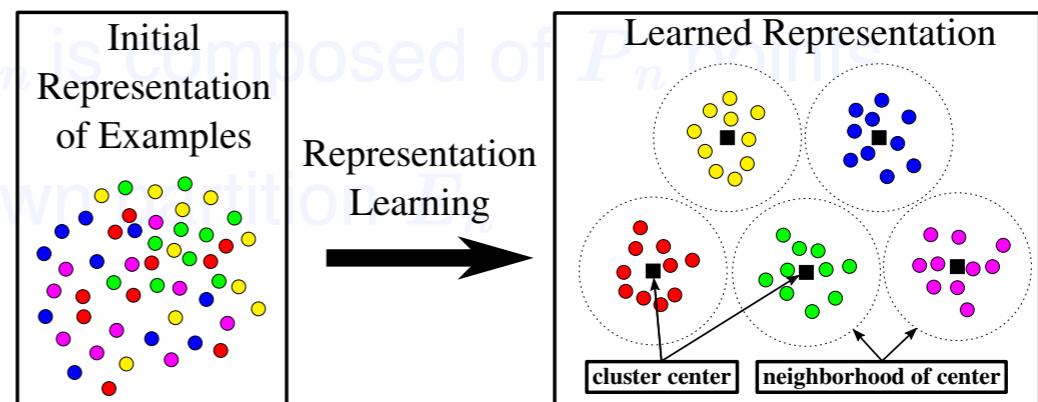
**ICML'17** Deep Spectral Clustering Learning

Marc T. Law<sup>1</sup> Raquel Urtasun<sup>1</sup> Richard S. Zemel<sup>1,2</sup>

optimizing w.r.t.  $W$ : learning similarity matrix

## Abstract

Clustering is the task of grouping a set of examples so that similar examples are grouped into the same cluster while dissimilar examples are in different clusters. The quality of a clustering depends on two problem-dependent factors which are *i*) the chosen similarity metric and *ii*) the data representation. Supervised cluster-



$$H(\alpha) = \frac{1}{N} \sum_n F(W_n(\alpha), \Pi_0(e_n, \alpha)) + C\|\alpha\|_1$$

$$k_\alpha(x, y) = \exp(- (x - y)^\top \text{diag}(\alpha)(x - y))$$

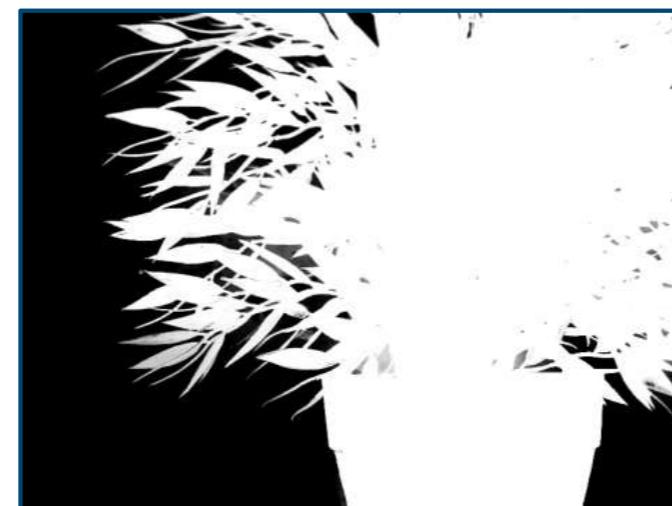
can be any other kernels

Please refer to paper for more details on derivation if you are interested



# Application of Spectral Clustering: Alpha Matting

- Image Matting (**Levin et al., TPAMI'08**)
  - A process to extract foreground objects from an image, along with an alpha matte (the opacity of the foreground color)



input image

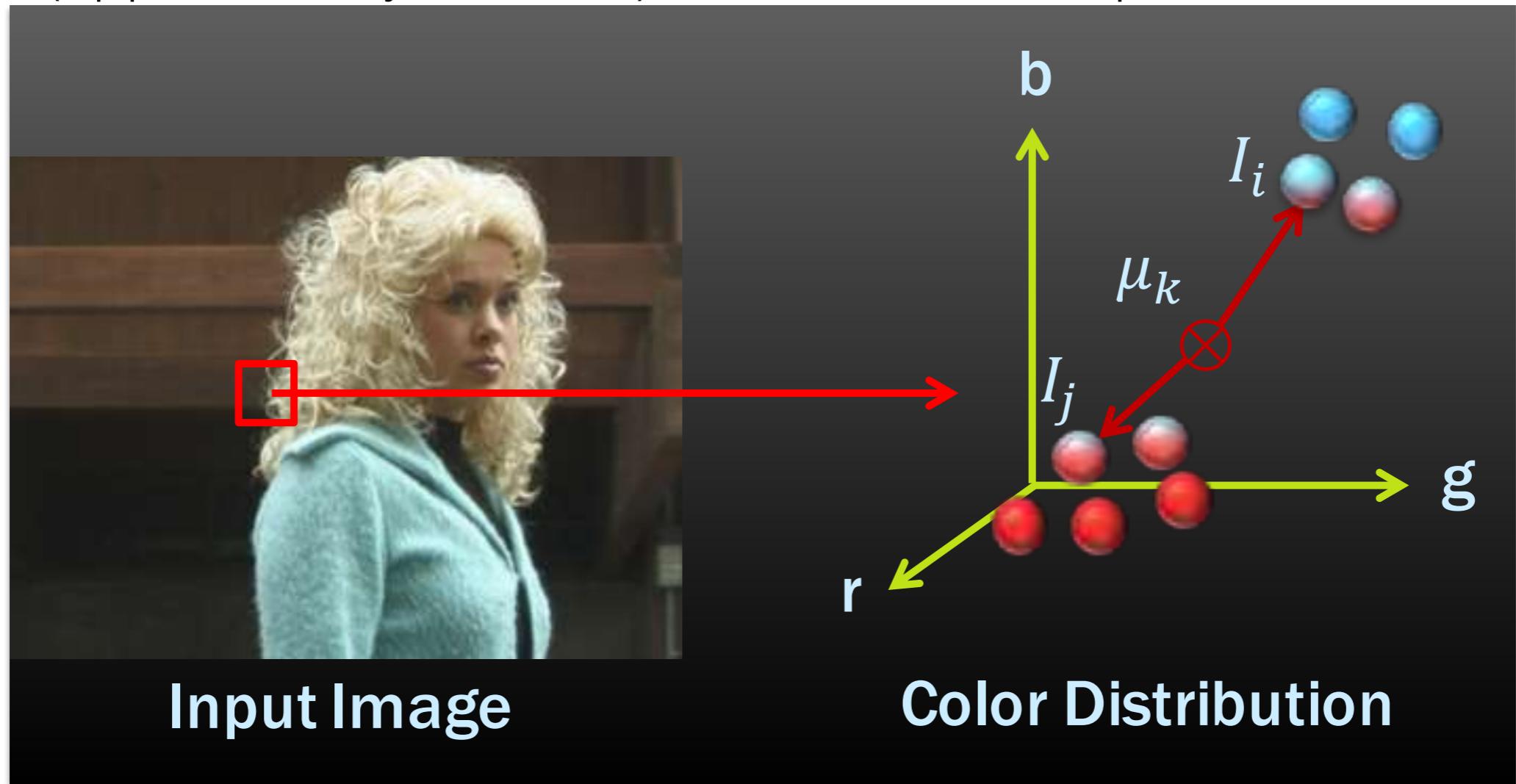
alpha matte

extracted foreground



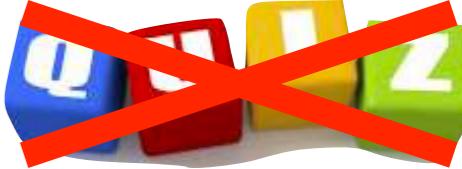
# Application of Spectral Clustering: Alpha Matting

- Matting Laplacian, with basic assumption: **color line model**
  - the foreground (or background) colors in a local window lie on a single line (approximately constant) in the RGB color space



$$\sum_{k|(i,j) \in \omega_k} \left( \delta_{ij} - \frac{1}{|\omega_k|} (1 + (\mathbf{I}_i - \boldsymbol{\mu}_k)^T (\boldsymbol{\Sigma}_k + \frac{\varepsilon}{|\omega_k|} \mathbf{U})^{-1} (\mathbf{I}_j - \boldsymbol{\mu}_k))) \right)$$

define similarity



# Application of Spectral Clustering: Alpha Matting



Input Image



Matting  
Laplacian



Smallest Eigenvectors

K-means  
Clustering  
&  
Linear  
Transformation



Matting Components



# Application of Spectral Clustering: Alpha Matting



⋮

⋮

⋮

Smallest Eigenvectors

$$E = [e^1 \quad \dots \quad e^s]$$

K-means

$$m^k$$

Projection into Eigen Space

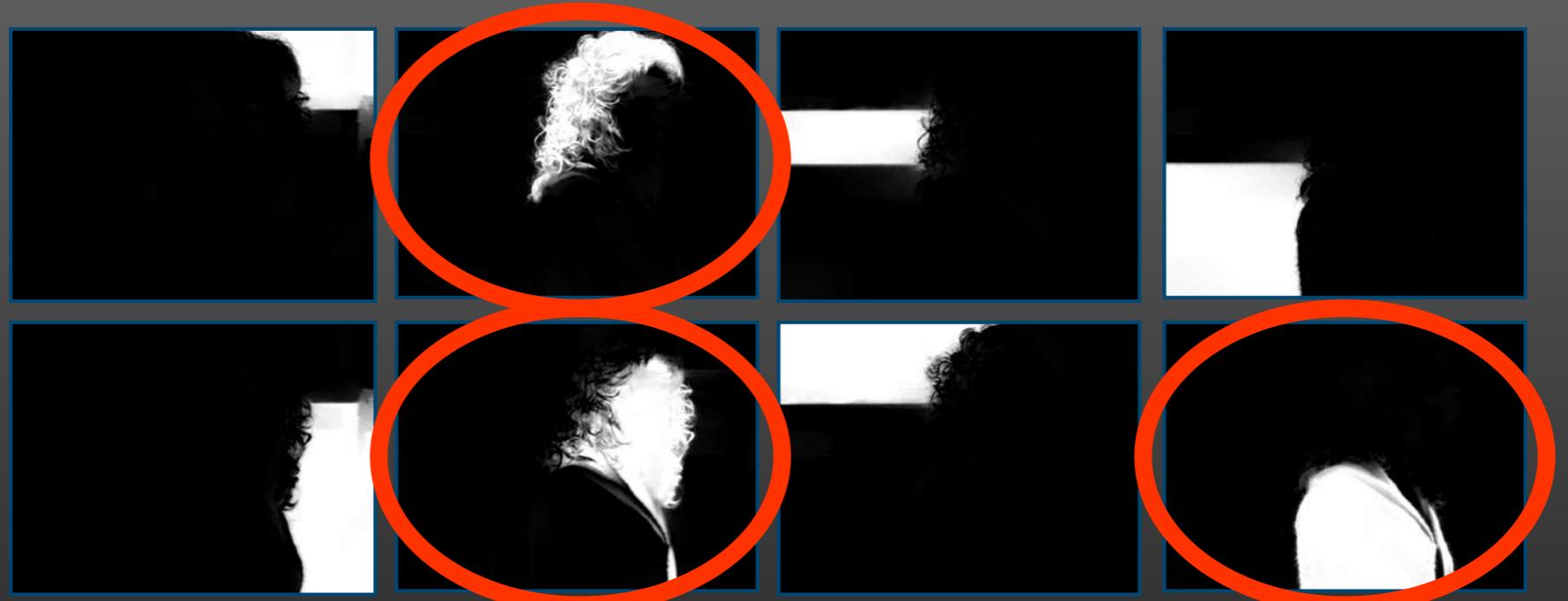
$$\alpha^k = E E^T m^k$$



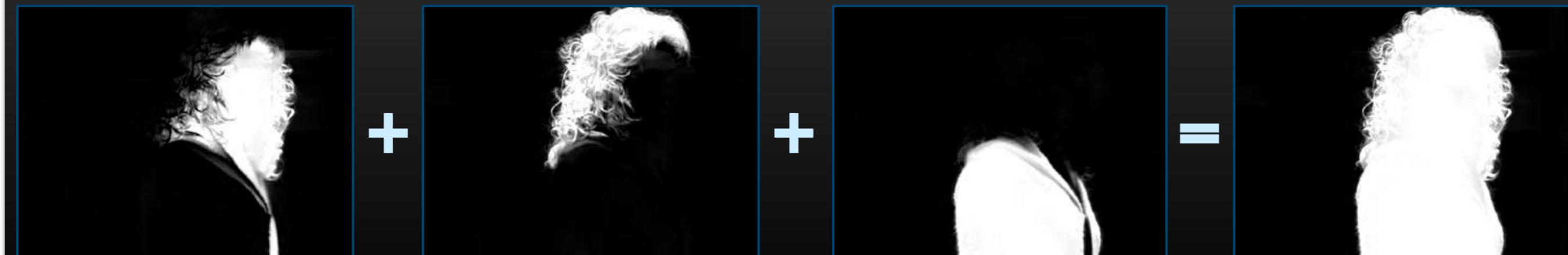
# Application of Spectral Clustering: Alpha Matting



Input Image



Matting Components



Selected Matting Components

Alpha Matte



# Application of Spectral Clustering: Alpha Matting

## Alpha Matte Generation

$$\vec{\alpha} = [\vec{\alpha}^1 \quad \dots \quad \vec{\alpha}^k] \vec{b}$$

$\vec{b}$ : Combination Vector

## Matting cost function

$$J(\vec{\alpha}) = \vec{\alpha}^T L \vec{\alpha}$$

$$= \vec{b}^T [\vec{\alpha}^1 \quad \dots \quad \vec{\alpha}^k]^T L [\vec{\alpha}^1 \quad \dots \quad \vec{\alpha}^k] \vec{b}$$

$$= \vec{b}^T \Phi \vec{b}$$

Evaluating All Grouping Hypothesis to Derive the Optimal Alpha Matte



# Application of Spectral Clustering: Alpha Matting





# Application of Spectral Clustering: Alpha Matting

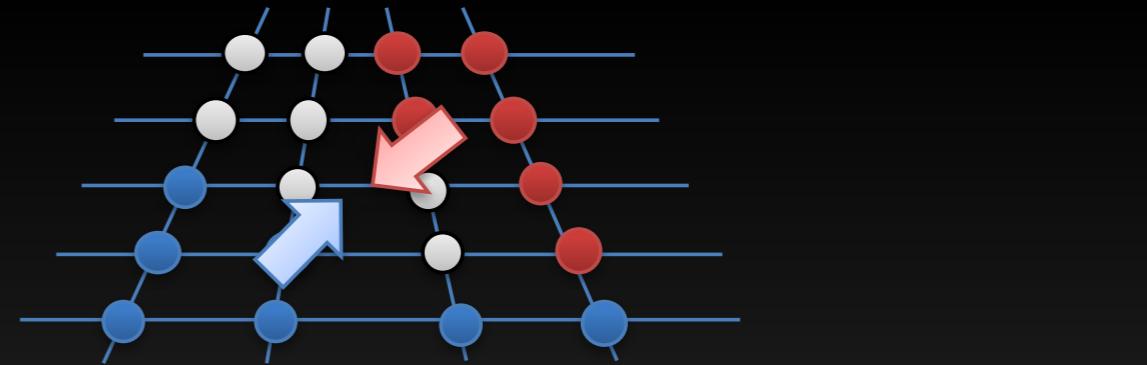
- Just for your reference: Supervised Matting (Close-Form Matting)



Input Image



User's Guidance,  $\vec{\beta}$



- Foreground Pixel
- Unknown Pixel
- Background Pixel

## Cost Function for Supervised Matting

$$E(\vec{\alpha}) = \underbrace{\vec{\alpha}^T L \vec{\alpha}}_{\text{Affinity Cost}} + \underbrace{(\vec{\alpha} - \vec{\beta})^T \Lambda (\vec{\alpha} - \vec{\beta})}_{\text{Data Cost}}$$

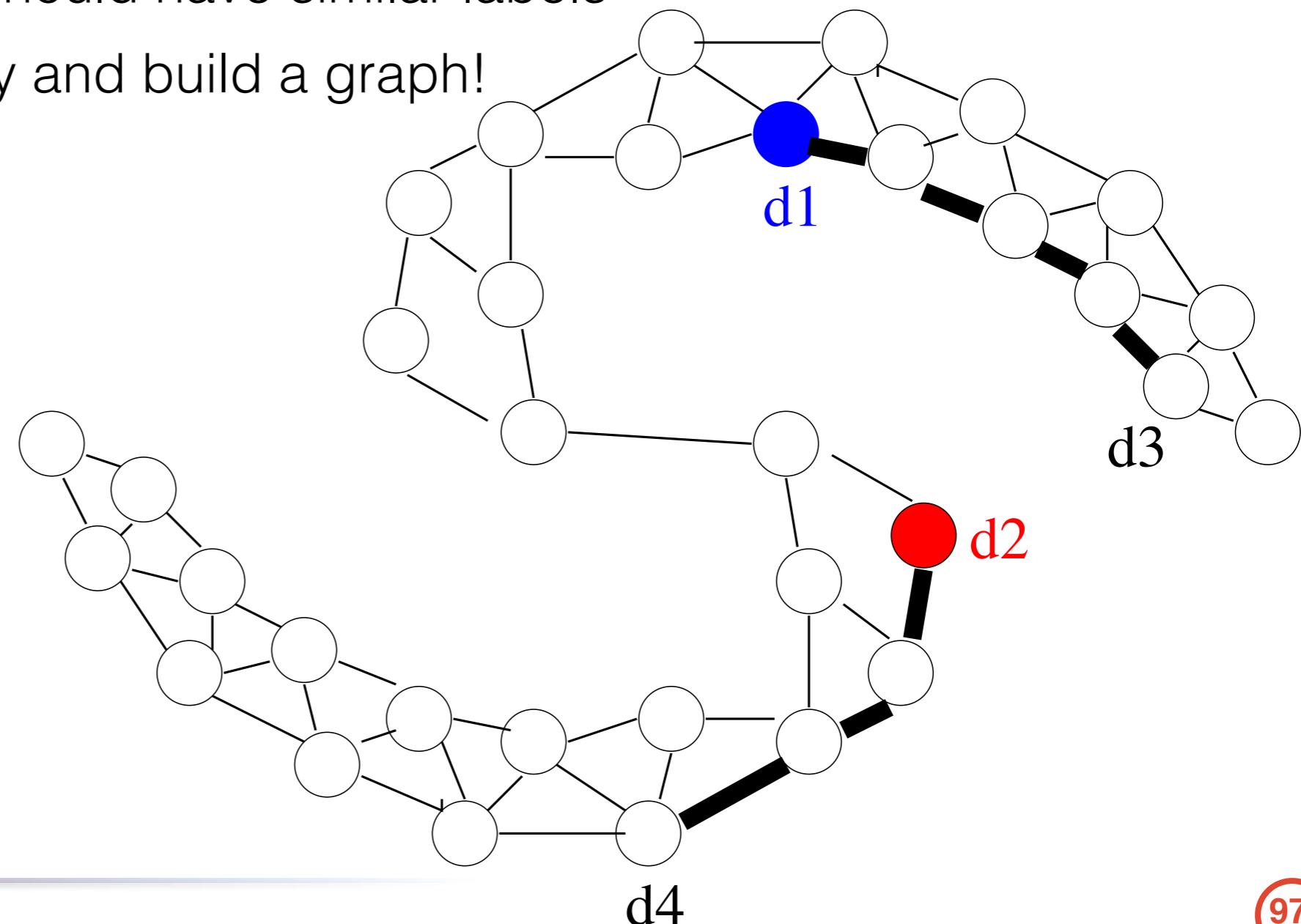
## Optimal Solution

$$\vec{\alpha}^* = (L + \Lambda)^{-1} \Lambda \vec{\beta}$$

	Foreground	Background	Unknown
$\vec{\beta}(i)$	1	0	-
$\Lambda(i, i)$	1	1	0

# Spectral Clustering for Semi-Supervised Learning: Label Propagation

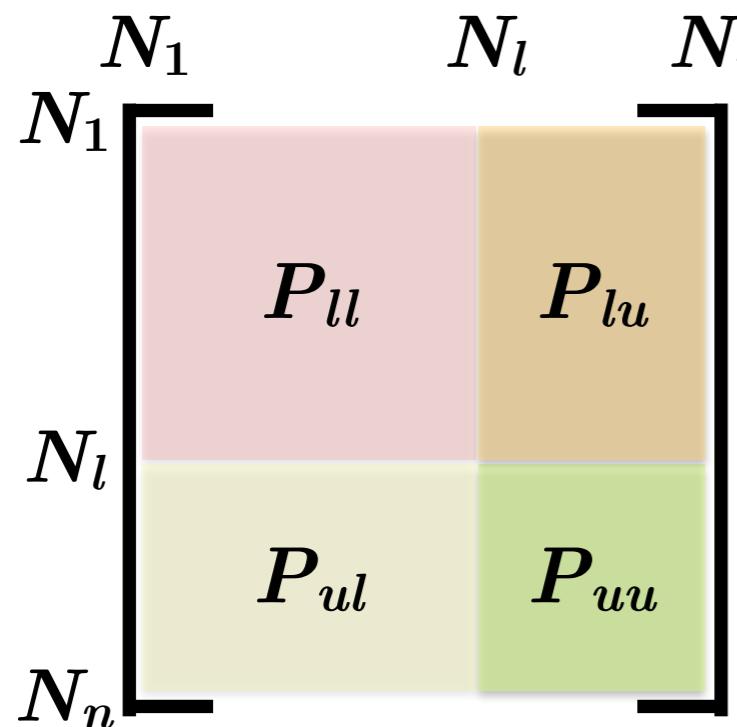
- Semi-supervised setting: only partial data points/nodes are with labels! want to know other unlabeled data their labels :D
  - Basic idea: want to propagate labels from labeled to unlabeled data
  - Similar data points should have similar labels
  - ... measure similarity and build a graph!



# Spectral Clustering for Semi-Supervised Learning: Label Propagation

- Define a transition probability matrix  $P$

$$P_{ij} = P(i \rightarrow j) = \frac{w_{ij}}{\sum_{k=1}^n w_{ik}} \quad \text{probability of jumping from node } i \text{ to } j$$



$N_1 \dots N_l = \text{Labeled data}$

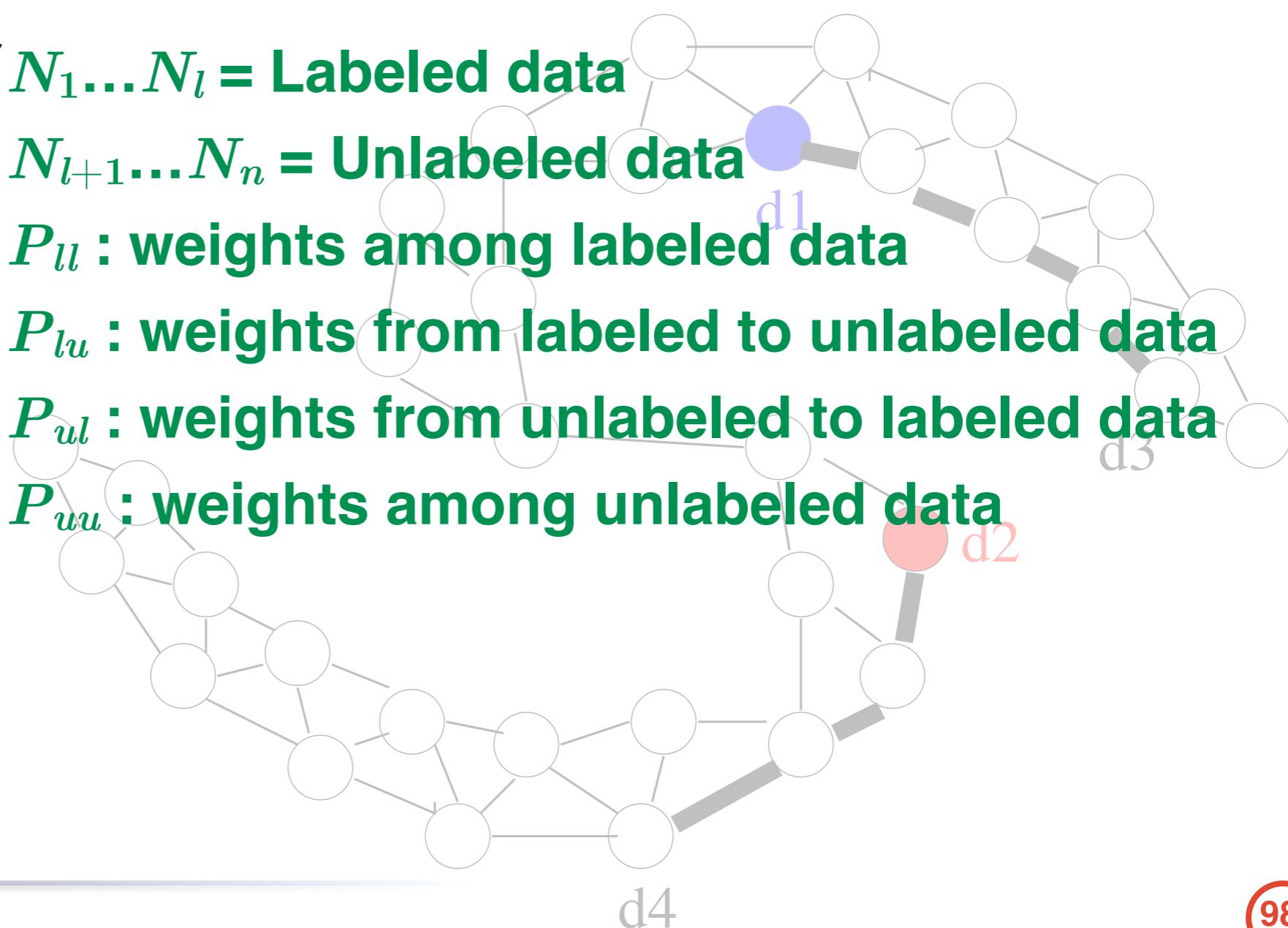
$N_{l+1} \dots N_n = \text{Unlabeled data}$

$P_{ll}$  : weights among labeled data

$P_{lu}$  : weights from labeled to unlabeled data

$P_{ul}$  : weights from unlabeled to labeled data

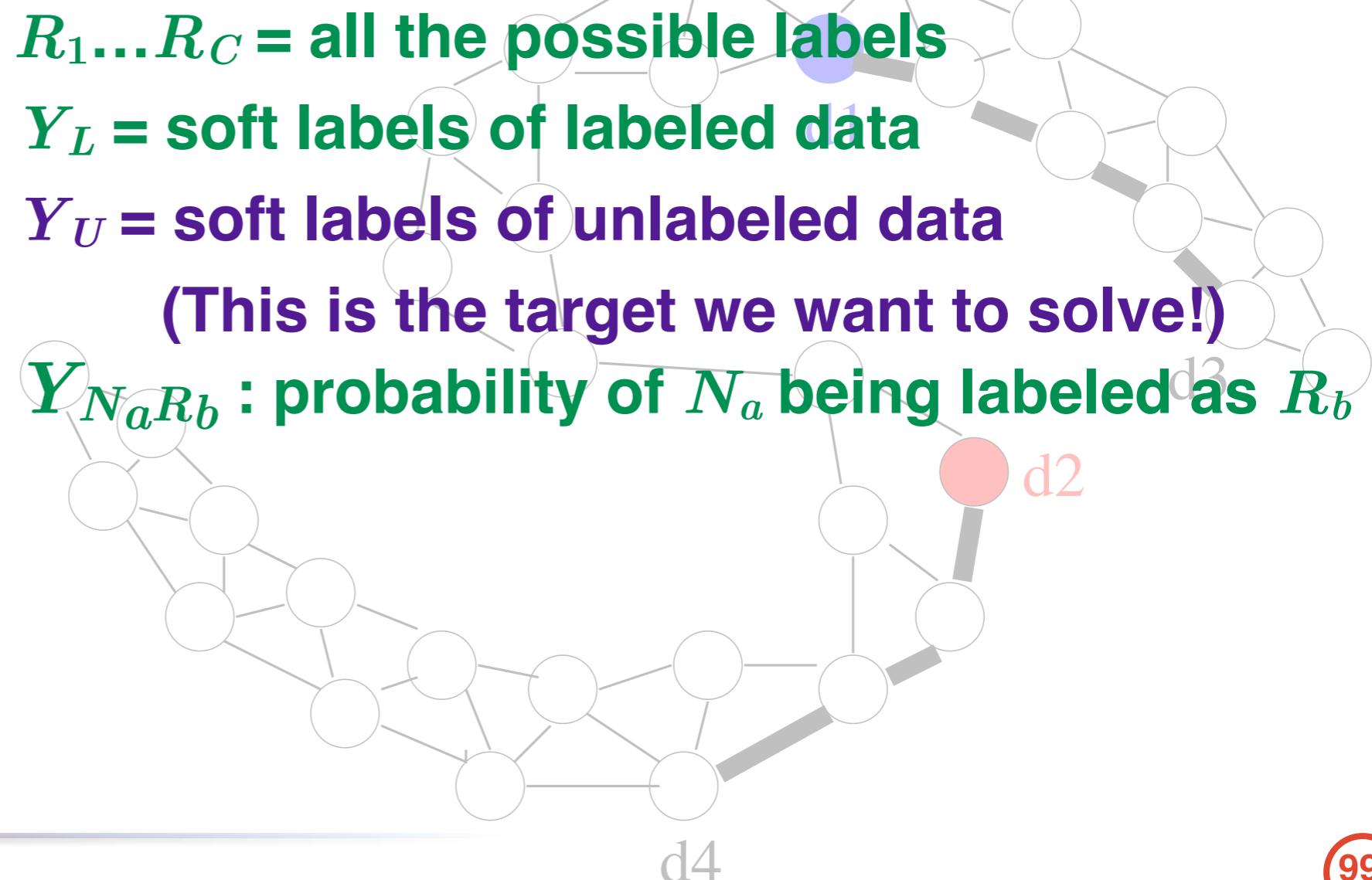
$P_{uu}$  : weights among unlabeled data



# Spectral Clustering for Semi-Supervised Learning: Label Propagation

- Define another label matrix  $\mathbf{Y}$ , holding the soft probabilities of each data point.

$$\begin{bmatrix} R_1 \\ N_1 \\ \vdots \\ N_l \\ \vdots \\ N_n \end{bmatrix} \quad \begin{bmatrix} Y_L \\ Y_U \end{bmatrix} \quad \begin{bmatrix} R_C \\ \vdots \end{bmatrix}$$



# Spectral Clustering for Semi-Supervised Learning: Label Propagation

- Label propagation step:
  - Update  $\mathbf{Y}$  during each iteration  $z$  :

$$\mathbf{Y}^{z+1} \leftarrow P\mathbf{Y}^z$$

- Since we only care about  $\mathbf{Y}_U$  :

$$\mathbf{Y}_U \leftarrow P_{uu}\mathbf{Y}_U + P_{ul}\mathbf{Y}_L$$

- After convergence :

$$\mathbf{Y}_U = (I - P_{uu})^{-1}P_{ul}\mathbf{Y}_L$$

**potential problem here:  
it fixes the given labels  $\mathbf{Y}_L$ ,  
what if some labels are wrong?  
we want to be flexible and disagree  
with given labels occasionally**

# Spectral Clustering for Semi-Supervised Learning: Label Propagation

- We can rephrase the problem as  
**“clustering subject to constraints”**

$$\min_f \sum_{i=1}^l \underbrace{(f(x_i) - y_i)^2}_{\text{label constraints but allow flexibility}} + \lambda f^\top \underbrace{\Delta f}_{\text{graph Laplacian}}$$

**the optimal solution is linked to label propagation by using normalized graph Laplacian**

You can refer to

1. Zhou et al., Learning with local and global consistency, NIPS'04
2. Wang et al., Flexible constrained spectral clustering, KDD'10

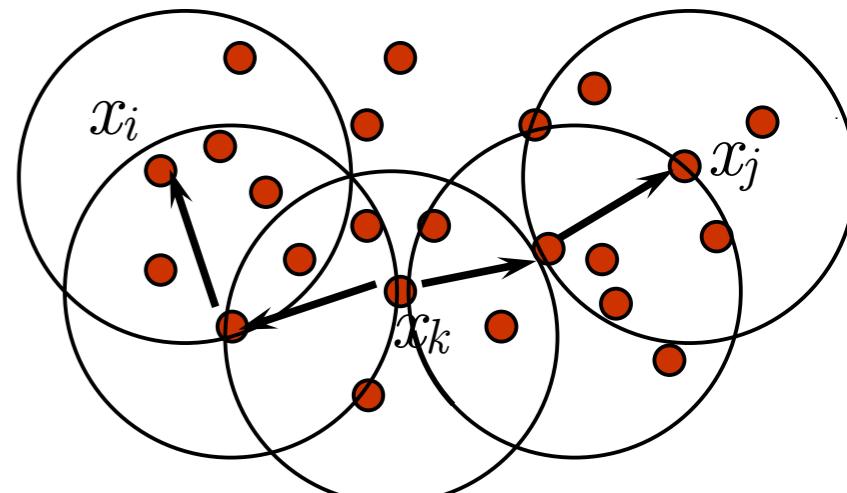
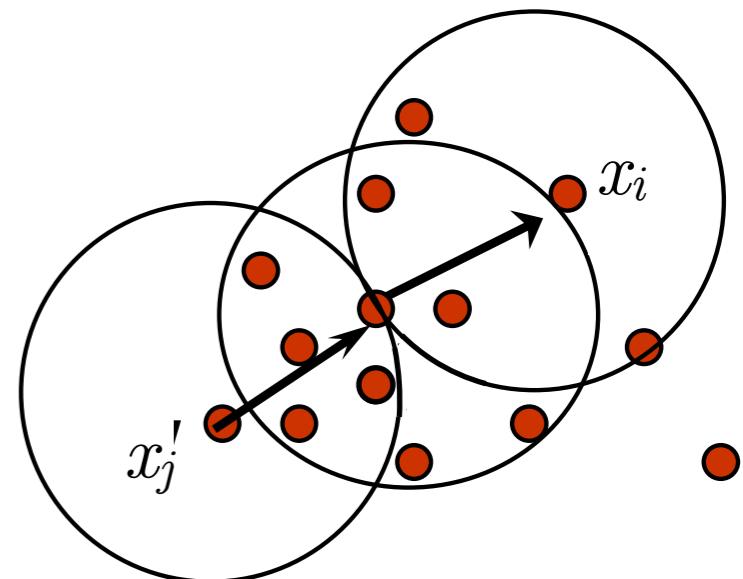
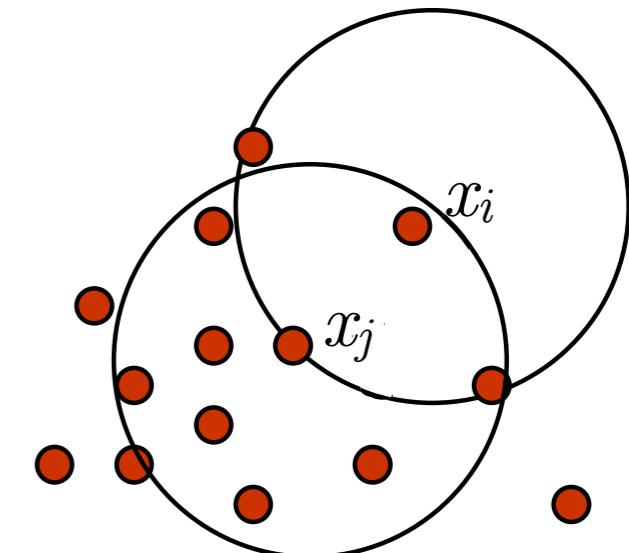
if you want to know more details about connection between label propagation and spectral clustering for semi-supervised learning

# DBSCAN

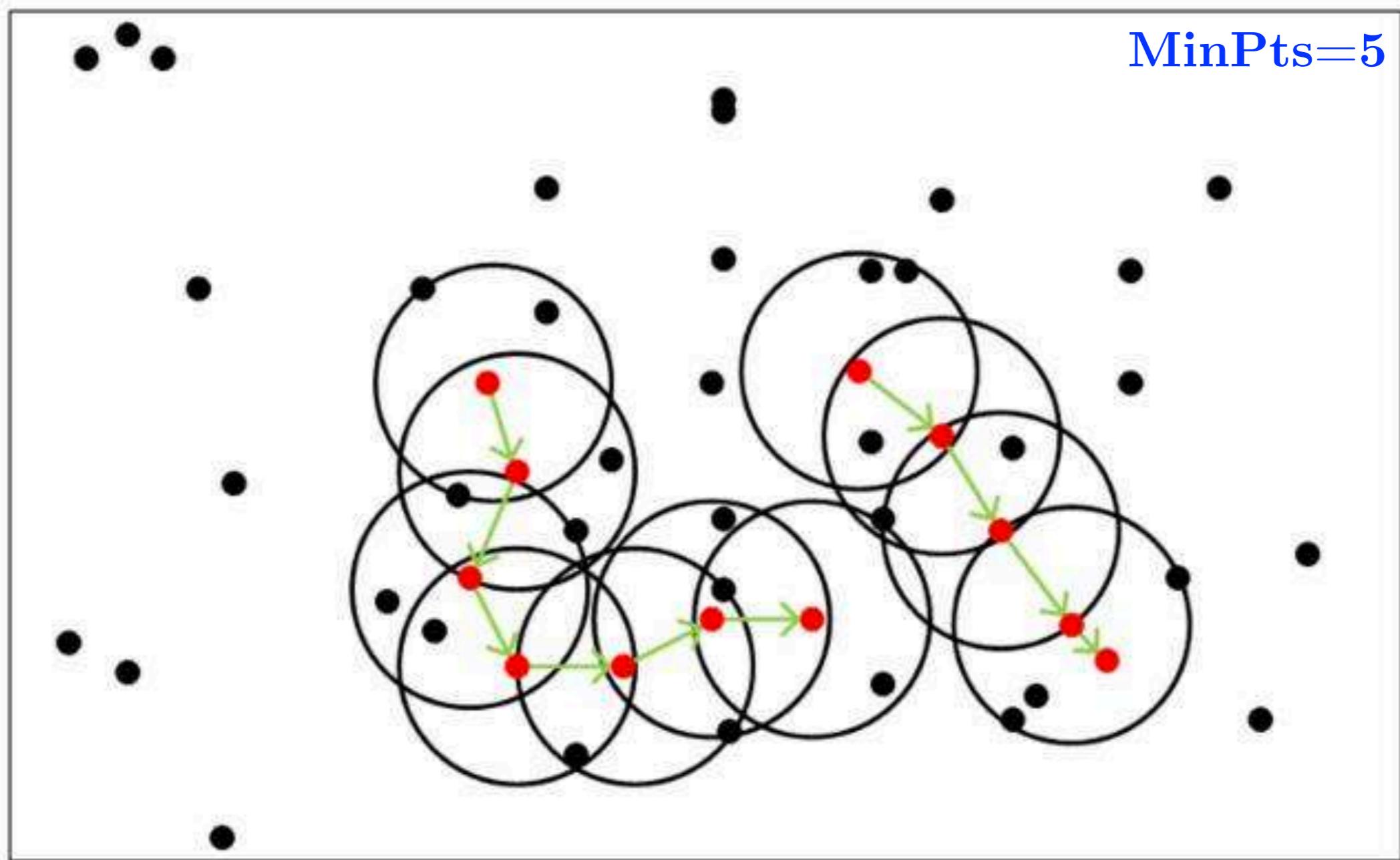
- Density-Based Spatial Clustering of Applications with Noise
  - Assume data  $D = (x_1, x_2, \dots, x_m)$
  - For  $x_j \in D$ , its  **$\varepsilon$ -neighbors**  $N_\varepsilon(x_j) = \{x_i \in D \mid d(x_i, x_j) < \varepsilon\}$
  - For  $x_j \in D$ , if its  $\varepsilon$ -neighbors  $N_\varepsilon(x_j)$  include at least **MinPts** data points, i.e.  $|N_\varepsilon(x_j)| \geq \text{MinPts}$ , then  $x_j$  is a **core point**
  - $x_j$  is a **border point** if its  $\varepsilon$ -neighbors include at least one core point
  - $x_j$  is a **noise/outlier point** if it is not core point nor border point

# DBSCAN

- $x_i$  is **directly density-reachable** from a point  $x_j$  if:  
 $x_i$  is within  $N_\varepsilon(x_j)$  and  $x_j$  is a **core point**  
 (note this property is asymmetric)
- $x_i$  is **density-reachable** from a point  $x_j$  if there is  
**a chain of points**  $p_1, p_2, \dots, p_T$  **satisfying**  
 $p_1=x_j$ ,  $p_T=x_i$  and  $p_{t+1}$  is directly  
**density-reachable** from  $p_t$   
 (i.e.  $p_1, p_2, \dots, p_{T-1}$  are all core points)  
 (note this property is asymmetric)
- $x_i$  is **density-connected** from a point  $x_j$  if there is  
**a point**  $x_k$  such that both  $x_i$  and  $x_j$  are  
**density-reachable** from  $x_k$   
 (note this property is symmetric)



# DBSCAN



MinPts=5

- red dots are **core points**, since each has at least  $\text{MinPts}$   $\epsilon$ -neighbors
- core points linked by green arrows construct a **chain with density-reachable property**
- all pairs of dots within  $\epsilon$ -neighbors of that chain are **density-connected**

# DBSCAN

## → Repeat

- arbitrarily select a point  $x$
- If  $x$  has been assigned to a cluster or labeled as noise, **continue**
- **else**
  - **if**  $|N_{\varepsilon}(x)| < \text{MinPts}$ , label  $x$  as border point or noise
  - **else**, label  $x$  as core point
    - form a new cluster  $C_x$ , merge all  $\varepsilon$ -neighbors into cluster  $C_x$
    - for all  $x' \in N_{\varepsilon}(x)$ , if  $|N_{\varepsilon}(x')| \geq \text{MinPts}$ ,  
then merge all *unprocessed* data points  $\in N_{\varepsilon}(x')$  into cluster  $C_x$
- **Until** all data points have been processed

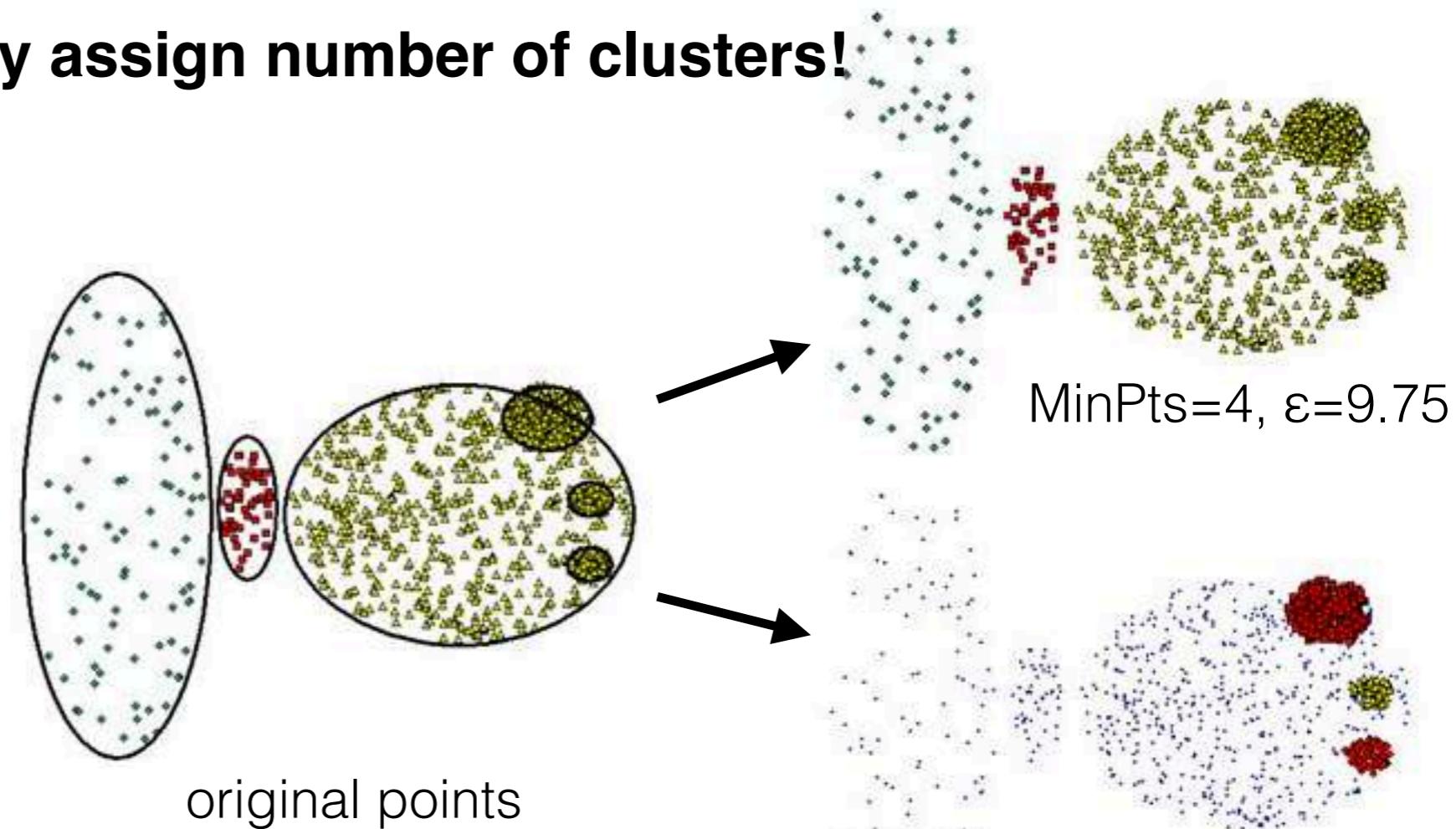
# DBSCAN

- **Pros**

- It is a density-based clustering method, it supports clustering on dataset with arbitrary shape (such as non-convex boundaries)
- It is able to discover noise/outlier points, insensitive to noise
- Insensitive to the initialization (in comparison to kmeans)
- **No need to manually assign number of clusters!**

- **Cons**

- problematic on data with varying density
- more parameters to be tuned in order to get nice results



# Hierarchical Clustering

---

- **agglomerative**: start with a partition of  $V$  into one-elementary subsets and recursively join clusters which are *maximally similar*,
- **divisive**: start with one cluster containing all elements of  $V$  and recursively 2-cut (bipartition) clusters that are *maximally dissimilar*.
- **Requirement**:
  - a formal definition of **similarity**
- **Result**:
  - a **recursive bipartition** of set  $V$
  - a **tree** with  $|V|$  leaves

# Hierarchical Clustering

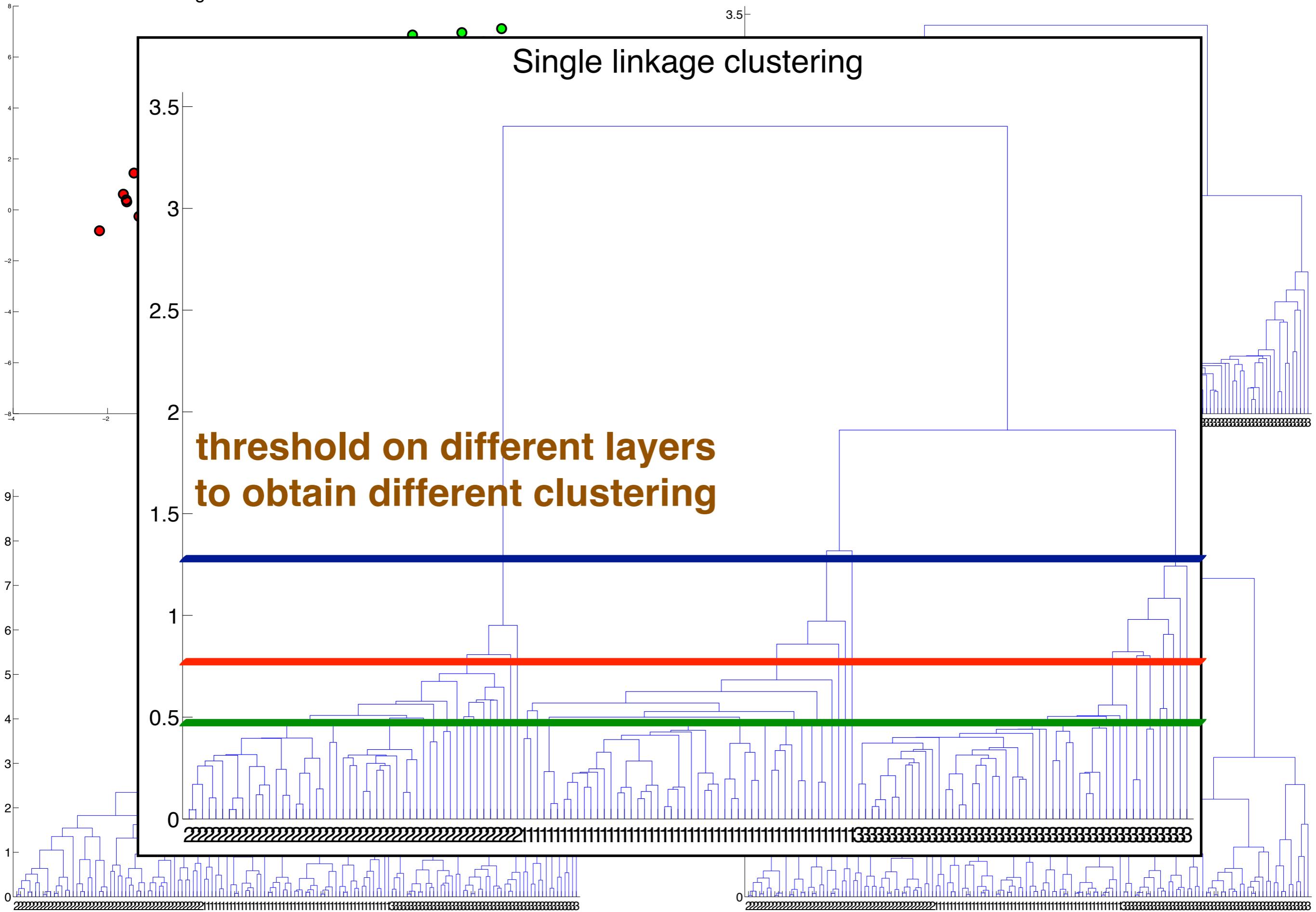
- **agglomerative hierarchical clustering**
  - Given:  $V$  with  $|V| = n$ , metric  $d$  on every pair of  $V$
  - initialize: we have  $n$  clusters denoted by  $C_i$ ,  $i = 1, 2, \dots, n$
  - **do**
    - compute for all pairs of clusters their similarities  $d_{ij} = D(C_i, C_j)$
    - Find the nearest two clusters  $C_i$  and  $C_j$  within all clusters
    - Combine  $C_i$  and  $C_j$  into a new cluster, keep others unchanged
  - **while** number of clusters  $> 1$
- **consecutively join clusters which are *most similar***
- **Question: how to measure similarity of cluster  $C_i$  and  $C_j$**

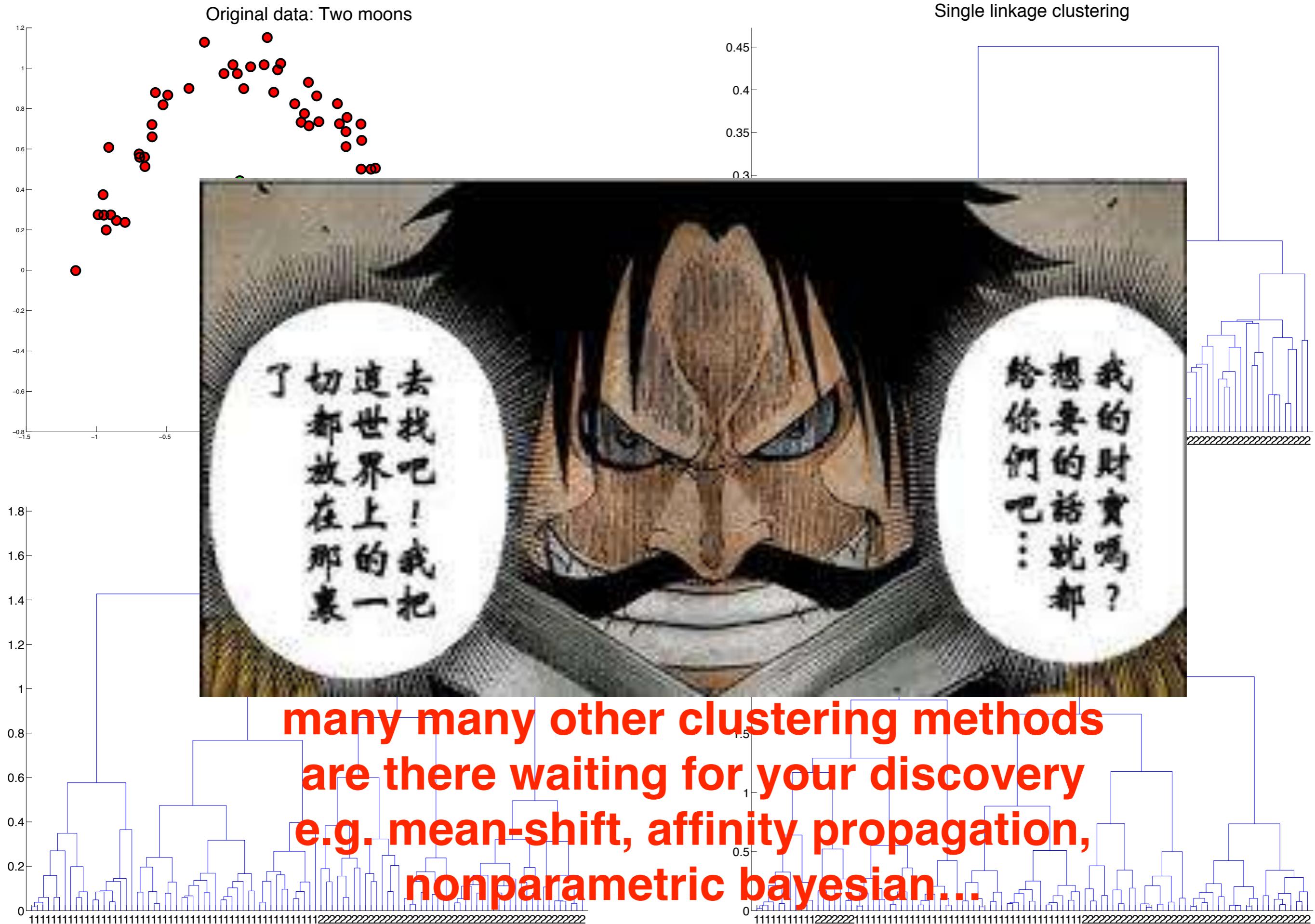
# Hierarchical Clustering

- **Question: how to measure similarity of cluster  $C_i$  and  $C_j$**
- **Single-linkage**  $d(C_i, C_j) = \min_{\mathbf{a} \in C_i, \mathbf{b} \in C_j} d(\mathbf{a}, \mathbf{b})$   
**shortest distance between members of these two clusters**
- **Average-linkage**  $d(C_i, C_j) = \sum_{\mathbf{a} \in C_i, \mathbf{b} \in C_j} \frac{d(\mathbf{a}, \mathbf{b})}{|C_i||C_j|}$   
**on average the points of both clusters are similar**
- **Complete-linkage**  $d(C_i, C_j) = \max_{\mathbf{a} \in C_i, \mathbf{b} \in C_j} d(\mathbf{a}, \mathbf{b})$   
**all points for both cluster are similar**

Original data: Three Gaussians

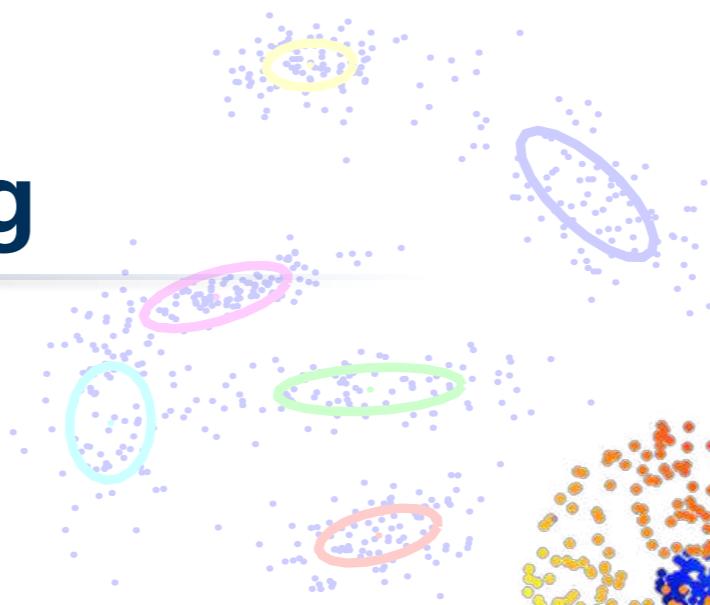
Single linkage clustering



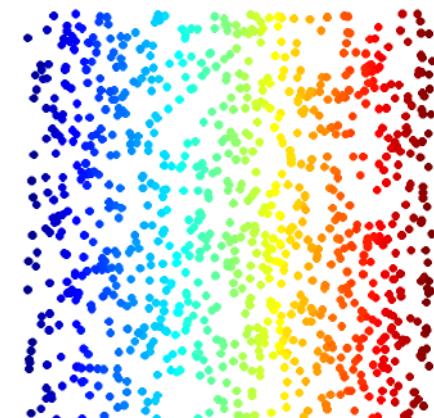
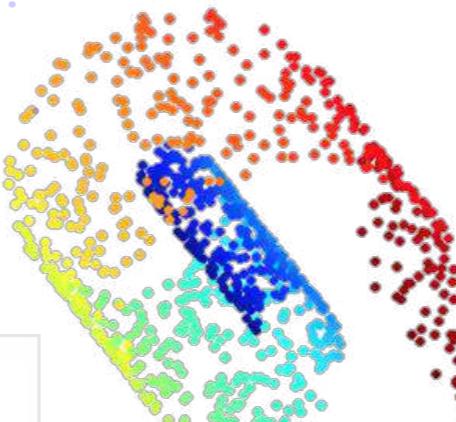
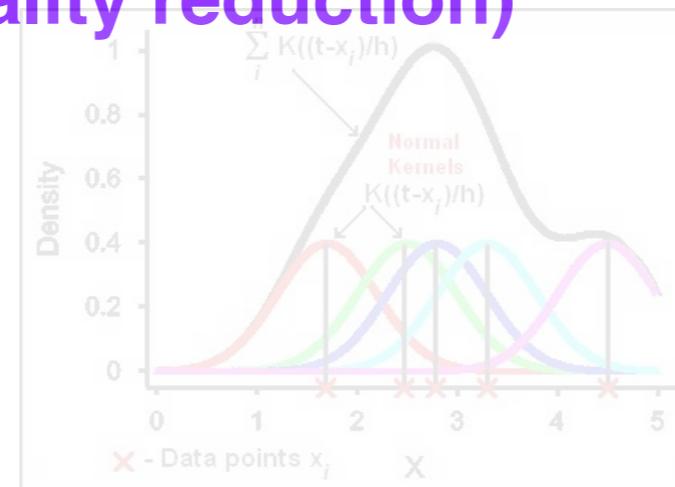


# Unsupervised Learning

✓ clustering



✓ embedding (dimensionality reduction)

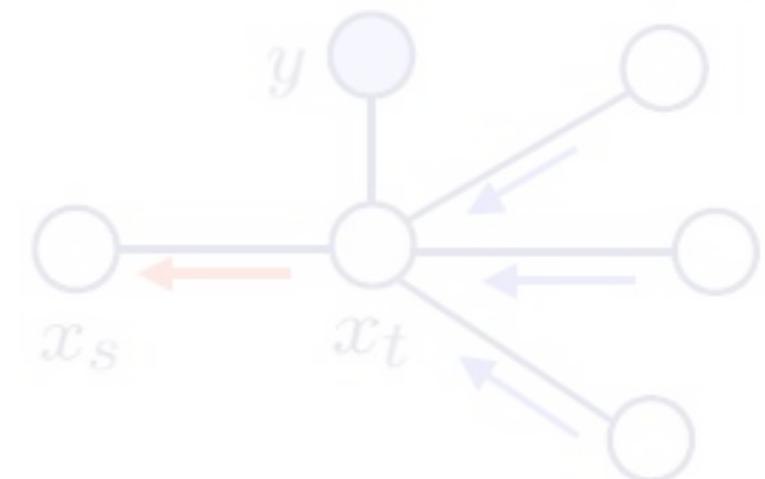


Looks like 3-D

Actually, 2-D

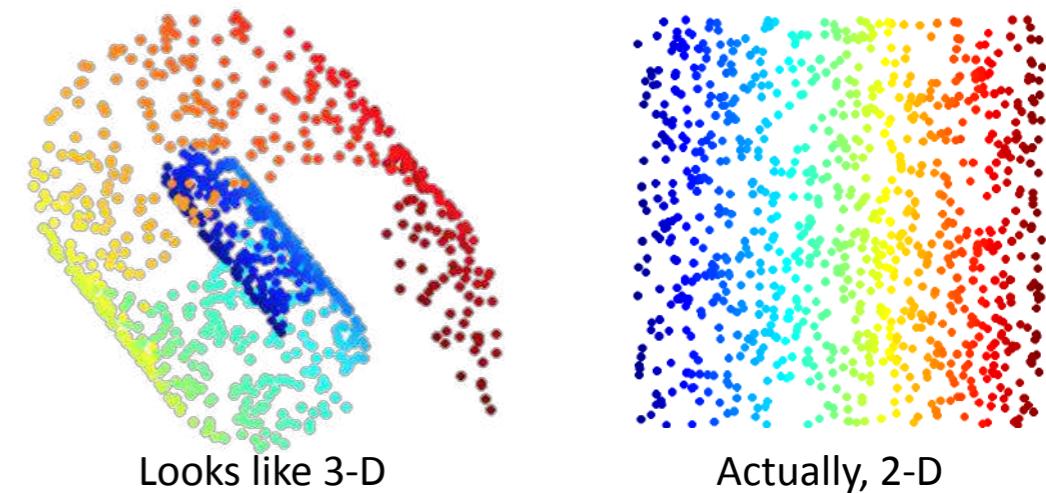
✓ density estimation

✓ finding good explanations  
(hidden causes) of data



# Dimensionality Reduction

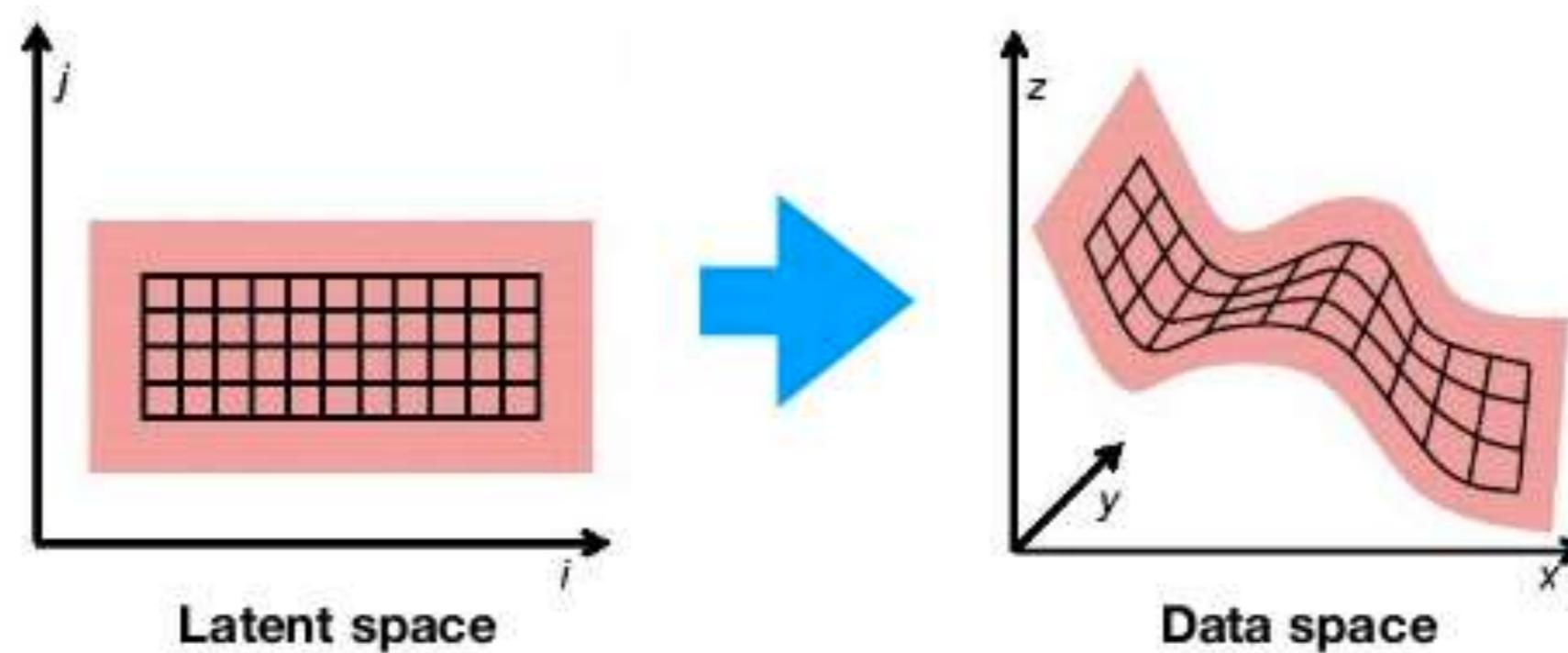
- Construction of a mapping  $\phi : \mathcal{X} \rightarrow \mathbb{R}^m$ , where the dimensionality  $m$  of the target space is usually much smaller than that of the input space  $\mathcal{X}$ . Generally, the mapping should preserve properties of the input space  $\mathcal{X}$ , e.g. distances
- ***Why should we do dimensionality reduction?***
  - **Manifold assumption:** internal degrees of freedom are much smaller than # measured features  
⇒ data lies along a low-dimensional structure in feature space
  - **Visualization:** interpretation of data in high-d space is difficult, embedding data into 2d or 3d can provide insight
  - **Data compression:** compress data but retain most of information



# Dimensionality Reduction

- ▶ **Manifold assumption:**

internal degrees of freedom are much smaller than # measured features  
⇒ data lies along a low-dimensional structure in feature space



3	4	2	1	9	5	6	2	1	8
8	9	1	2	5	0	0	6	6	4
6	7	0	1	6	3	6	3	7	0
3	7	7	9	4	6	6	1	8	2
2	9	3	4	3	9	8	7	2	5
1	5	9	8	3	6	5	7	2	3
9	3	1	9	1	5	8	0	8	4
5	6	2	6	8	5	8	8	9	9
3	7	7	0	9	4	8	5	4	3
7	9	6	4	7	0	6	9	2	3

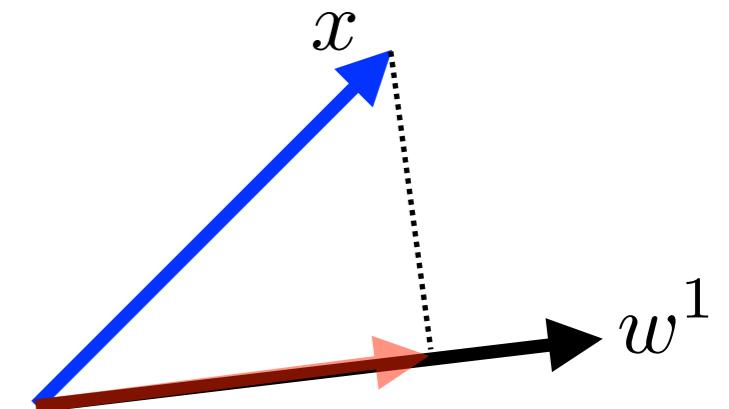
images of hand-written digits are assumed to be described by digit class and style, so we get 2 degree-of-freedom in latent space it is much smaller in comparison to number of features in data space (=number of pixels)

# Dimensionality Reduction

- **Supervised dimensionality reduction**
  - for datasets with desired class labels (e.g. datasets for classification)
  - Linear Discriminative Analysis (LDA)
- **Unsupervised dimensionality reduction**
  - for datasets with no labels (e.g. datasets for clustering)
  - Principal Components Analysis (PCA, also called Karhunen-Loeve-Transformation)
  - Kernel PCA
  - IsoMap
  - Locally Linear Embedding (LLE)
  - Laplacian Eigenmaps
  - Independent Component Analysis (ICA)
  - Stochastic Neighbor Embedding (SNE), t-Distributed SNE (t-SNE)
  - Large-scale Information Network Embedding (LINE) - leave to your reading

# Principal Components Analysis (PCA)

- Goal: we want to find a **orthogonal projection**  $W$  in which the data  $x$  after projection  $z = Wx$  will have **maximum variance**, i.e. **minimum mean square error (MSE)**  
**(come back to this later)**
- warm start: project on to a line!



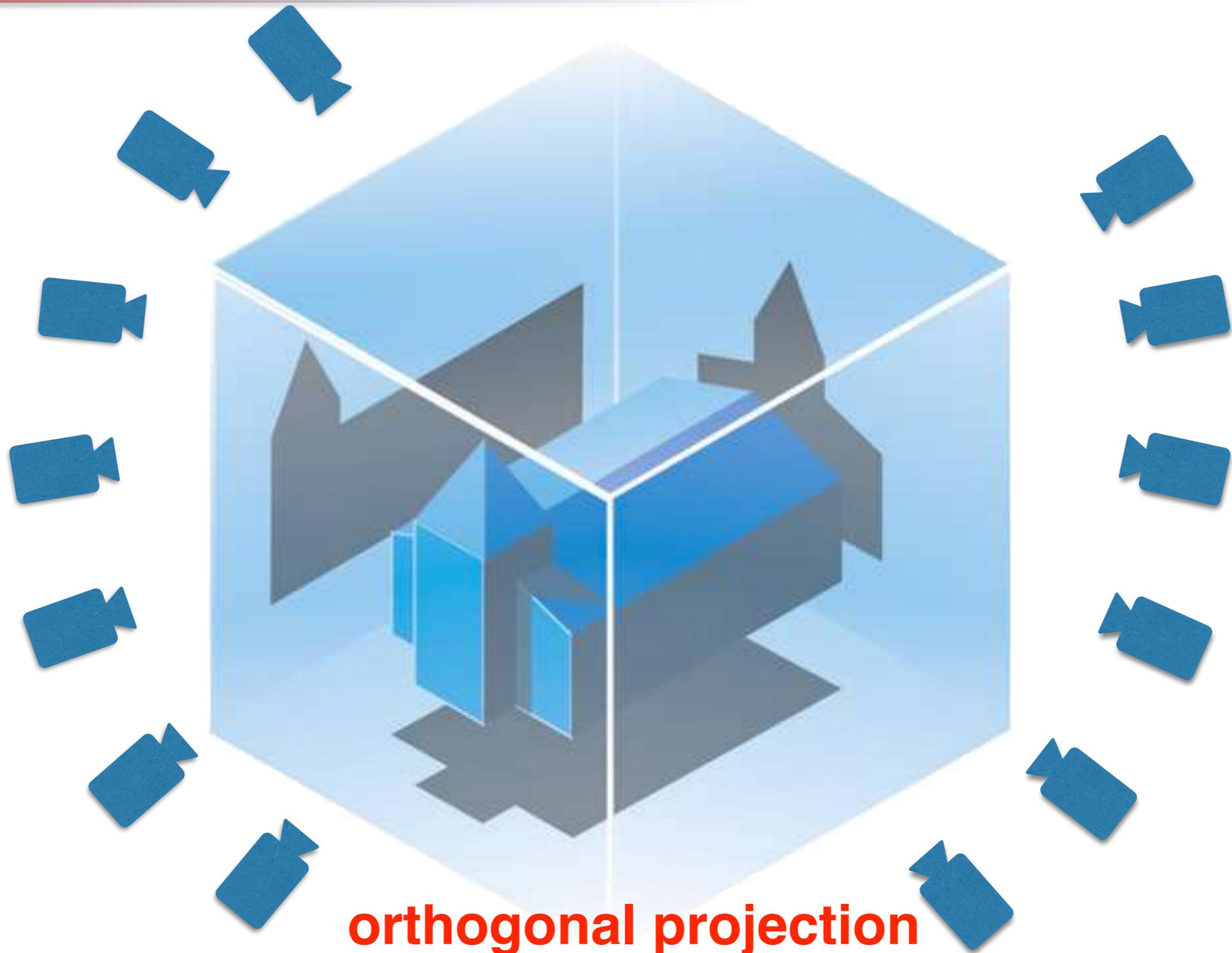
$$z_1 = w^1 \cdot x$$

$$\|w^1\| = 1$$

$$\xrightarrow{w^1} \operatorname{argmax}_{w^1} Var(z_1) = \frac{1}{N} \sum_{z_1} (z_1 - \bar{z}_1)^2, \text{ subject to } \|w^1\| = 1$$

$$\text{where } \bar{z}_1 = \frac{1}{N} \sum z_1 = \frac{1}{N} \sum w^1 \cdot x = w^1 \cdot \frac{1}{N} \sum x = w^1 \cdot \bar{x}$$

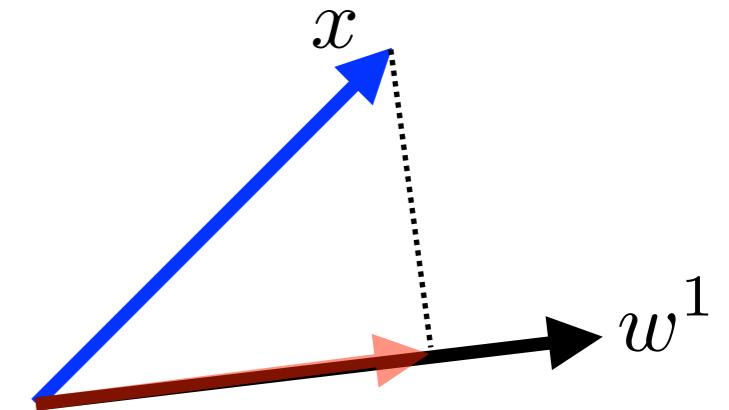
# Principal Components Analysis (PCA)



# Principal Components Analysis (PCA)

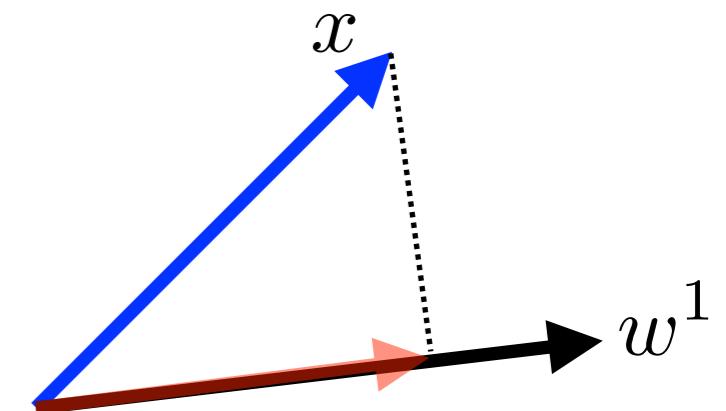
- Goal: we want to find a **orthogonal projection**  $W$  in which the data  $x$  after projection  $z = Wx$  will have **maximum variance**, i.e. **minimum mean square error (MSE)**

$$\begin{aligned}\operatorname{argmax}_w Var(z_1) &= \frac{1}{N} \sum_{z_1} (z_1 - \bar{z}_1)^2 \\ &= \frac{1}{N} \sum_x (w^1 \cdot x - w^1 \cdot \bar{x})^2 \\ &= \frac{1}{N} \sum_x (w^1 \cdot (x - \bar{x}))^2 \\ &= \frac{1}{N} \sum_x (w^1)^\top (x - \bar{x})(x - \bar{x})^\top (w^1) \\ &= (w^1)^\top \left[ \frac{1}{N} \sum_x (x - \bar{x})(x - \bar{x})^\top \right] (w^1)\end{aligned}$$



# Principal Components Analysis (PCA)

- Goal: we want to find a **orthogonal projection**  $W$  in which the data  $x$  after projection  $z = Wx$  will have **maximum variance**, i.e. **minimum mean square error (MSE)**
- warm start: project on to a line!



$$z_1 = w^1 \cdot x$$

$$\|w^1\| = 1$$

→  $\underset{w^1}{\operatorname{argmax}} (w^1)^\top S(w^1)$  , subject to  $\|w^1\| = 1$

remember?

Rayleigh quotient!

$S$  symmetric  
and PSD

$$\text{where } S = \left[ \frac{1}{N} \sum_x (x - \bar{x})(x - \bar{x})^\top \right]$$

optimal with  $w^1$  the eigenvector related to maximum eigenvalue!

# Principal Components Analysis (PCA)

- Goal: we want to find a **orthogonal projection**  $W$  in which the data  $x$  after projection  $z = Wx$  will have **maximum variance**, i.e. **minimum mean square error (MSE)**
- step further: project on to second line (orthogonal to first line)!

$$z_2 = w^2 \cdot x$$

$$\|w^2\| = 1$$

$$w^1 \cdot w^2 = 0$$

→  $\underset{w^2}{\operatorname{argmax}} (w^2)^\top S(w^2)$  , subject to  $\|w^2\| = 1$  and  $w^1 \cdot w^2 = 0$

**optimal with  $w^2$  the eigenvector related to 2nd largest eigenvalue!**

- **orthogonal projection**  $W$  is composed of  
 $k$  first largest eigenvectors of covariance matrix of  $x$   
principal components

# Principal Components Analysis (PCA)

- Goal: we want to find a **orthogonal projection**  $W$  in which the data  $x$  after projection  $z = Wx$  will have **maximum variance**, i.e. **minimum mean square error (MSE)**

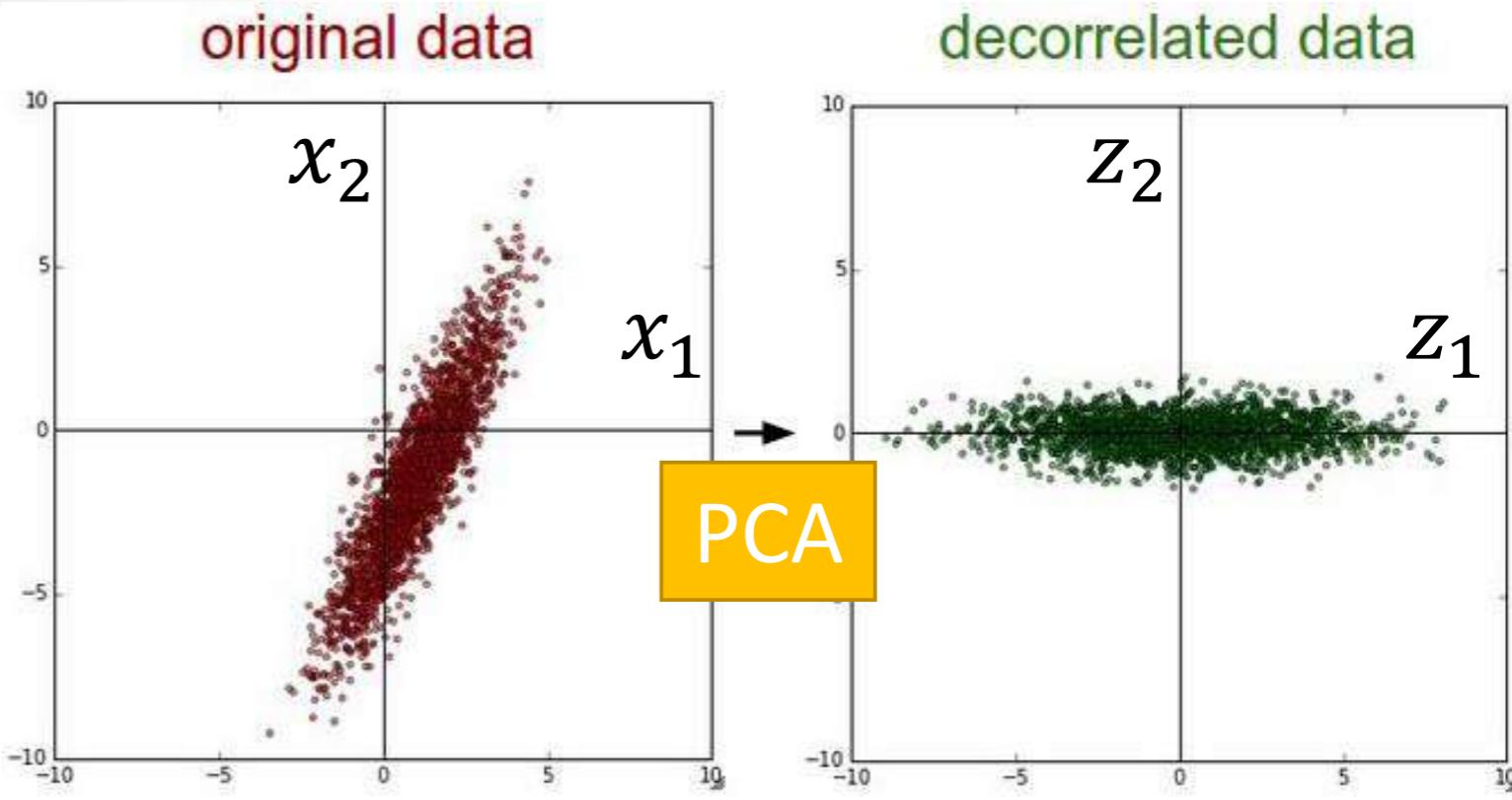
(note here the shape of variables  $W$  and  $x$  are changed  
to simplify the process of derivation)

**the dimensionality of principal components  
is the same as original data space**

$$\begin{aligned}\|x - \underbrace{xWW^\top}_{\substack{\text{coordinates} \\ \text{in the space} \\ \text{built upon} \\ \text{principal} \\ \text{components}}} z\|^2 &= \text{trace}( (x - xWW^\top)(x - xWW^\top)^\top ) \\ &= \text{trace}( (x - xWW^\top)(x^\top - WW^\top x^\top)^\top ) \\ &= \text{trace}(xx^\top) - 2 \cdot \text{trace}(xWW^\top x^\top) + \text{trace}(xWW^\top WW^\top x^\top) \\ &= \text{constant} - \text{trace}(xWW^\top x^\top) \\ &= \text{constant} - \text{trace}(W^\top x^\top x W) \\ &= \text{constant} - \text{constant} \cdot \underline{W^\top SW} \text{ maximize variance}\end{aligned}$$

$xWW^\top$  linear combination of principal components  
which tries to reconstruct original  $x$

# Principal Components Analysis (PCA)



$$z = Wx$$

$$Cov(z) = \frac{1}{N} \sum (z - \bar{z})(z - \bar{z})^\top = WSW^\top, \text{ where } S = Cov(x)$$

$$= WS [w^1 \cdots w^k]$$

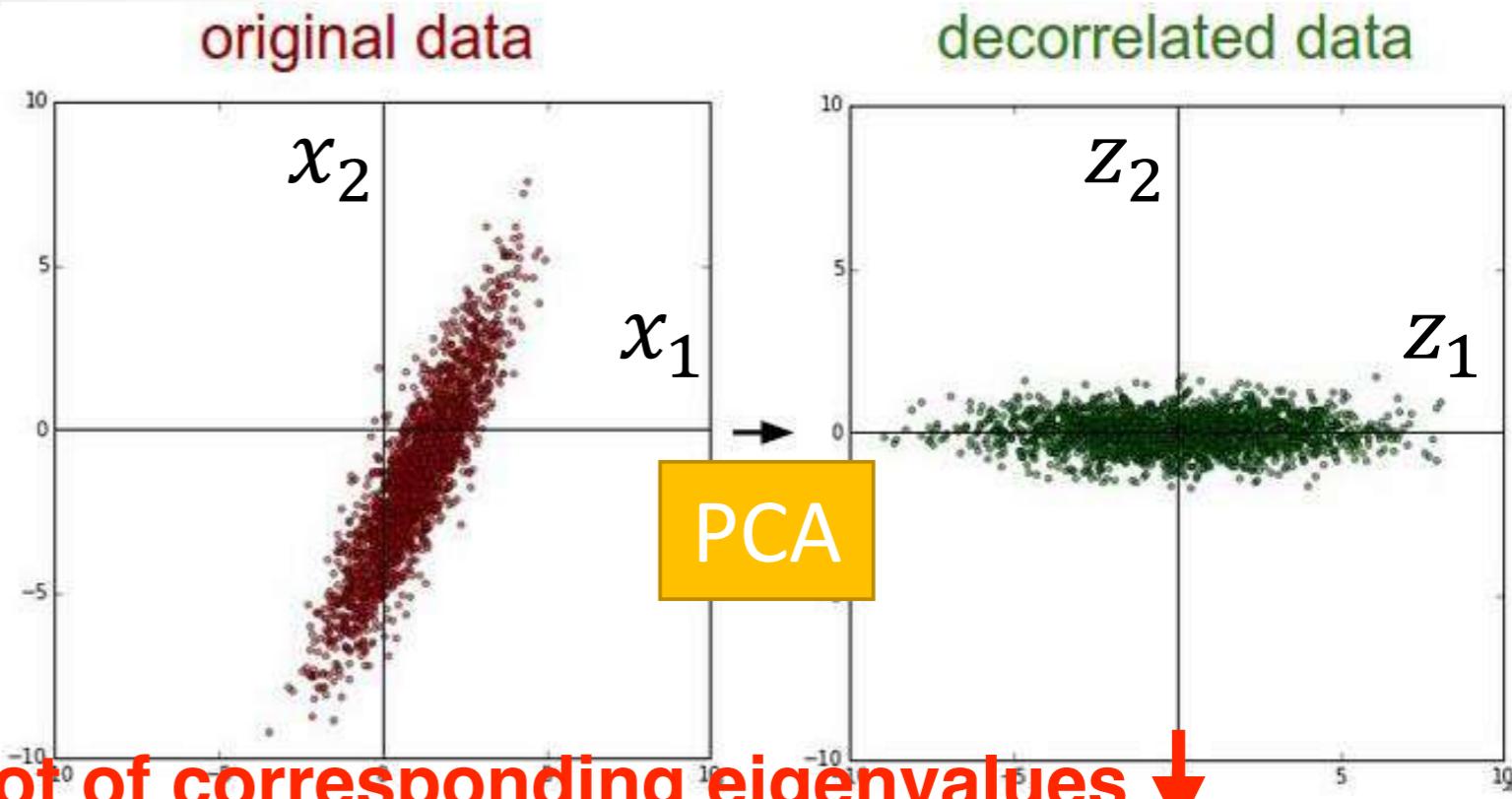
$$= W [Sw^1 \cdots Sw^k]$$

$$= W [\lambda_1 w^1 \cdots \lambda_k w^k]$$

$$= [\lambda_1 W w^1 \cdots \lambda_k W w^k]$$

$$= [\lambda_1 e_1 \cdots \lambda_k e_k] = D \text{ which is a diagonal matrix}$$

# Principal Components Analysis (PCA)

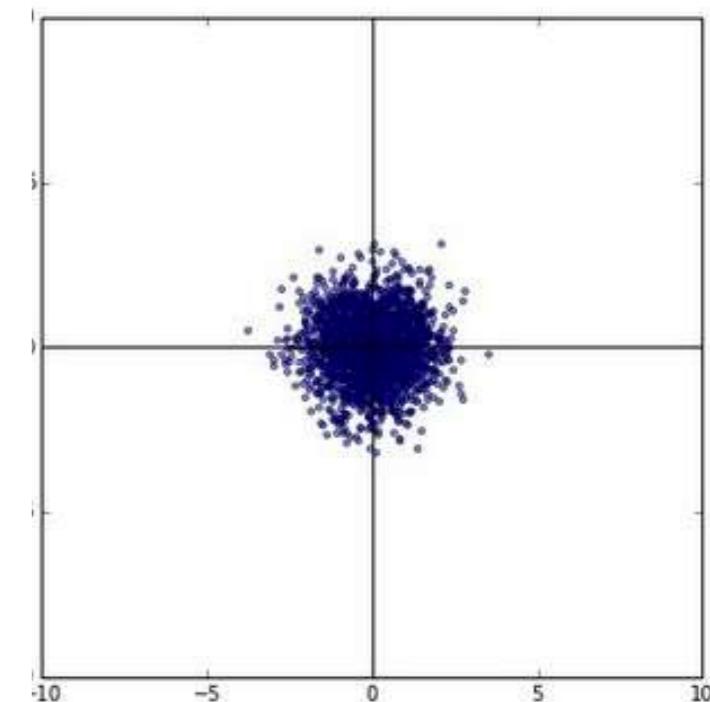


**rescale by the square-root of corresponding eigenvalues**

It is so-called “PCA whitening”, in practice:

1. pre-processing of data
  - resulting features are uncorrelated  
(efficient feature representation)

2. whitening “spheres” the data
  - eliminates differences in scaling  
(equal numerical range for every feature)



# Principal Components Analysis (PCA)

- PCA - two points of view:
  - the principal  $k$ -components span the  $k$ -dimensional subspace which yields the best approximation of the data (Euclidean norm),
  - the subspace spanned by the first  $k$  principal components contains “most” of the variance in the data.
  - *eigenvectors*: principal axes of maximum variance subspace.
  - *eigenvalues*: variance of projected inputs along principal axes
  - *estimated dimensionality*: number of significant eigenvalues
- PCA - a simple coordinate transformation
  - translation - mean of data points becomes new origin
  - rotation - change of the initial orthonormal basis into a new orthonormal basis which is defined by the data

Original faces



Recovered faces



eigenface

# Kernel PCA

- Non-linear extension of PCA
  - map (nonlinear) data into feature space and do PCA
  - we want to use kernel trick!
- **One key idea:** since  $Sw = \lambda w$  **eigenvalue problem for covariance matrix**
$$\begin{aligned}\rightarrow \underline{w} &= \frac{1}{\lambda} Sw \\ &= \frac{1}{\lambda} \left( \sum_i x_i x_i^\top \right) w \quad \text{assume centered already} \\ &= \frac{1}{\lambda} \left( \sum_i x_i^\top w \right) x_i \\ &= \sum_i \frac{x_i^\top w}{\lambda} x_i = \sum_i \underline{\alpha_i x_i}\end{aligned}$$

**all eigenvectors lie in the span of data points!**

# Kernel PCA

- Non-linear extension of PCA
  - map (nonlinear) data into feature space and do PCA
  - we want to use kernel trick!



- **eigenvalue problem of covariance matrix in feature space:**

$$\frac{1}{N} \Phi(X) \Phi(X)^\top w = \lambda w \quad \text{assume centered already}$$

with  $w = \sum_i \alpha_i \Phi(x_i) = \Phi(X)\alpha$

**all eigenvectors  
lie in the span of  
mapped data points!**

$$\rightarrow \Phi(X) \Phi(X)^\top \Phi(X) \alpha = \lambda N \Phi(X) \alpha$$

$$\Phi(X)^\top \Phi(X) \Phi(X)^\top \Phi(X) \alpha = \Phi(X)^\top \lambda N \Phi(X) \alpha$$

$$\Phi(X)^\top \Phi(X) \Phi(X)^\top \Phi(X) \alpha = \lambda N \Phi(X)^\top \Phi(X) \alpha$$

$$KK\alpha = \lambda NK\alpha$$

$$K\alpha = \lambda N\alpha$$

# Kernel PCA

- How to project data into principal components of kernel PCA?

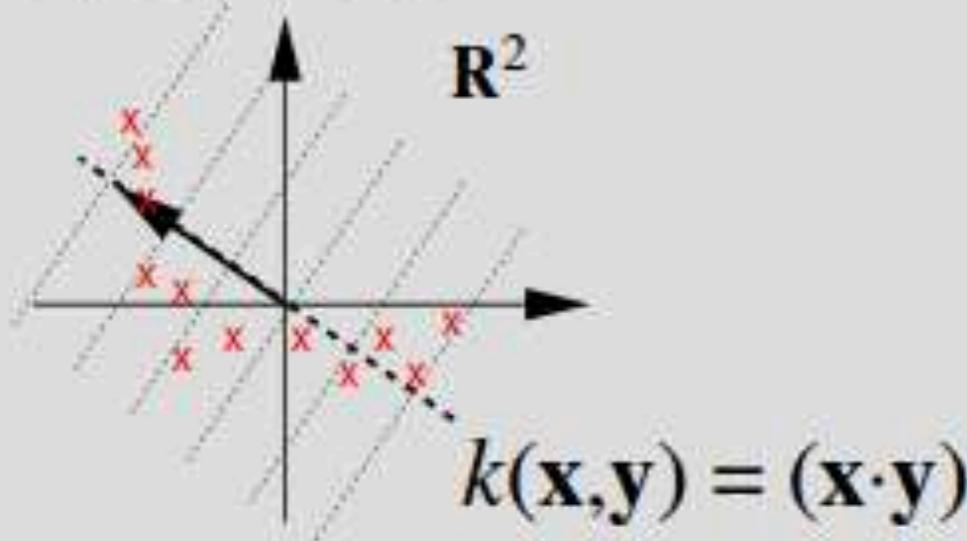
$$W\Phi(x_{\text{new}}) = \sum_i \alpha_i \Phi(x_i)\Phi(x_{\text{new}}) = \sum_i \alpha_i K(x_i, x_{\text{new}})$$

- We was assuming data in feature space are centered already

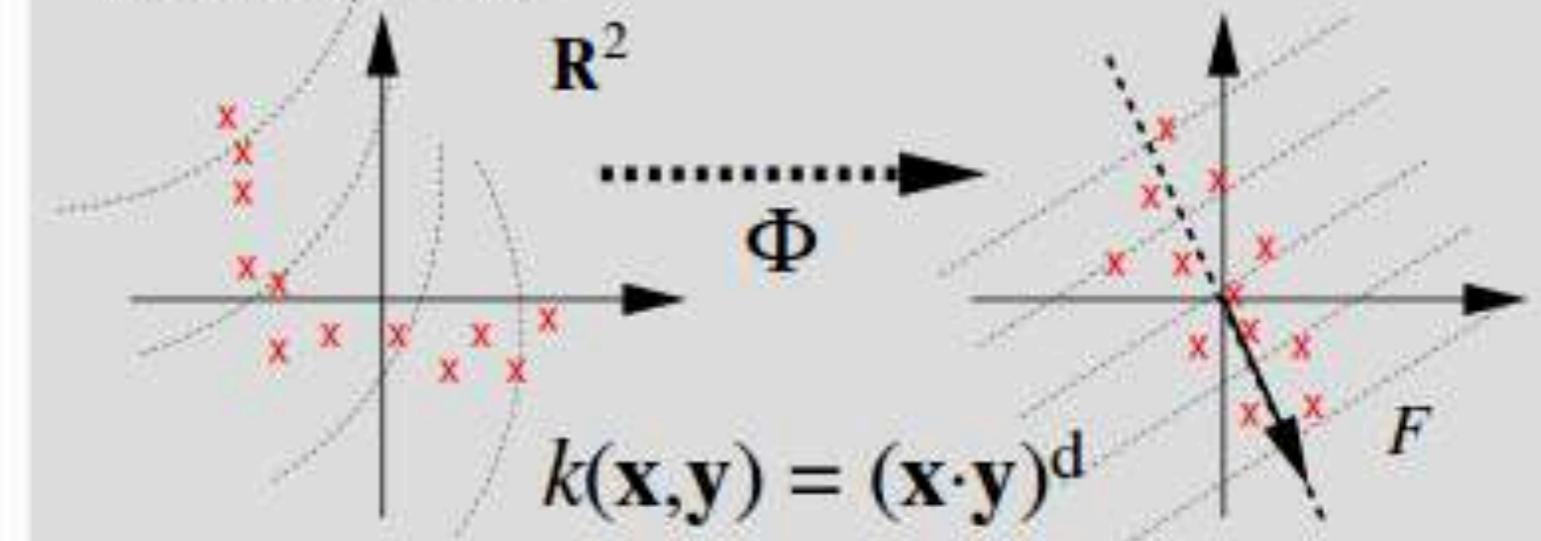
$$\begin{aligned} K_{ij}^C &= \langle \Phi_i^C \Phi_j^C \rangle \quad \text{where } \Phi_i = \Phi(x_i) \\ &= (\Phi_i - \frac{1}{N} \sum_k \Phi_k)^T (\Phi_j - \frac{1}{N} \sum_l \Phi_l) \\ &= \Phi_i^T \Phi_j - \frac{1}{N} \sum_l \Phi_i^T \Phi_l - \frac{1}{N} \sum_k \Phi_k^T \Phi_j + \frac{1}{N^2} \sum_k \sum_l \Phi_k^T \Phi_l \\ &= K_{ij} - \frac{1}{N} \sum_l K_{il} - \frac{1}{N} \sum_k K_{kj} + \frac{1}{N^2} \sum_k \sum_l K_{kl} \end{aligned}$$

$$\rightarrow K^C = K - \mathbf{1}_N K - K \mathbf{1}_N + \mathbf{1}_N K \mathbf{1}_N$$

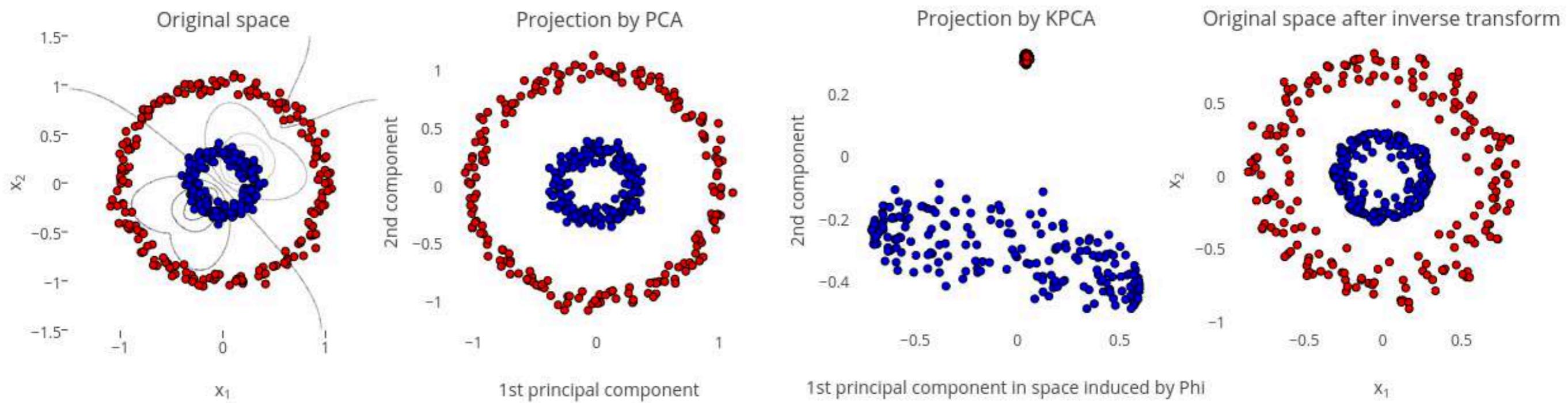
## linear PCA



## kernel PCA

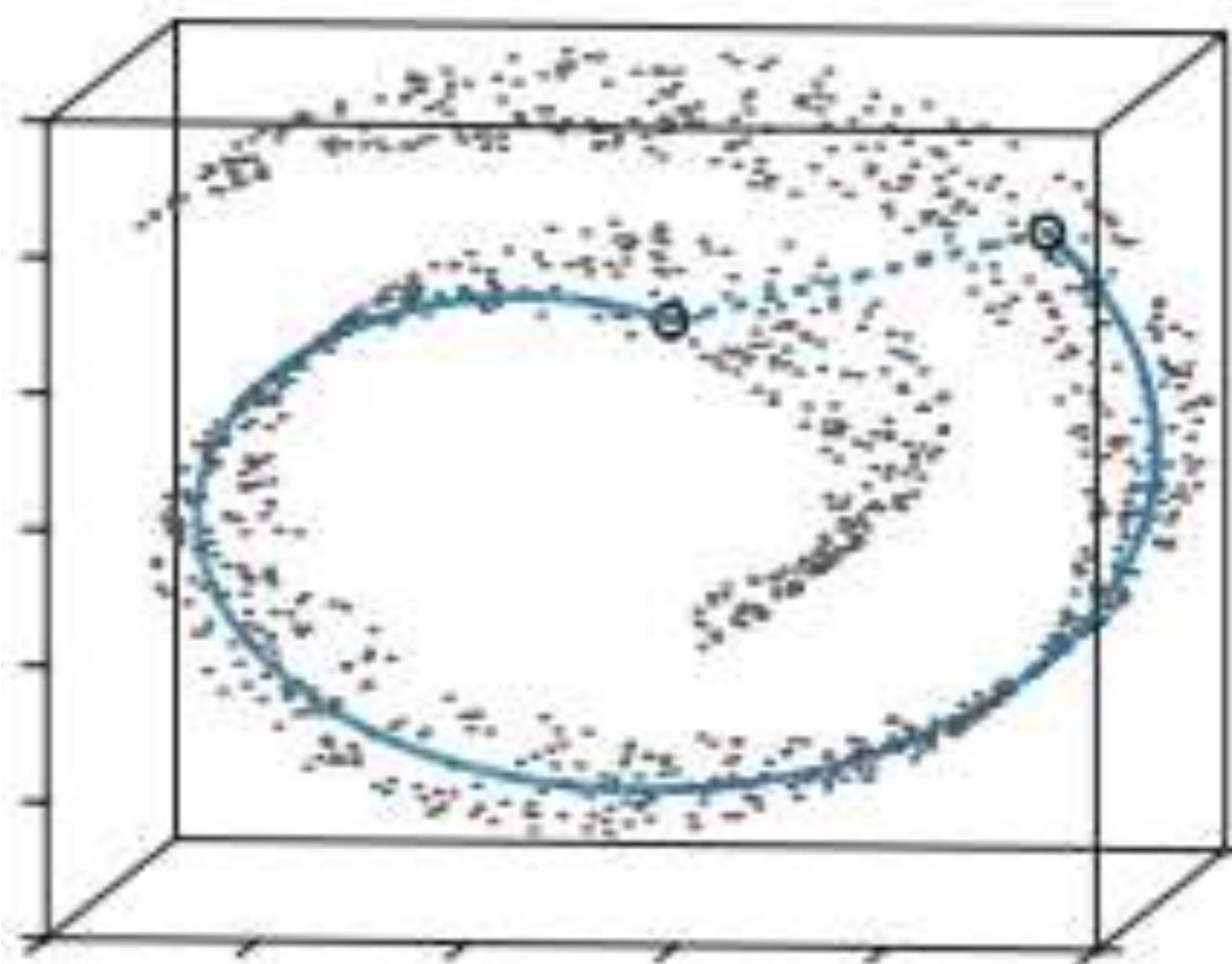


**Fig. 1.** Basic idea of kernel PCA: by using a nonlinear kernel function  $k$  instead of the standard dot product, we implicitly perform PCA in a possibly high-dimensional space  $F$  which is nonlinearly related to input space. The dotted lines are contour lines of constant feature value.



# IsoMap

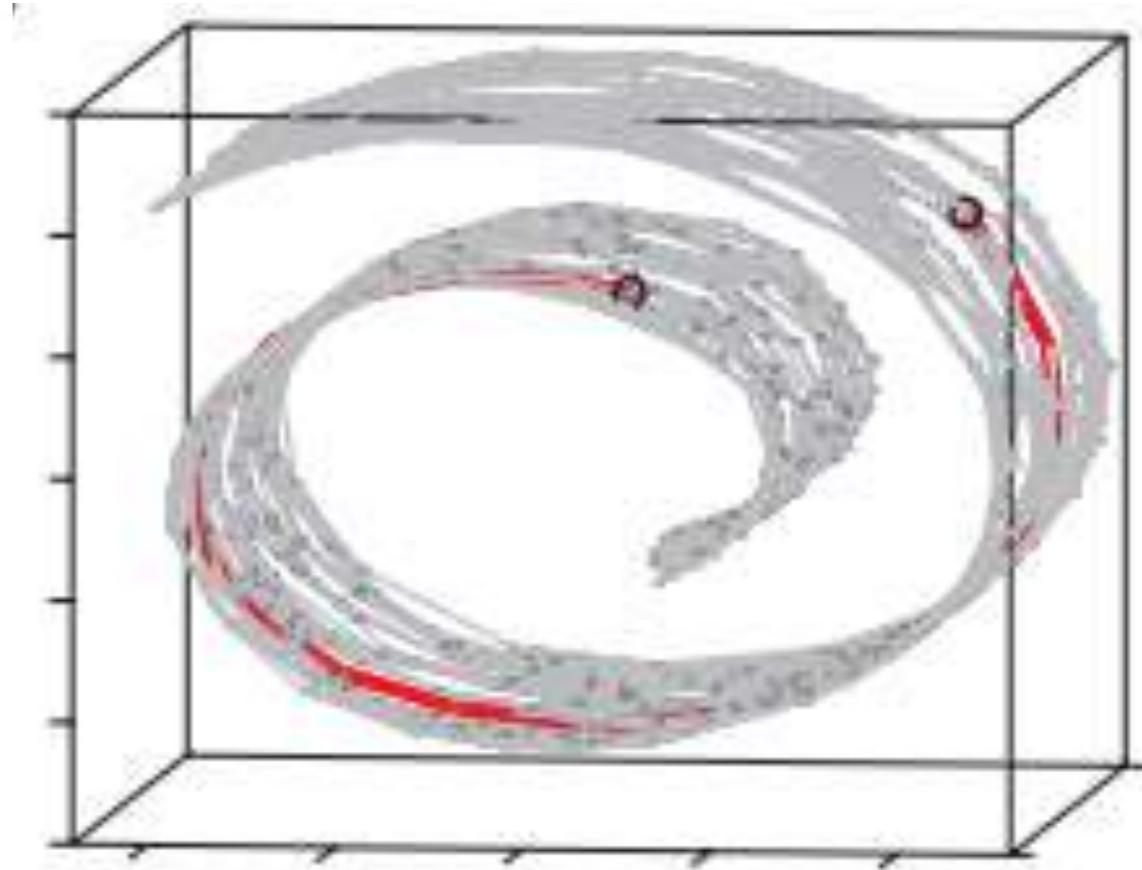
- IsoMap basic idea:
  - **preserve structure by geodesic distance** instead of euclidean distance



For two arbitrary points on a nonlinear manifold,  
their Euclidean distance in the high-dimensional input space  
may not accurately reflect their intrinsic similarity,  
as measured by geodesic distance along the low-dimensional manifold.

# IsoMap

- IsoMap basic idea:
  - Global Approach
  - On a low dimensional embedding
    - Nearby points should be nearby
    - Faraway points should be faraway
  - Estimate the geodesic distance between faraway points.
  - For neighboring points Euclidean distance is a good approximation to the geodesic distance.
  - For faraway points approximate the geodesic distance by the shortest path on a neighborhood graph  
**(Graph is discretized approximation of manifold)**



once we know the pairwise distance between all data points (in high-d)  
→ try to find a low dimensional embedding where the pairwise distance between embedded data points are kept as in original data space

# IsoMap

- Use classical **Multidimensional Scaling (MDS)** algorithm
  - assume that observed  $n \times n$  distance matrix  $\mathbf{D}$  is a matrix of Euclidean distances derived from a raw  $n \times q$  data matrix  $\mathbf{X}$ , which is not observed representation in low-d space
  - define a  $n \times n$  matrix:  $B = \mathbf{X}\mathbf{X}^\top$   
the element of  $\mathbf{B}$  are given by:  $b_{ij} = \sum_{k=1}^q x_{ik}x_{jk}$
  - the squared Euclidean distances between rows of  $\mathbf{X}$  in terms of  $\mathbf{B}$ :  $d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$
  - **Idea:** If the  $b_{ij}$  could be found in terms of the  $d_{ij}$  in the equation above, then we can derive  $\mathbf{X}$  from  $\mathbf{B}$  by factoring  $\mathbf{B}$ . (i.e.  $\mathbf{D} \rightarrow \mathbf{B} \rightarrow \mathbf{X}$ )
  - the elements of  $\mathbf{B}$  can be found from  $\mathbf{D}$  as

$$b_{ij} = -\frac{1}{2} [d_{ij}^2 - d_{i\cdot}^2 - d_{\cdot j}^2 + d_{..}^2]$$

$$\text{let } J = I - \frac{1}{n} \mathbf{1} \mathbf{1}^\top$$

$$\text{where } d_{i\cdot}^2 = \frac{\sum_{j=1}^n d_{ij}^2}{n}, d_{\cdot j}^2 = \frac{\sum_{i=1}^n d_{ij}^2}{n}, d_{..}^2 = \frac{\sum_{i=1}^n \sum_{j=1}^n d_{ij}^2}{n^2} \rightarrow B = -\frac{1}{2} J D^2 J$$

# IsoMap

- Use classical **Multidimensional Scaling (MDS)** algorithm
  - assume that observed  $n \times n$  distance matrix  $D$  is a matrix of Euclidean distances derived from a raw  $n \times q$  data matrix  $X$ , which is not observed representation in low-d space
  - define a  $n \times n$  matrix:  $B = XX^\top$   
the element of  $B$  are given by:  $b_{ij} = \sum_{k=1}^q x_{ik}x_{jk}$
  - the squared Euclidean distances between rows of  $X$  in terms of  $B$ :  $d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$
  - **Idea:** If the  $b_{ij}$  could be found in terms of the  $d_{ij}$  in the equation above, then we can derive  $X$  from  $B$  by factoring  $B$ . (i.e.  $D \rightarrow B \rightarrow X$ )

$B$  is symmetric, PSD with rank  $q \rightarrow B$  has  $q$  none negative and  $n-q$  zero eigenvalues.

Spectral decomposition:  $B = V\Lambda V^\top$

where  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_q, 0, \dots, 0)$

$V$  is composed of corresponding normalized eigenvectors

let  $J = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$

$\rightarrow B = -\frac{1}{2}JD^2J$

# IsoMap

- Use classical **Multidimensional Scaling (MDS)** algorithm
  - assume that observed  $n \times n$  distance matrix  $D$  is a matrix of Euclidean distances derived from a raw  $n \times q$  data matrix  $X$ , which is not observed representation in low-d space
  - define a  $n \times n$  matrix:  $B = XX^\top$   
the element of  $B$  are given by:  $b_{ij} = \sum_{k=1}^q x_{ik}x_{jk}$
  - the squared Euclidean distances between rows of  $X$  in terms of  $B$ :  $d_{ij}^2 = b_{ii} + b_{jj} - 2b_{ij}$
  - **Idea:** If the  $b_{ij}$  could be found in terms of the  $d_{ij}$  in the equation above, then we can derive  $X$  from  $B$  by factoring  $B$ . (i.e.  $D \rightarrow B \rightarrow X$ )

**$B$  is symmetric, PSD with rank  $q$ :**

$B$  can be chosen as  $B = V^* \Lambda^* V^{*\top}$   
where  $V^*$  contains the first  $q$  eigenvectors  
and  $\Lambda^*$  the first  $q$  eigenvalues

**mapped data in  
low-dim space!**

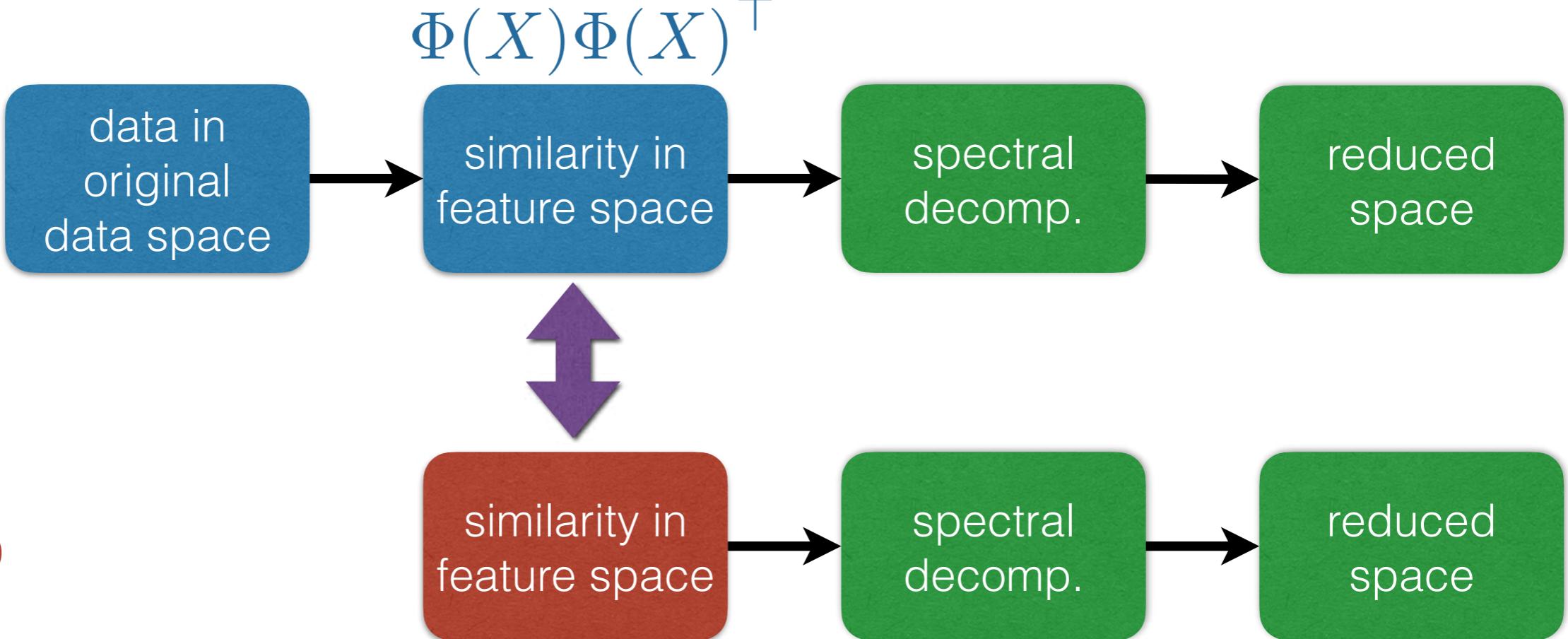
solution for  $X$  is:

$$X = V^* \Lambda^{*\frac{1}{2}}$$

choose your favorite low-d

# IsoMap

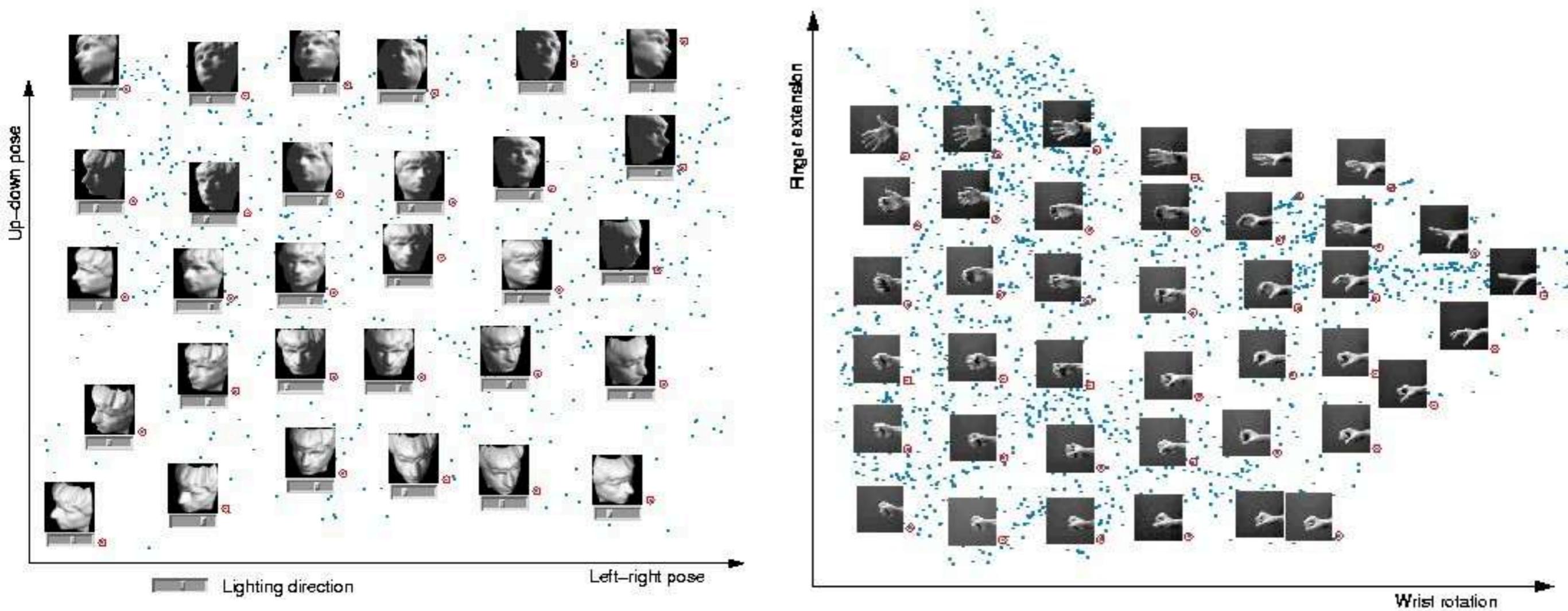
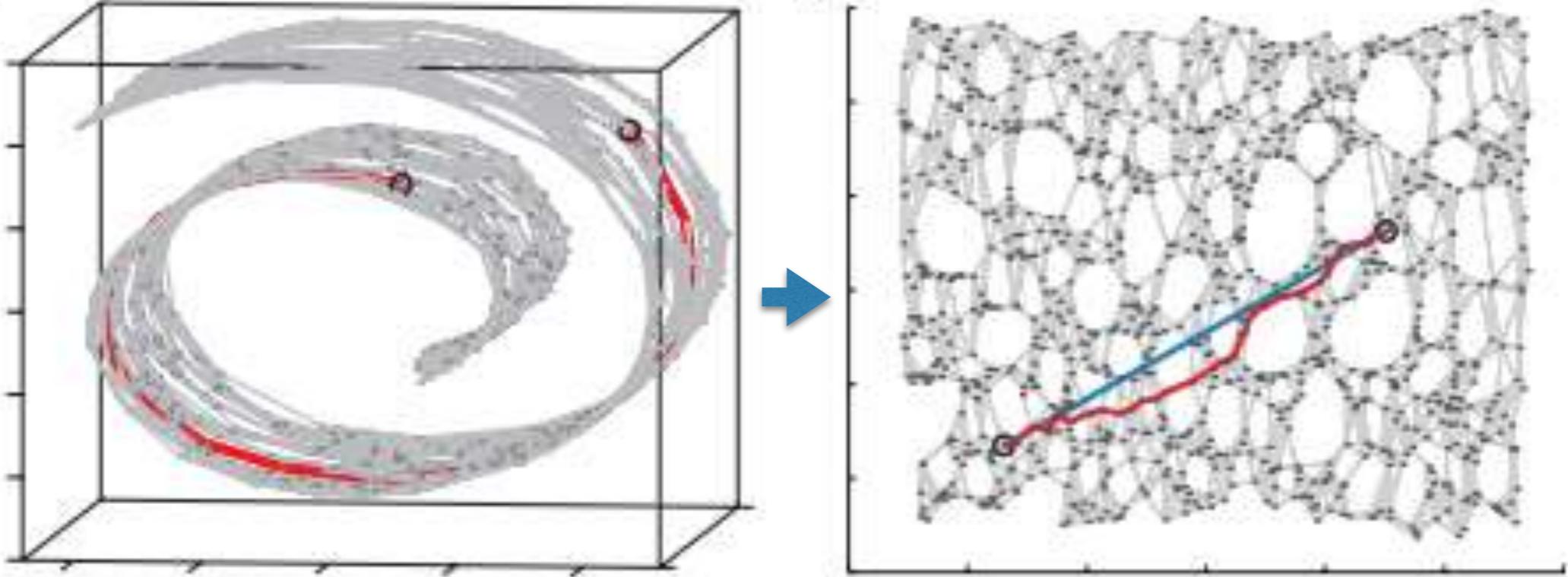
**kernel  
PCA**



$$B = -\frac{1}{2} J D^2 J$$

**$B$  shows similarity in feature space;  
where  $D$  shows geodesic distance**

# IsoMap

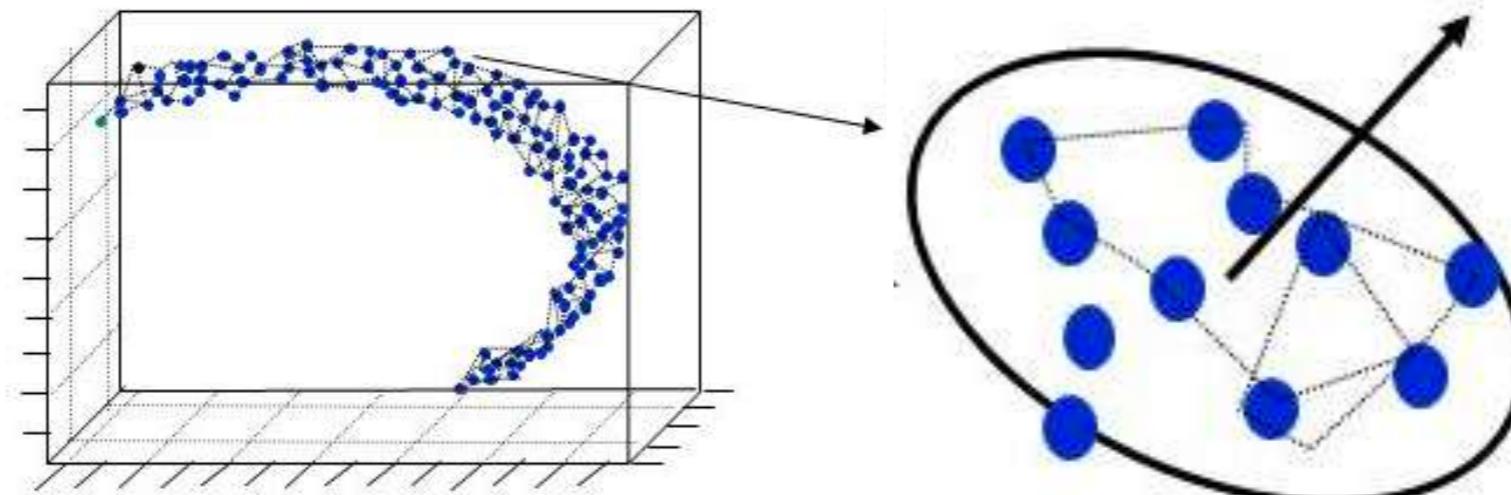


# IsoMap - Potential drawbacks

- May suffer from nonconvexity such as holes on manifolds
  - Basic assumption: given data is sampled sufficiently dense, graph shortest path distance will approximate closely the original geodesic distance as measured in manifold
  - Therefore could be problematic if assumption doesn't hold nicely
- Expensive computation
  - need to compute pairwise shortest paths between “all” data points
    - **Global**
    - **Non-sparse**
    - **MDS eigenvalue calculation on a full  $n \times n$  matrix**

# Locally Linear Embedding (LLE)

- **LLE** basic idea:
  - **Local** Approach
  - On a low dimensional embedding
    - **Nearby** points should be **nearby**
    - **Faraway** points should be **faraway**
  - **manifold** is a topological space which is **locally Euclidean**
  - simple assumption: suppose data are sampled from smooth underlying manifold. Provided there is sufficient data (such that the manifold is well-sampled), we expect **each data point and its neighbors to lie on or close to a locally linear patch of the submanifold.**

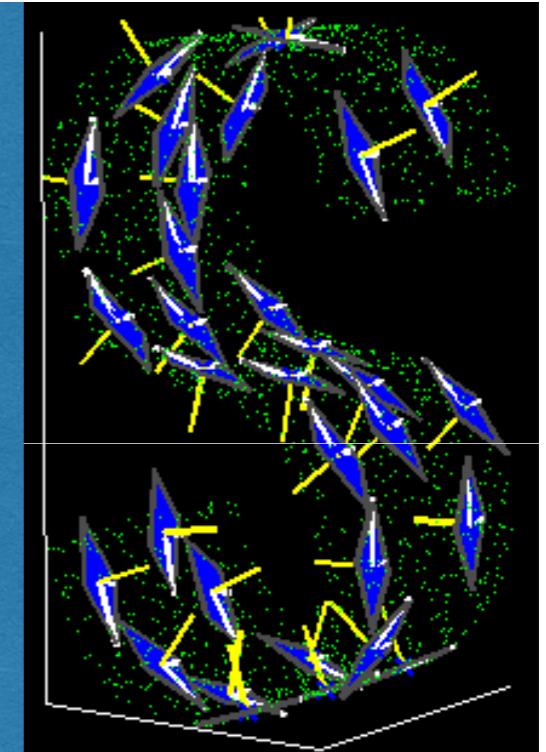


# Locally Linear Embedding (LLE)

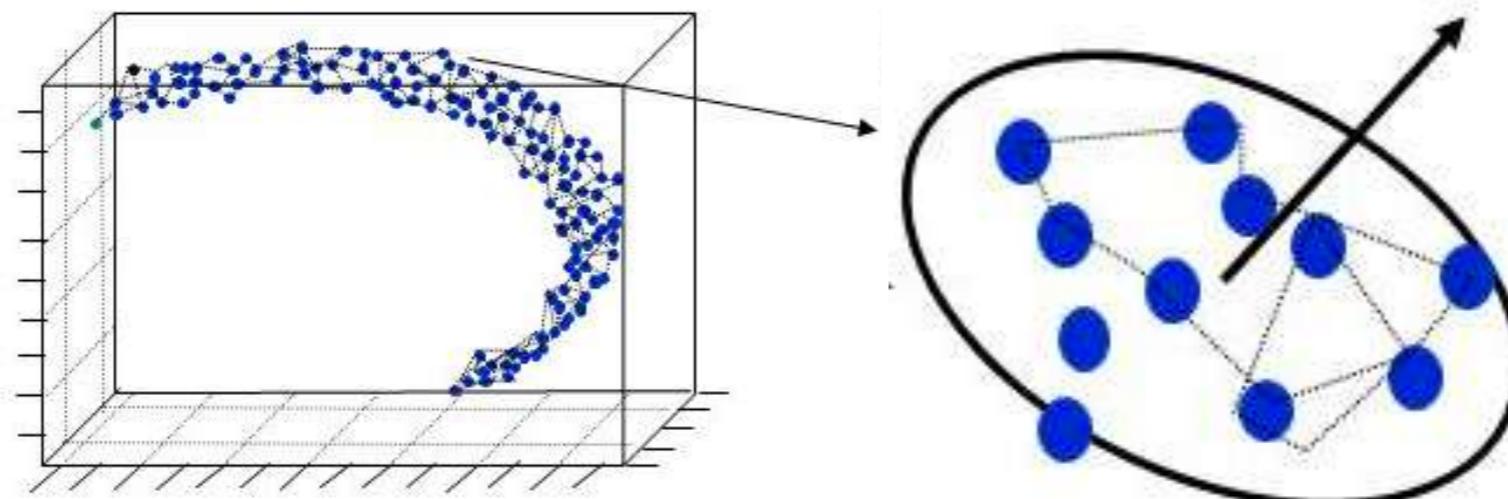
- 

Idea:

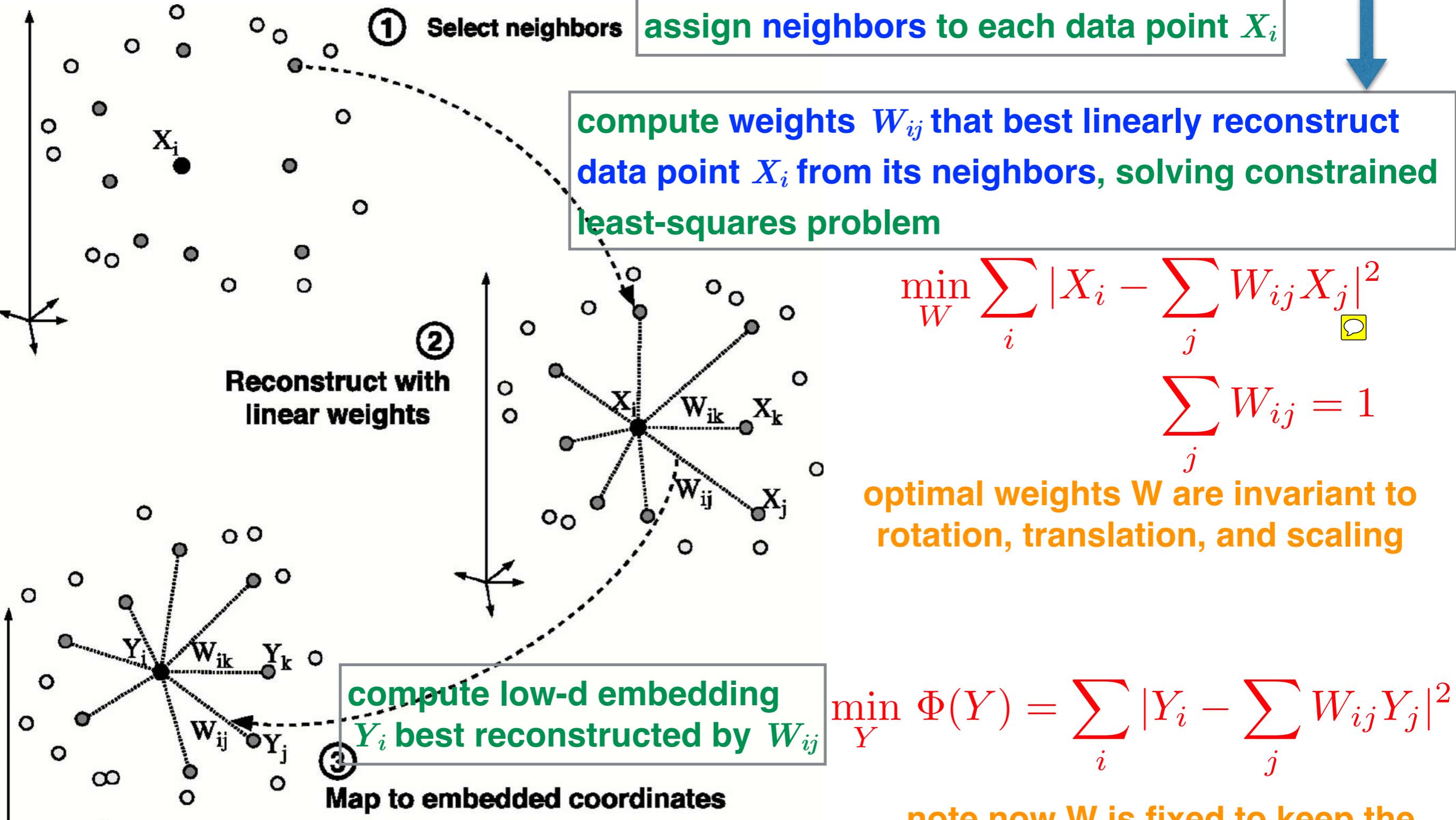
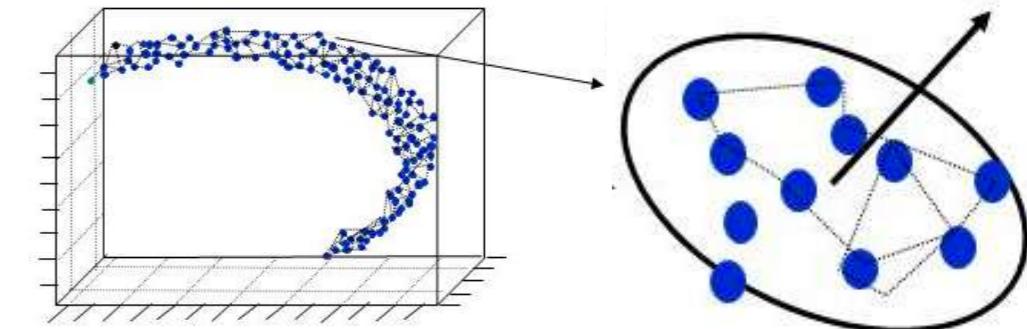
1. approximate data by set of linear patches
2. glue these patches together on a low-d subspace, such that neighborhood relationship between patches are preserved



- simple assumption: suppose data are sampled from smooth underlying manifold. Provided there is sufficient data (such that the manifold is well-sampled), we expect **each data point and its neighbors to lie on or close to a locally linear patch of the submanifold.**



# Locally Linear Embedding (LLE)



$$\min_Y \Phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2$$

note now  $W$  is fixed to keep the local structure after embedding

# Locally Linear Embedding (LLE)

- Look into equations...

$$\min_W \sum_i |X_i - \sum_j W_{ij} X_j|^2 \quad \text{and} \quad \sum_j W_{ij} = 1$$

$$\begin{aligned} |X_i - \sum_j W_{ij} X_j|^2 &= |\sum_j W_{ij}(X_i - X_j)|^2 \quad (\text{due to } \sum_j W_{ij} = 1) \\ &= \sum_{jk} W_{ij} W_{ik} C_{jk}, \text{ where } C_{jk} = (X_i - X_j)(X_i - X_k) \end{aligned}$$

(we can use Lagrange with constraint  $\sum_j W_{ij} = 1$  to solve)

$$\Rightarrow W_{ij} = \frac{\sum_j C_{ij}^{-1}}{\sum_{lm} C_{lm}^{-1}}$$

# Locally Linear Embedding (LLE)

- Look into equations...

$$\min_Y \Phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2$$

$$\Phi(Y) = \sum_{ij} M_{ij} (Y_i \cdot Y_j)$$

$$\text{where } M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki} W_{kj}$$

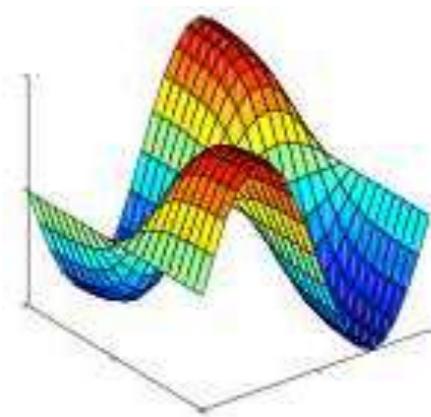
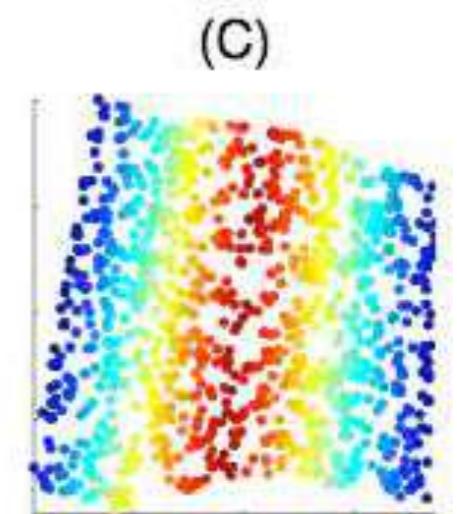
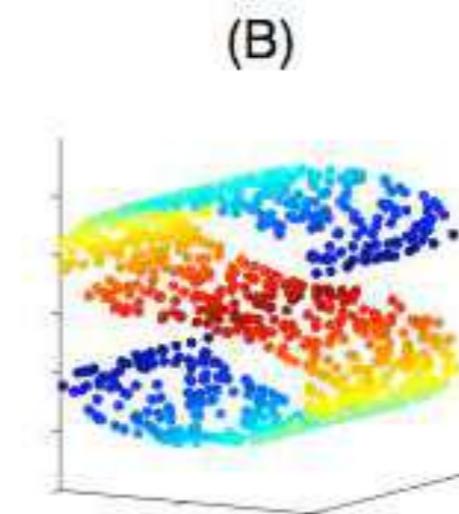
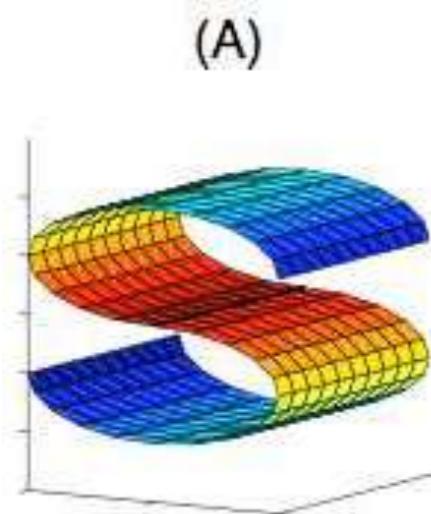
$$\delta_{ij} = 1 \text{ if } i = j \text{ and 0 otherwise}$$

$$\text{which can be shown that } M = (I - W)^\top (I - W)$$

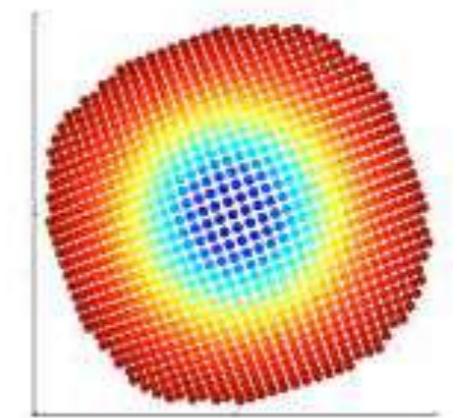
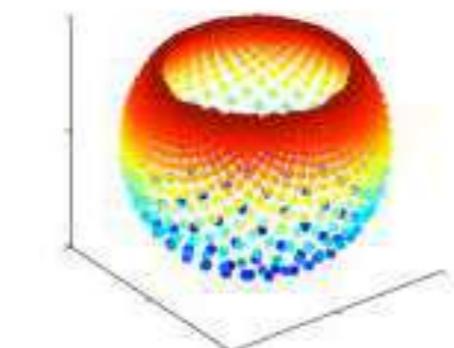
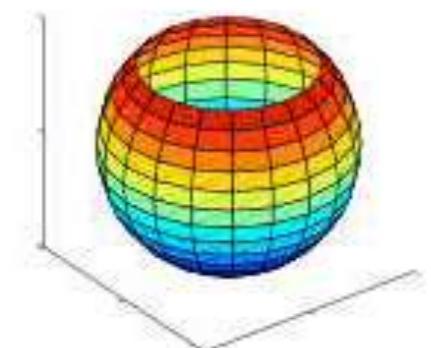
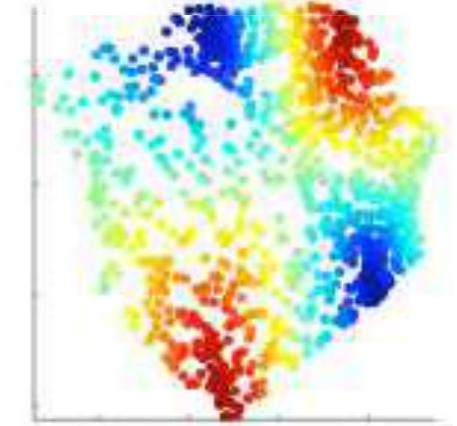
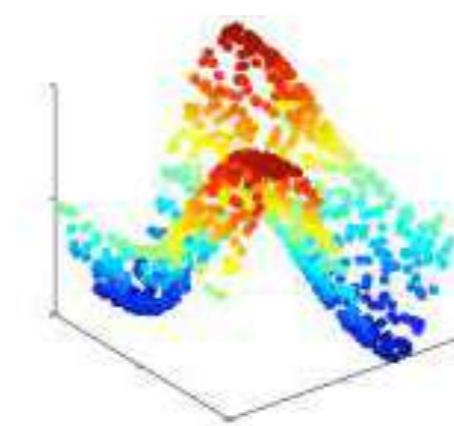
**since  $Y$  can be translated by a constant displacement without affecting total cost, set be centered on origin:**  $\sum_i Y_i = 0$   
**to avoid degenerate solutions, constrain  $Y$  vectors to have unit covariance, with outer products satisfying:**  $\frac{1}{N} \sum_i Y_i Y_i^\top = I$

**by Rayleitz-Ritz theorem, optimal given by bottom  $d+1$  eigenvectors of  $M$  discard bottom eigenvector  $[1, 1, \dots, 1]$  with eigenvalue 0 (due to  $\sum Y = 0$ )**

# Locally Linear Embedding (LLE)



use  
**N=1000 points and  
k=8 nearest neighbors**

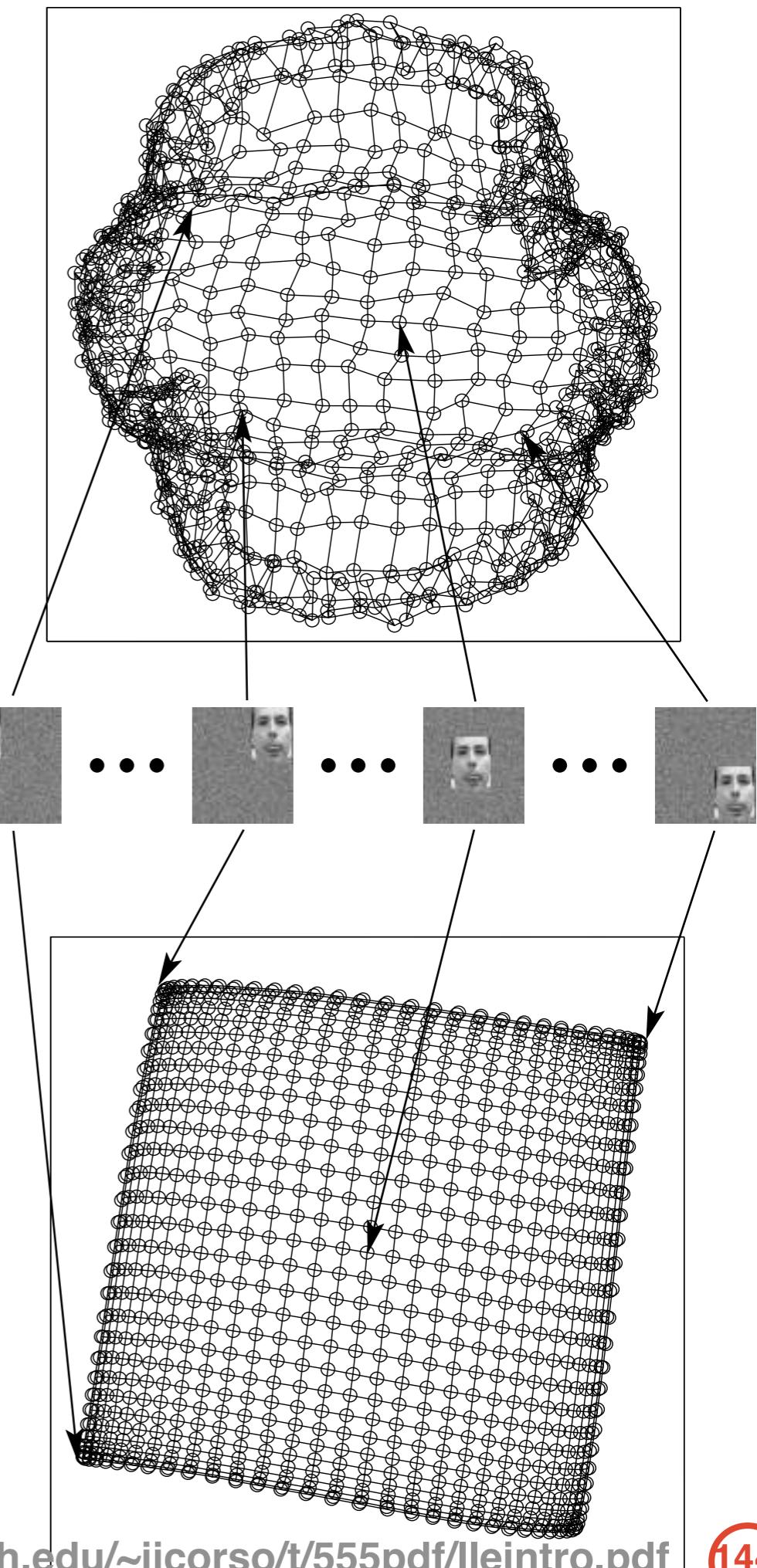


# Locally Linear Embedding (LLE)

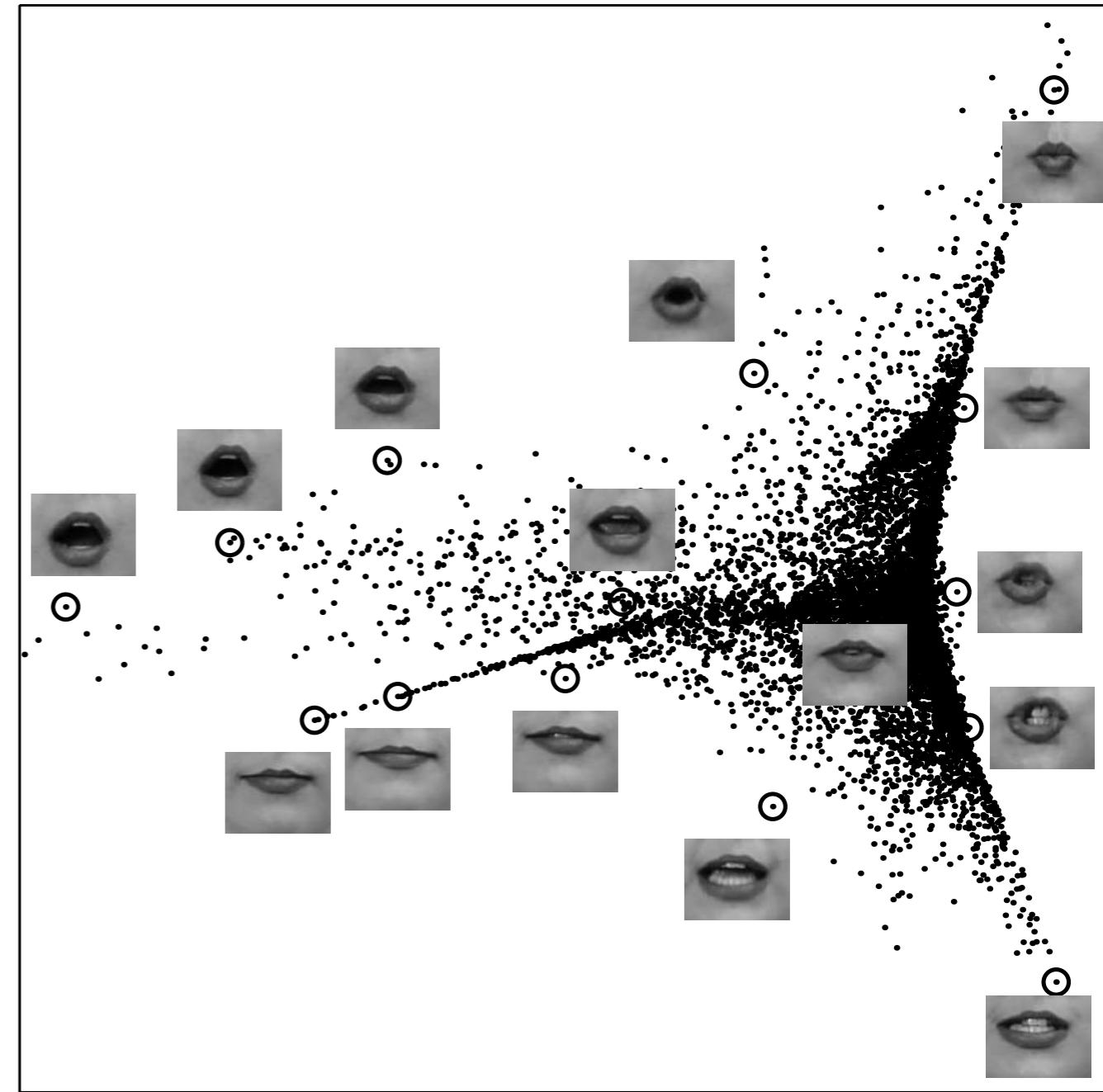
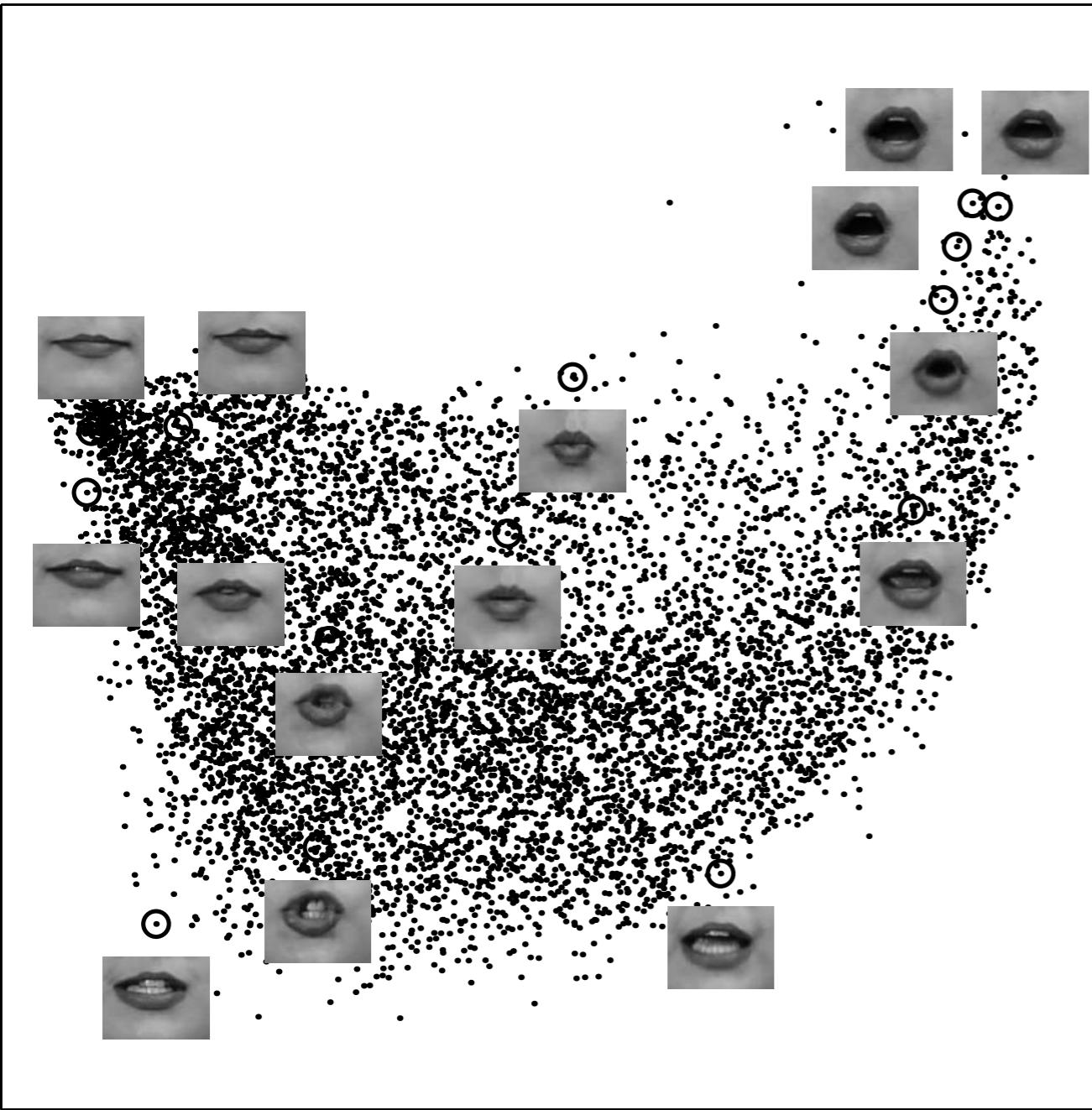


The results of PCA (top) and LLE (bottom), applied to images of a single face translated across a two-d background of noise.

Note how LLE maps the images with corner faces to the corners of its two dimensional embedding, while PCA fails to preserve the neighborhood structure of nearby images.



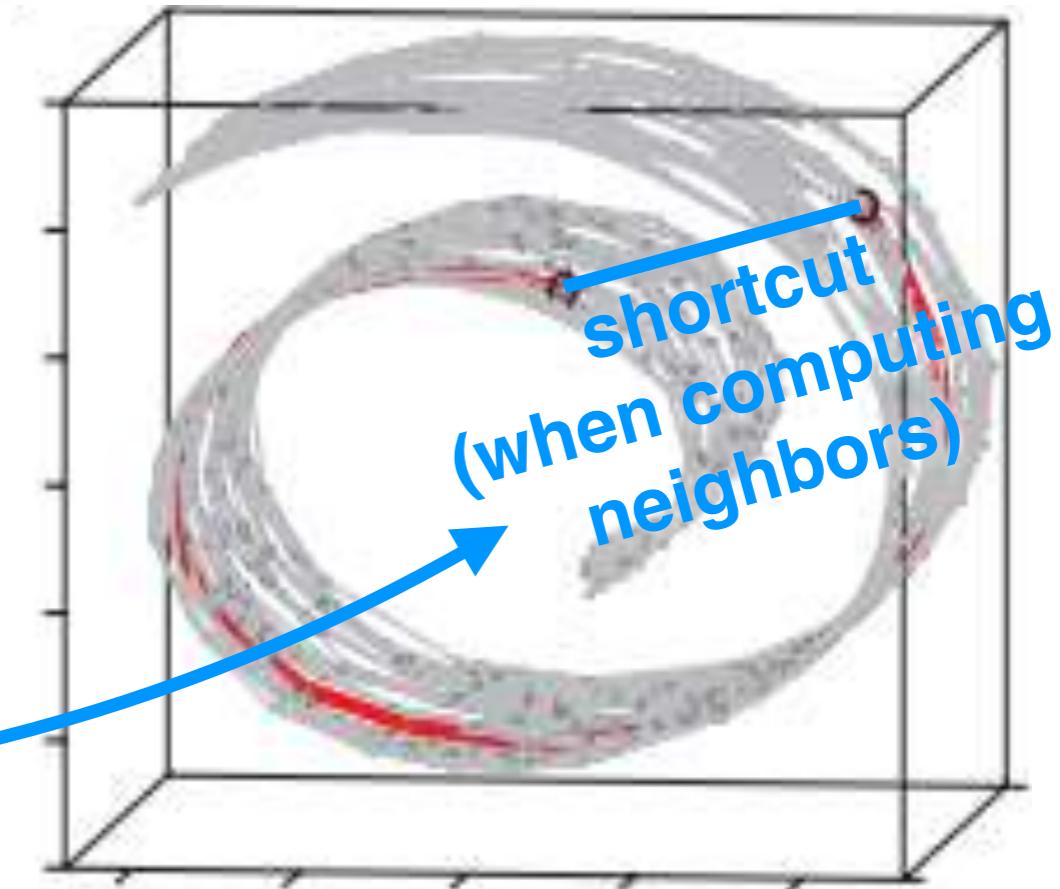
# Locally Linear Embedding (LLE)



Images of lips described by first two coordinates of PCA (left) & LLE (right). Representative lips are shown next to circled points in different parts of each space. LLE keeps nonlinear structure in the data.

# Locally Linear Embedding (LLE) versus IsoMap

- Many similarities
  - Graph-based
  - Spectral method
  - No local minima
  - Nonparametric  
(only heuristic is **neighborhood**)
  - Sensitive to “**shortcuts**”
- Essential Differences
  - **LLE** has **sparse** matrix (**local**), **IsoMap** has **dense** matrix (**global**)
  - **LLE** preserves **weights**, **IsoMap** preserves **distances**
  - LLE is much faster:)



# Laplacian Eigenmaps

- Map nearby inputs to nearby outputs, where the nearness is encoded by graph
- Summary of Laplacian Eignemaps algorithm:
  - identify **k-nearest neighbors** (as in LLE, we care **local** structure!)
  - Assign weights to neighbors to build a similarity graph  $\mathbf{W}$
  - Compute the graph Laplacian  $\mathbf{L}$  based on similarity graph  $\mathbf{W}$
  - Solve eigenvalue problem of the graph Laplacian  $\mathbf{L}$ , optimal embedding is given by bottom  $d+1$  eigenvectors of  $\mathbf{L}$ , discard bottom eigenvector  $[1, 1, \dots, 1]$  with eigenvalue 0

**Yes, exactly what we have learnt from spectral clustering!**

**eigenvectors:** functions from nodes to  $\mathbb{R}$  in a way that  
“close by” points are assigned “close by” values

**eigenvalues:** measure *smoothness* →  
how close are the values of neighboring points

# Laplacian Eigenmaps

- Map nearby inputs to nearby outputs, where the nearness is encoded by graph

1. Weighted graph is discretized representation of manifold
2. Laplacian measure smoothness of functions over manifold and graph

$$\text{manifold: } \int_{\Omega} |\nabla f|^2 d\omega = \int f \Delta f d\omega$$

$$\text{graph: } \sum_{ij} W_{ij} (f_i - f_j)^2 = f^\top L f$$

**Yes, exactly what we have learnt from spectral clustering!**

**eigenvectors:** functions from nodes to  $\mathbb{R}$  in a way that “close by” points are assigned “close by” values

**eigenvalues:** measure *smoothness* → how close are the values of neighboring points

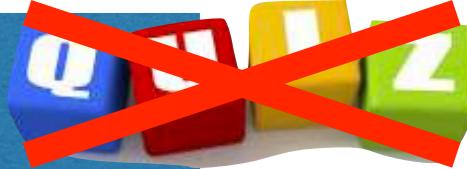


## Analysis on Manifolds

- Consider Riemannian manifold  $\Omega \in \mathbb{R}^D$ 
  - a real differentiable manifold in which tangent space is equipped with dot product
- Laplace Beltrami operator
  - $\Omega$  has a “natural” operator  $\Delta$  on differentiable functions
  - $\Delta$  is a second order differential operator defined as “divergence of the gradient”
$$\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$$
  - assume  $\mathcal{L}^2(\Omega)$  is space of all square integrable functions on  $\Omega$
  - $\Delta$  is a self-adjoint positive semi-definite operator and its eigenfunctions form the basis
  - Thus all  $f$  in  $\mathcal{L}^2(\Omega)$  can be rewritten as  $f(x) = \sum_i \alpha_i e_i(x)$

## Smoothness Functional

- Defined as  $S(f) = \int_{\Omega} |\nabla f|^2 d\omega = \int f \Delta f d\omega = \langle \Delta f, f \rangle_{\mathcal{L}^2(\Omega)}$ 
  - value close to zero implies  $f$  being smooth
  - since  $S(e_i) = \langle \Delta e_i, e_i \rangle = \lambda_i$
  - we have  $S(f) = \langle \Delta f, f \rangle = \left\langle \sum_i \alpha_i \Delta e_i, \sum_i \alpha_i e_i \right\rangle = \sum_i \lambda_i \alpha_i$



# Analysis on Manifolds

- Consider Riemannian manifold  $\Omega \in \mathbb{R}^D$ 
  - a real differentiable manifold in which tangent space is equipped with dot product
- Laplace Beltrami operator
  - $\Omega$  has a “natural” operator  $\Delta$  on differentiable functions
  - $\Delta$  is a second order differential operator defined as “divergence of the gradient”
$$\Delta = \sum_i \frac{\partial^2}{\partial x_i^2}$$
  - assume  $\mathcal{L}^2(\Omega)$  is space of all square integrable functions on  $\Omega$
  - $\Delta$  is a self-adjoint positive semi-definite operator and

**nice reference:**

wiki of course: [https://en.wikipedia.org/wiki/Discrete\\_Laplace\\_operator](https://en.wikipedia.org/wiki/Discrete_Laplace_operator)

Wardetzky et al., Discrete Laplace operators: No free lunch

<https://graphics.stanford.edu/courses/cs468-13-spring/assets/lecture12-lu.pdf>

<http://brickisland.net/cs177/?p=248> and <http://brickisland.net/cs177/?p=309>

S  
.

$$\mathcal{S}(f) = \|f - \nabla f + u\omega\| = \|f - f \Delta f + u\omega\| = \langle \Delta f, f \rangle_{\mathcal{L}^2(\Omega)}$$

choosing the lowest  $d$  eigenfunctions provides a maximally smooth approximation to the manifold

- we have  $S(f) = \langle \Delta f, f \rangle = \left\langle \sum_i \alpha_i \Delta e_i, \sum_i \alpha_i e_i \right\rangle = \sum_i \lambda_i \alpha_i$

# Laplacian Eigenmaps versus LLE

- More similar than different
  - Graph-based
  - Spectral method
  - Sparse eigenvalue problem
  - Similar results in practice
- Essential differences
  - **Laplacian eigenmaps** preserves **locality**, **LLE** preserves **local linearity**
  - Laplacian eigenmaps uses **graph Laplacian**
  - **Relation to Kernel-PCA:**  
*one can see Laplacian eigenmaps as Kernel-PCA with a special data-dependent kernel (pseudo-inverse of the graph Laplacian)*
  -

# Independent Component Analysis (ICA)

- **Motivation:** cocktail party problem - blind source separation

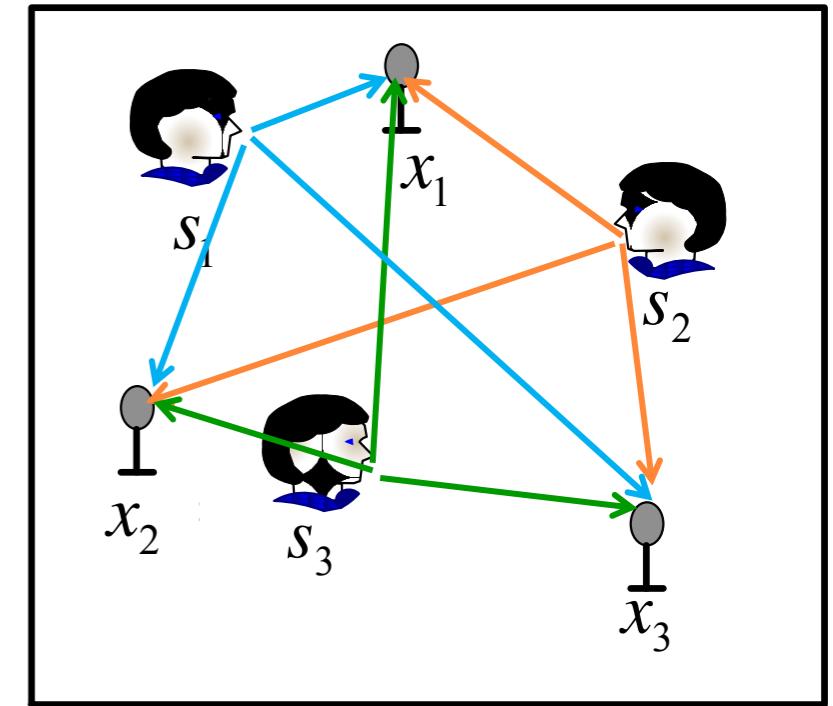
- $k$  different speakers (sources)

$$s_1(t), s_2(t), \dots, s_k(t)$$

- $d$  microphones (sensors)

$$x_1(t), x_2(t), \dots, x_d(t)$$

- **Assumption:** measured signal is linear superposition of sources



$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t)$$

$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t)$$

$$x_3(t) = a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t)$$

- **Goal:** having only the signal of the microphones, find the sources

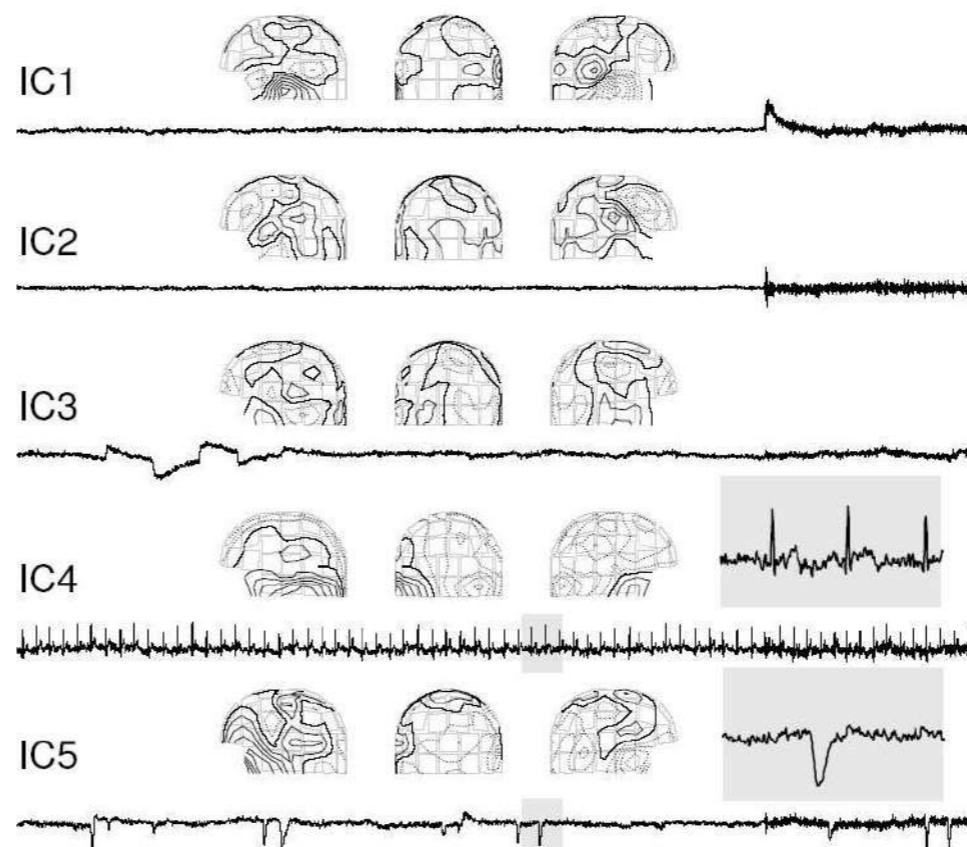
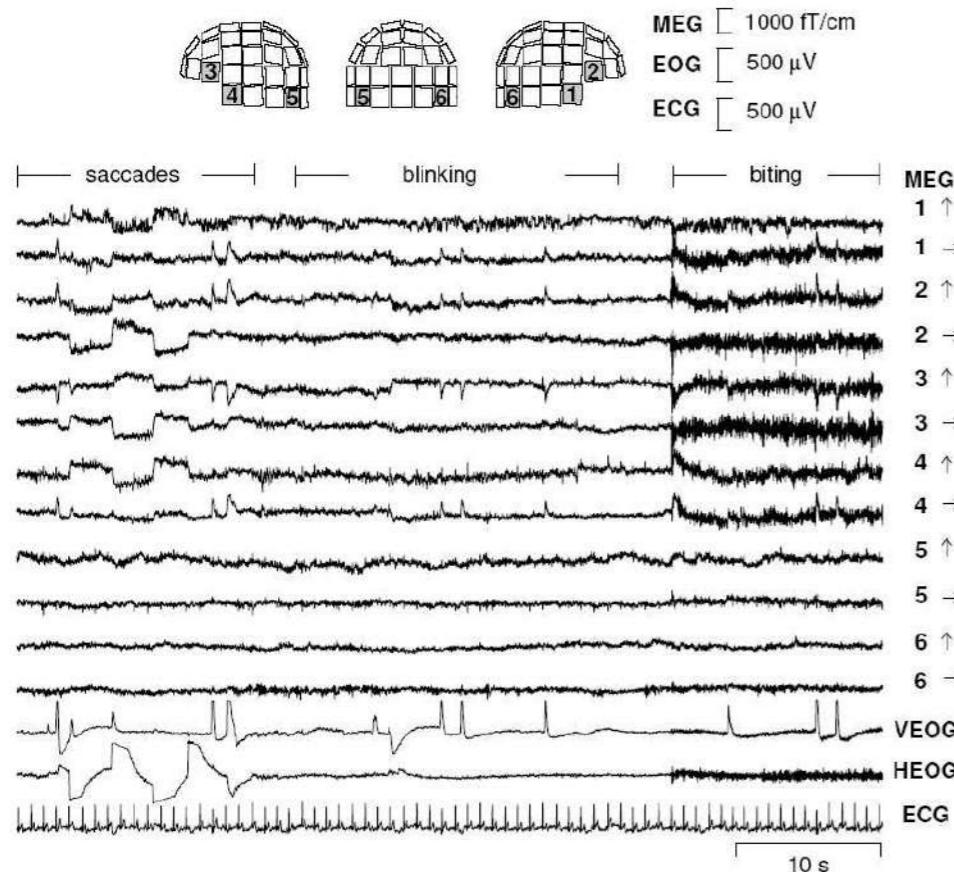
- determine  $A$ , where:

$$x(t) = \underline{A} s(t)$$

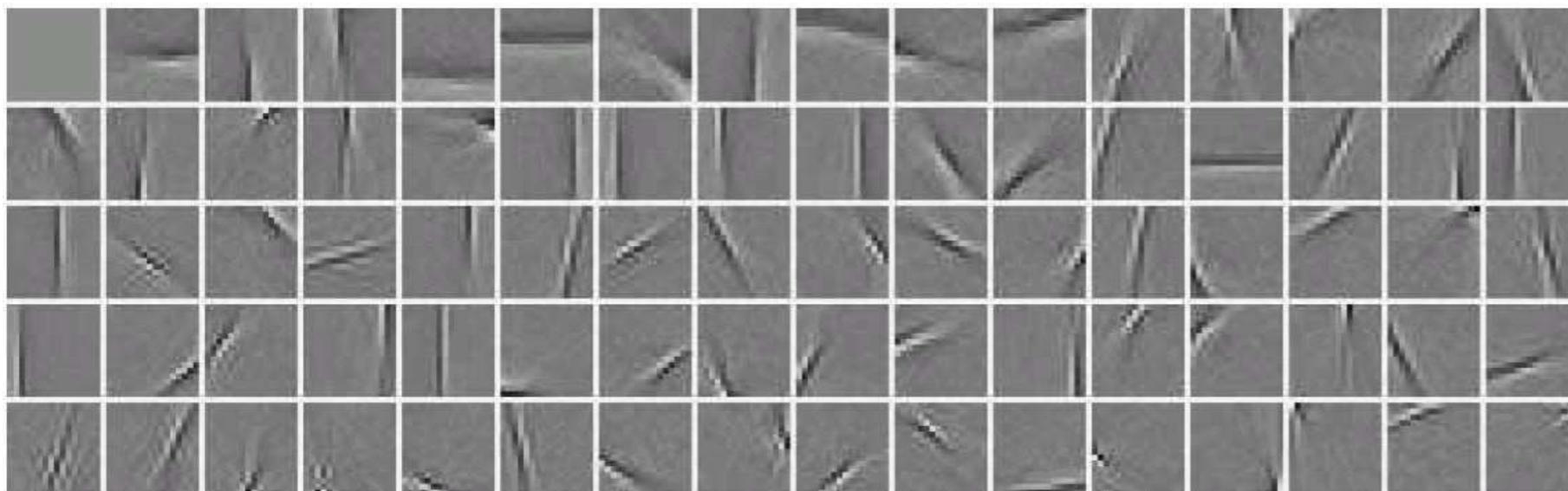
**mixing  
matrix**

# Independent Component Analysis (ICA)

- Application scenarios



EEG analysis



ICA components for  
16x16-patches  
of natural images;  
see that independent  
components look  
like edge detectors

# Independent Component Analysis (ICA)

$$x(t) = \underline{A} s(t)$$

**mixing matrix**

**find unmixing matrix  $W = A^{-1}$ , such that given  $x(t)$ ,  
we can recover sources by  $s(t) = Wx(t)$**

$$W = \begin{bmatrix} -w_1^\top & - \\ \vdots & \\ -w_n^\top & - \end{bmatrix}$$

$$s_j(t) = w_j^\top x_j(t)$$

**given only  $x(t)$ , If we have no prior knowledge about  $s(t)$  and  $A$ ,  
there will be inherent ambiguities in  $A$  that are impossible to recover**

# Independent Component Analysis (ICA)

- Motivation for ICA
  - Assumption: source signals are **independent** random variables → find **independent components** those are maximally independent
  - What kind of independent components can we hope for?

- **non-Gaussian sources**

if  $s(t)$  is Gaussian distributed  $\Rightarrow x = As$  is again Gaussian distributed,  
 $\sim \mathcal{N}(0, I)$

covariance of  $x$  is:  $E [xx^\top] = E [Ass^\top A^\top] = AA^\top$

now let  $R$  orthogonal matrix ( $RR^\top = R^\top R = I$ ), and  $A' = AR$

if we replace  $A$  by  $A'$ , then  $x' = A's$ , the covariance of  $x'$  is still:

$$E [x'(x')^\top] = E [A'ss^\top (A')^\top] = E [ARss^\top (AR)^\top] = AR^\top RA^\top = AA^\top$$

- **sources can be identified only up to rescaling (also sign)**

$x(t) = As(t) = (AD^{-1})(Dst)$  where  $D$  is diagonal matrix

w.l.o.g, set constraint  $\mathbb{E} [s(t)s(t)^\top] = 1_k$  to limit possible solutions

- **assume  $\mathbb{E}[s(t)] = 0$**

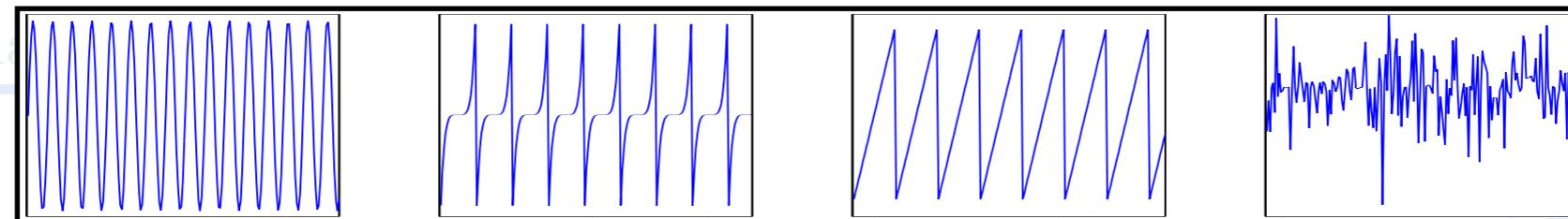
$x(t) - E [x(t)] = A(s - E [s(t)])$ , mixing matrix unchanged. we get set  $E [s(t)] = 0$

- **sources cannot be ordered**

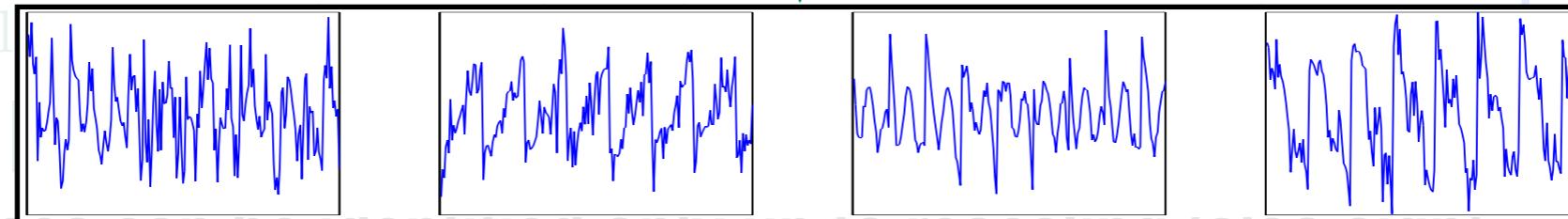
no way to tell  
if  $s(t)$  is mixed  
by  $A$  or  $A'$   
→ limit  $s(t)$  to be  
non-Gaussian

# Independent Component Analysis (ICA)

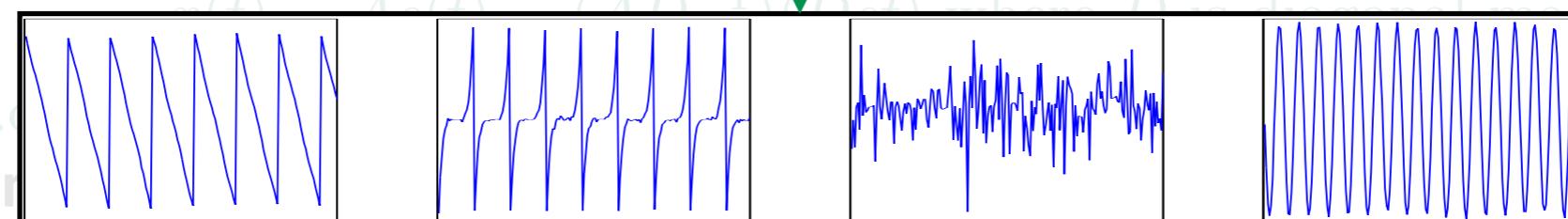
- Motivation for ICA
  - Assumption:** source signals are **independent** random variables  
→ find **independent components** those are maximally independent
  - What kind of independent components can we hope for?
    - non-Gaussian sources



now let  $R$  orthogonal matrix ( $RR^T = R^T R = I$ ), and  $A' = AR$



- sources can be identified only up to rescaling (also sign)



$x(t) - E[x(t)] = A(s - E[s(t)])$ , mixing matrix unchanged. we get set  $E[s(t)] = 0$

- sources cannot be ordered

if you reorder/change permutation of  $s(t)$ , by reordering rows of  $A$  similarly, equation still holds

# Independent Component Analysis (ICA)

- **Whitening as a pre-processing step for ICA**

- whitening  $H$  transforms the signal  $x(t)$ :

$$y(t) = Hx(t) = HA s(t)$$

transformed signals  $y(t)$  becomes **uncorrelated** between each other

$$\underline{1_d} = E[y(t)y(t)^\top] = E[Hx(t)x(t)^\top H^\top] = HA(E[ss^\top])A^\top H^\top = \boxed{HAA^\top H^\top}$$

**uncorrelated (identity matrix):**

$s(t)$  are independent

**covariance b/w different  $y(t)$  = 0;**

$$E[ss^\top] = 1_k$$

**covariance of a  $y(t)$  itself = 1**

- simplifies problem since mixing matrix  $HA$  for  $y(t)$  is **orthogonal**

$$1_d = (HA)(HA)^\top$$

- **New problem: find the orthogonal matrix  $B=HA$**

$$y(t) = Bs(t)$$

such that  $B^\top y(t) = B^\top Bs(t) = s(t)$  is maximally independent

<https://stats.stackexchange.com/questions/122472/why-do-we-whiten-the-data-before-running-ica>

# Independent Component Analysis (ICA)

- Steps for ICA
  - apply whitening to the data:  $y(t) = \mathbf{H} x(t)$
  - find orthogonal demixing matrix  $\mathbf{B}^T$  so  $\mathbf{B}^T y(t)$  is maximally independent

**Different criteria:**

- **maximize non-gaussianity of  $\mathbf{B}^T y(t)$**
- **minimize mutual information between  $\mathbf{B}^T y(t)$**   
**mutual information is zero if and only if joint density of  $\mathbf{B}^T y(t)$  factorizes into the product of the marginal densities  $\rightarrow \mathbf{B}^T y(t)$  is independent**

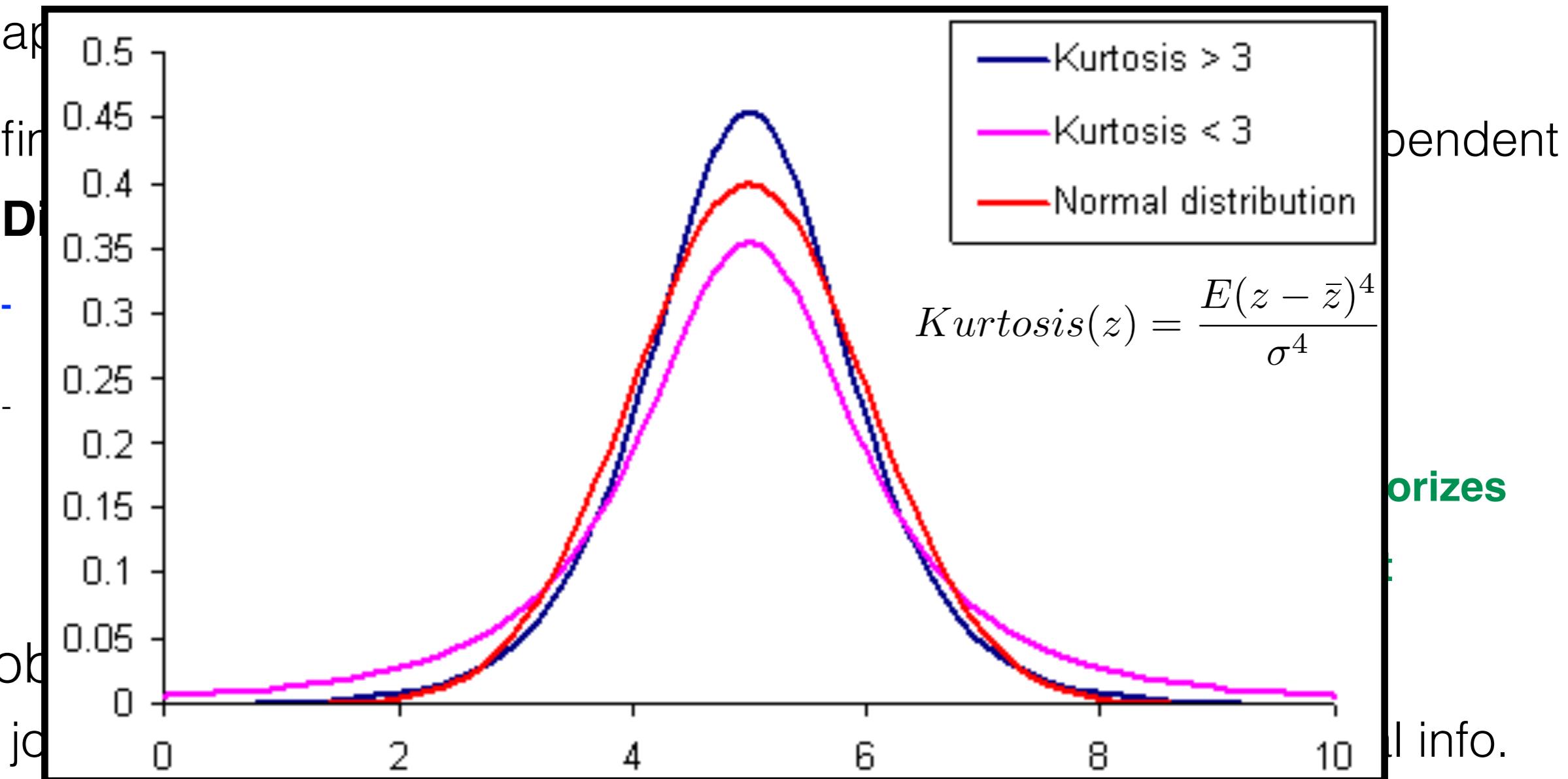
- Problems:
  - joint density of  $\mathbf{B}^T y(t)$  hard to estimate  $\rightarrow$  problems with mutual info.
  - instead: minimize higher order correlations e.g. **kurtosis**

many other methods, please refer to wiki:

# Independent Component Analysis (ICA)

- Steps for ICA

- ▶ approach
  - ▶ first step



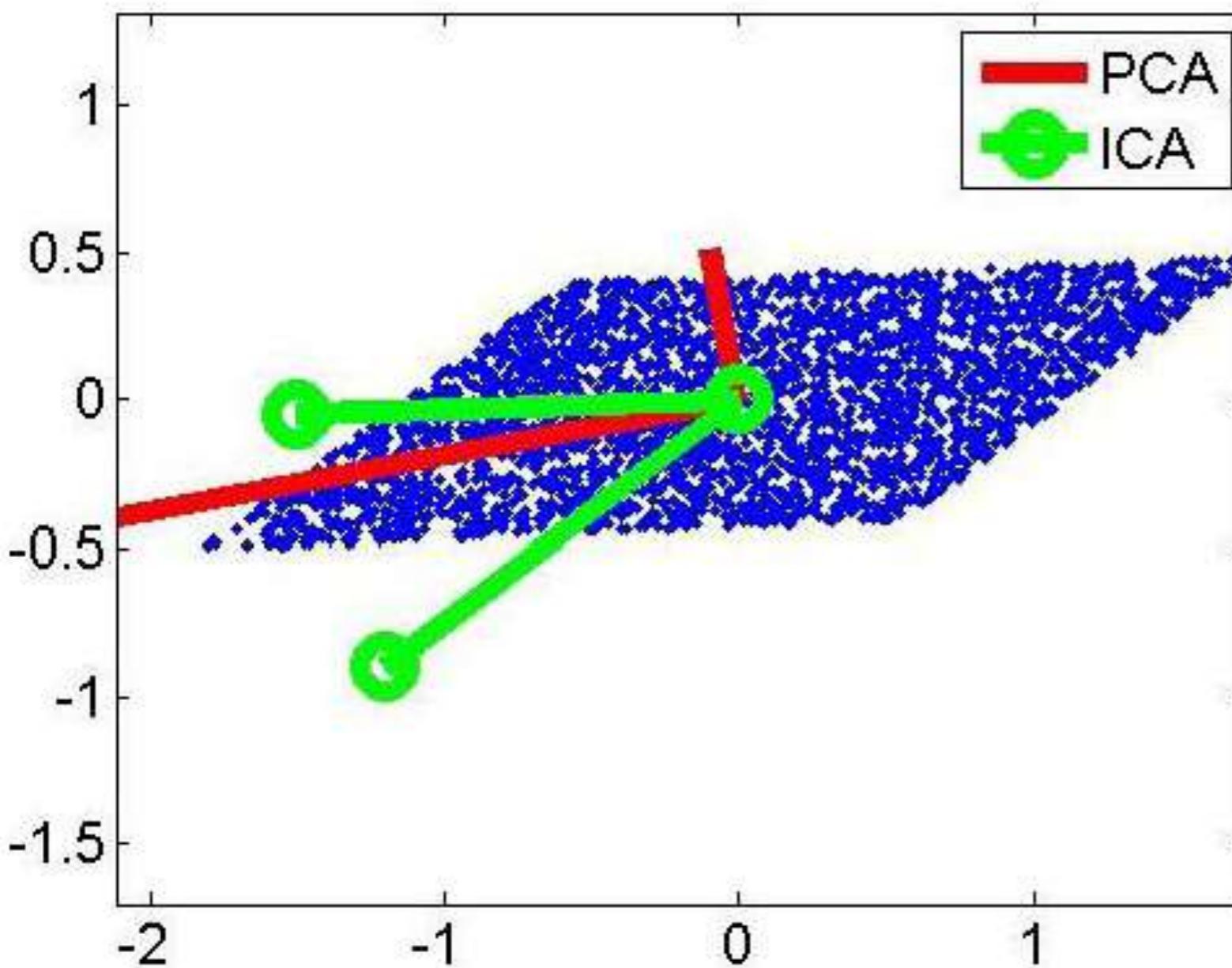
- Probability distributions
  - ▶ joint distributions

- ▶ instead: minimize higher order correlations e.g. **kurtosis**

many other methods, please refer to wiki:

# Independent Component Analysis (ICA)

- ICA versus PCA
  - **Similar:** Both ICA & PCA try to find a ***new basis*** to represent the data!
  - **Different:**



PCA

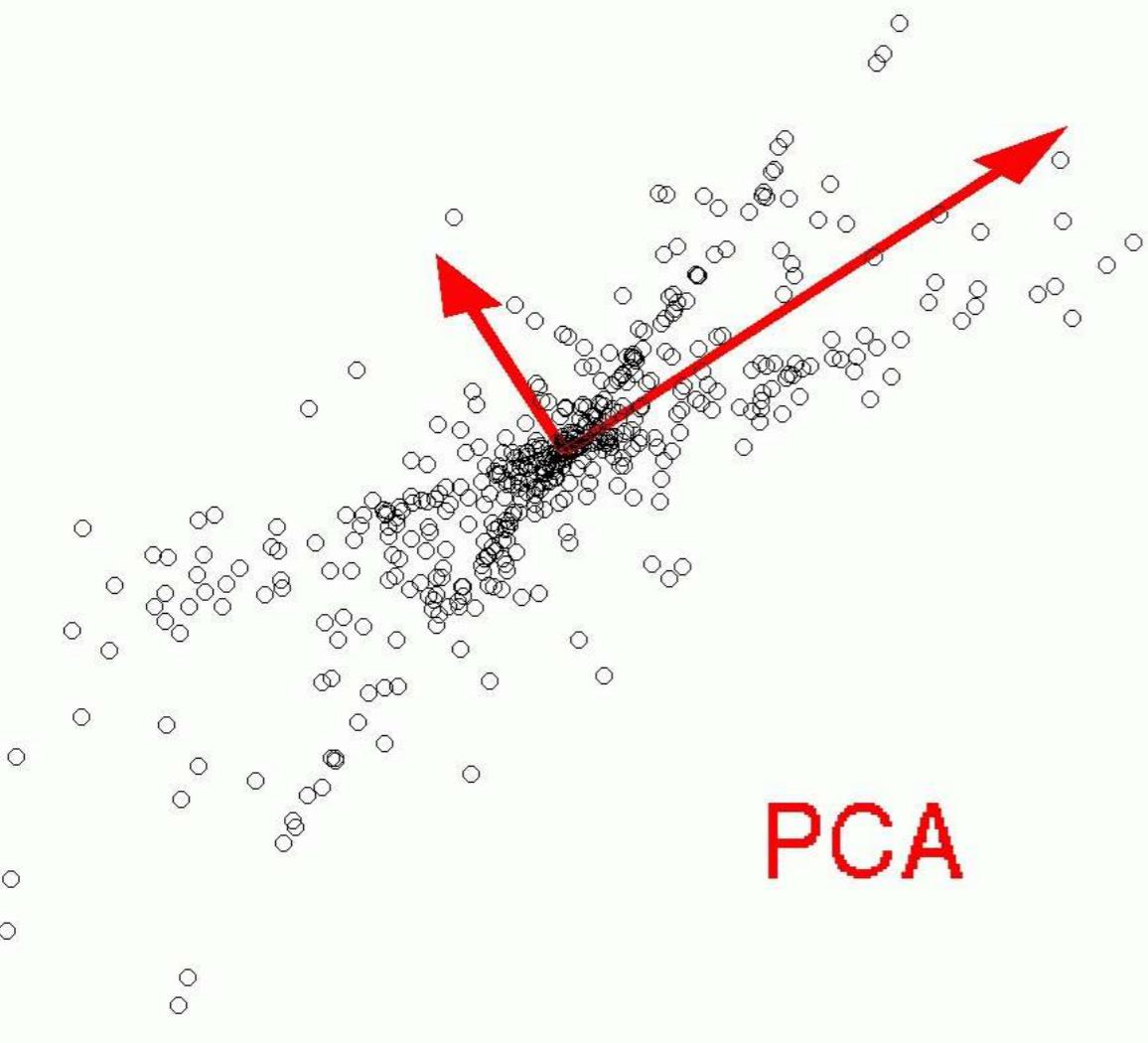
→ principle components  
are orthogonal

ICA

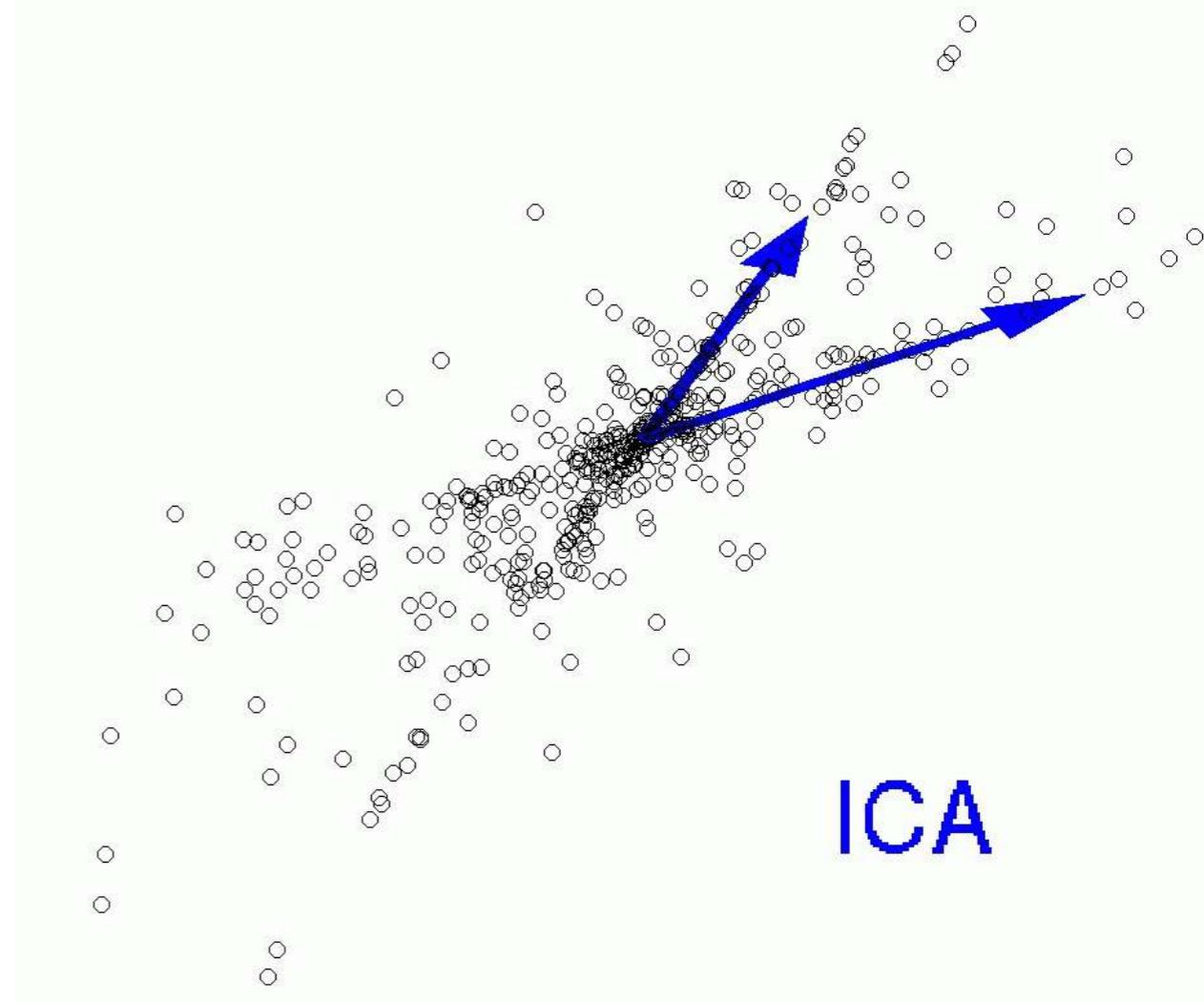
→ independent components  
have no constraints on  
orthogonality

# Independent Component Analysis (ICA)

- ICA versus PCA
  - **Similar:** Both ICA & PCA try to find a ***new basis*** to represent the data!
  - **Different:**



PCA



ICA

maximal variance directions

independent components

# SNE (Stochastic Neighbor Embedding)

- Converting the high-dimensional euclidean distances into **conditional probability that represent similarities**

- Similarity of data points in high dimensionality

**probability of picking  $j$**

**as neighbor of  $i$**

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma_i^2))}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / (2\sigma_i^2))}$$

$$\underline{p_{i|i} = 0}$$

**only care pairwise similarity**

**(the only computation needed in high-d space)**

- Similarity of data points in low dimensionality

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}$$

$$\underline{q_{i|i} = 0}$$

- preserve pairwise similarity between high-d and low-d, want:**

$$p_{j|i} = q_{j|i}$$

use **KL divergence** to measure the **distance b/w distributions of high-d and low-d pairwise similarities**

$$C = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

# SNE (Stochastic Neighbor Embedding)

- Picking radius of gaussian that is used to compute  $p$  (high-D)

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma_i^2))}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / (2\sigma_i^2))}$$

different  $\sigma$  for each  $i$

- We need to use different radii in different parts of the space so that we keep the **effective number of neighbors** about **constant**
  - A big radius leads to a high entropy for the distribution over neighbors of  $i$
  - A small radius leads to a low entropy
  - So decide what entropy you want then find the radius that produces that entropy
- A **fixed perplexity** (a measure of the effective number of neighbors, defined as the two to the power of Shannon entropy)

$$Perp(P_i) = 2^{H(P_i)}$$

**perplexity**

$$H(P_i) = - \sum_j p_{j|i} \log_2 p_{j|i}$$

**entropy**

**for each  $i$ , use binary to search suitable  $\sigma_i$**

# SNE (Stochastic Neighbor Embedding)

- **Asymmetric** KL divergence  $KL(P||Q) \neq KL(Q||P)$

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

- For points where  $p_{j|i}$  is large and  $q_{j|i}$  is small, we lose a lot
  - Nearby points in high-D (large  $p_{j|i}$ ) really want to be nearby in low-D (SNE prefers to preserve local structure)
- For points where  $q_{j|i}$  is large and  $p_{j|i}$  is small we lose a little (we waste some of the probability mass in the  $Q_i$  distribution)
  - Widely separated points in high-D have a soft preference for being widely separated in low-D (it is less harmful to put high-D dissimilar pair in low-D closer neighbor)

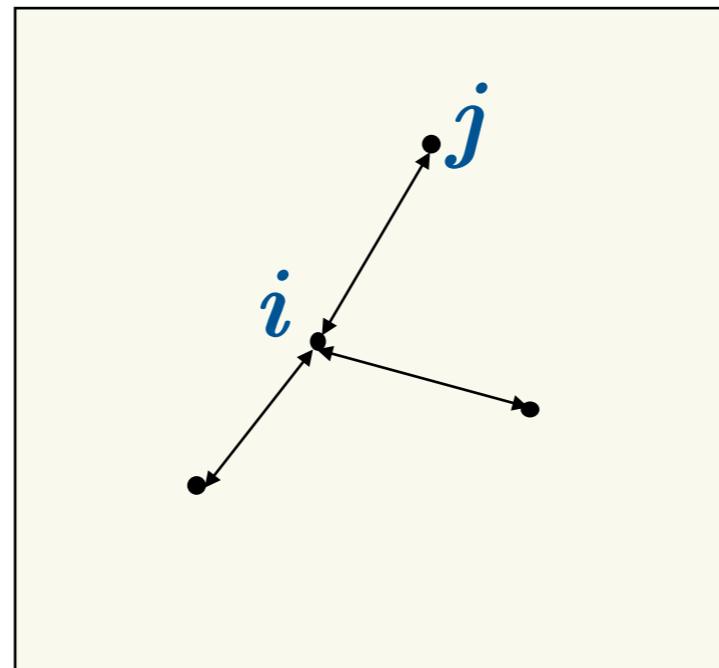
# SNE (Stochastic Neighbor Embedding)

- **Asymmetric** KL divergence  $KL(P||Q) \neq KL(Q||P)$

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

- gradient

$$\frac{\partial C}{\partial y_i} = \sum_{j \neq i} (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$



points are pulled towards each other if  $p$  are bigger than  $q$ ,  
and pushed away if  $q$  are bigger than  $p$

# SNE (Stochastic Neighbor Embedding)

- **Symmetric** SNE

- original cost function is based on “**summation**” of KL divergences between conditional probabilities

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

- We now want to have a simpler cost function based on **a single KL divergence between a joint probability distribution!**

$$C = KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

- The obvious way to **redefine** the pairwise similarities is

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma^2))}{\sum_{k \neq l} \exp(-\|x_l - x_k\|^2 / (2\sigma^2))}$$

**same  $\sigma$  for all;  
compare with all  
pairwise distance**

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_l - y_k\|^2)}$$

# SNE (Stochastic Neighbor Embedding)

- **Symmetric** SNE

- such that  $p_{ji} = p_{ij}$ ,  $q_{ji} = q_{ij}$ , the main advantage is simplifying gradient

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

- However, in practice we symmetrize (or average) the conditionals

**joint probability of  
picking the pair  $i, j$**

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$



$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma^2))}{\sum_{k \neq l} \exp(-\|x_l - x_k\|^2 / (2\sigma^2))}$$

**same  $\sigma$  for all;  
compare with all  
pairwise distance**

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_l - y_k\|^2)}$$

# SNE (Stochastic Neighbor Embedding)

- **Symmetric** SNE

- such that  $p_{ji} = p_{ij}$ ,  $q_{ji} = q_{ij}$ , simplifying gradient

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

- However, in practice we symmetrize (or average) the conditionals

**joint probability of  
picking the pair  $i, j$**

$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$  **original computation of  $p_{ij}$  will have very small value if  $x_i$  is outlier (where  $\|x_i - x_j\|^2$  is large), thus  $y_i$  has little contribution to cost we want to have all data contribute!**



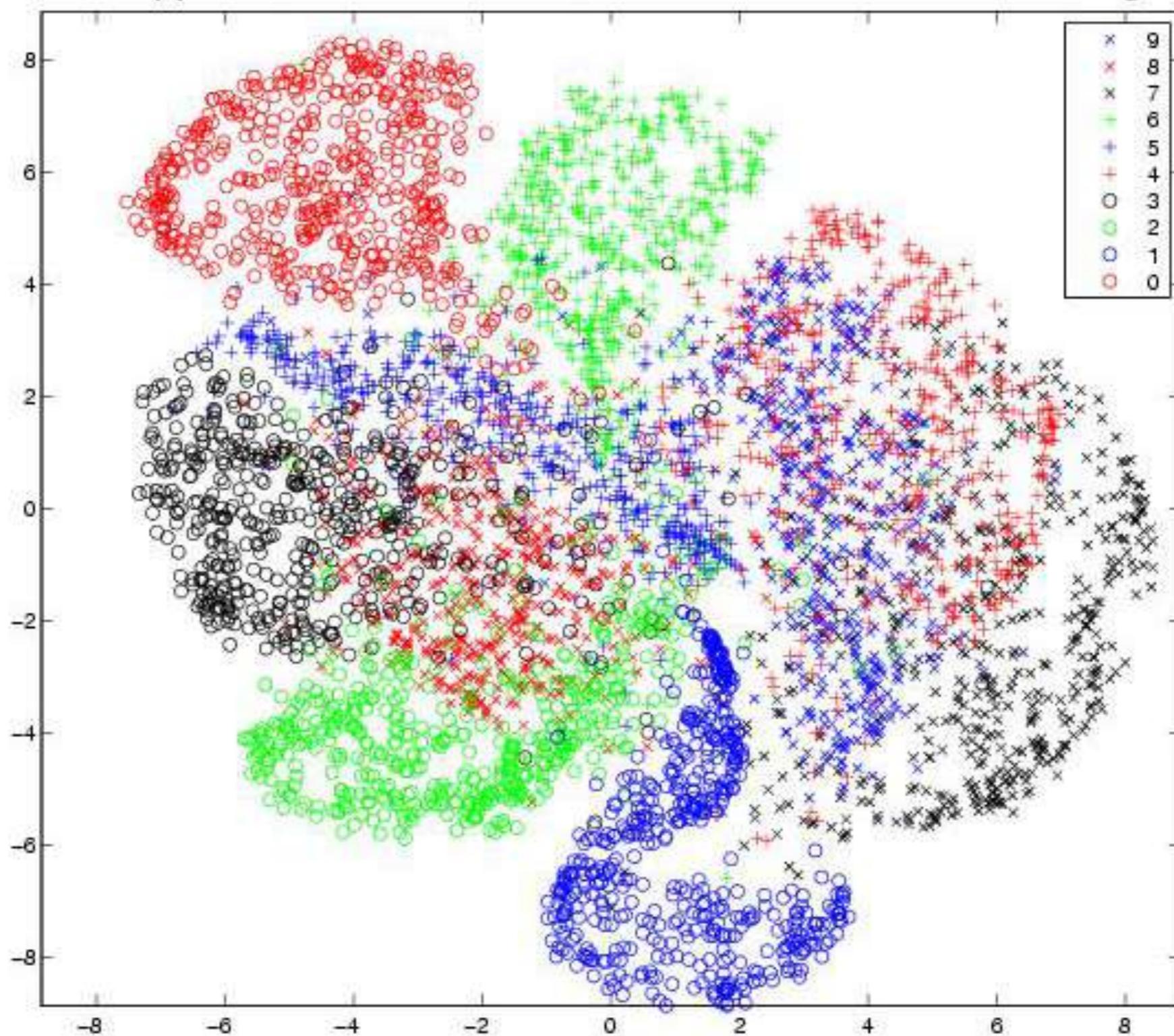
$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma^2))}{\sum_{k \neq l} \exp(-\|x_l - x_k\|^2 / (2\sigma^2))}$$

**same  $\sigma$  for all;  
compare with all pairwise distance**

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_l - y_k\|^2)}$$

# SNE (Stochastic Neighbor Embedding)

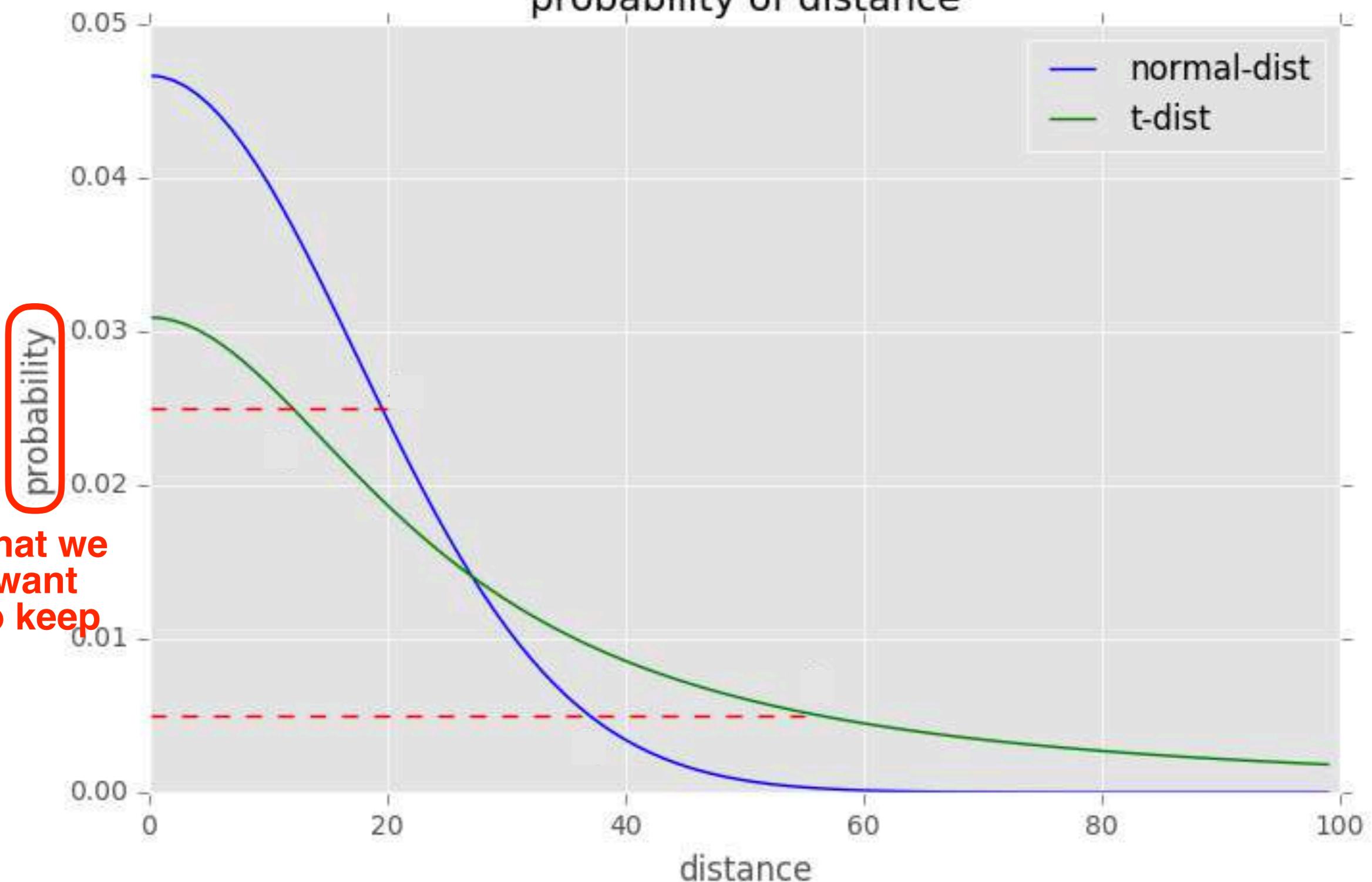
SNE applied to 30-dimensional PCA codes of 5000 MNIST digits



# t-SNE

- **Crowding problem:** when the output dimensionality is smaller than the effective dimensionality of data on the input, the **neighborhoods are mismatched:**
  - in a high-D space, points can have many close-by neighbors in different directions. In a 2D space, you essentially have to arrange close-by neighbors in a circle around the central point, which constrains relationships among neighbors
  - in a high-D space you can have many points that are equidistant from one another; in 2D, at most 3 points are equidistant from each other
  - volume of a sphere scales as  $r^d$  in  $d$  dimensions
    - on a 2D display, there is much less area available at radius  $r$  than the corresponding volume in the original space

## probability of distance



in low-D, small probability can be achieved by using “not-so-far” distance, therefore will be crowded (points do not need to be too far to achieve low probability)  
→ use distribution with longer-tail, such that data should be further away in low-D in order to achieve low probability  
→ Student t-distribution in low-D (Gaussian still in high-D)

# t-SNE

- In high-D, Gaussian disb. to turn distances into probabilities; in low-D, Student t-disb. is used to alleviate crowding problem

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma^2))}{\sum_{k \neq l} \exp(-\|x_l - x_k\|^2 / (2\sigma^2))}$$

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_i - y_j\|^2)^{-1}}$$

$$C = KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

large  $p_{ij}$  modeled by small  $q_{ij}$ : large penalty  
small  $p_{ij}$  modeled by large  $q_{ij}$ : small penalty

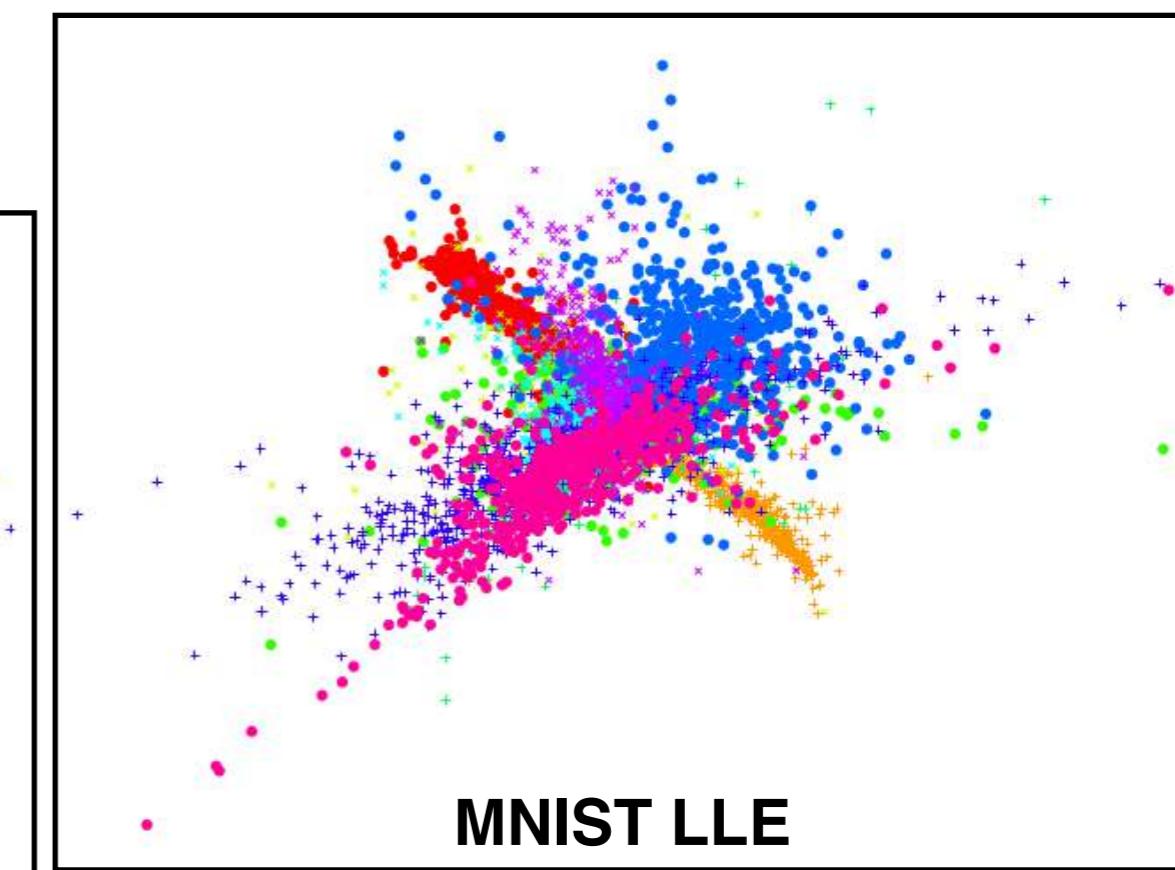
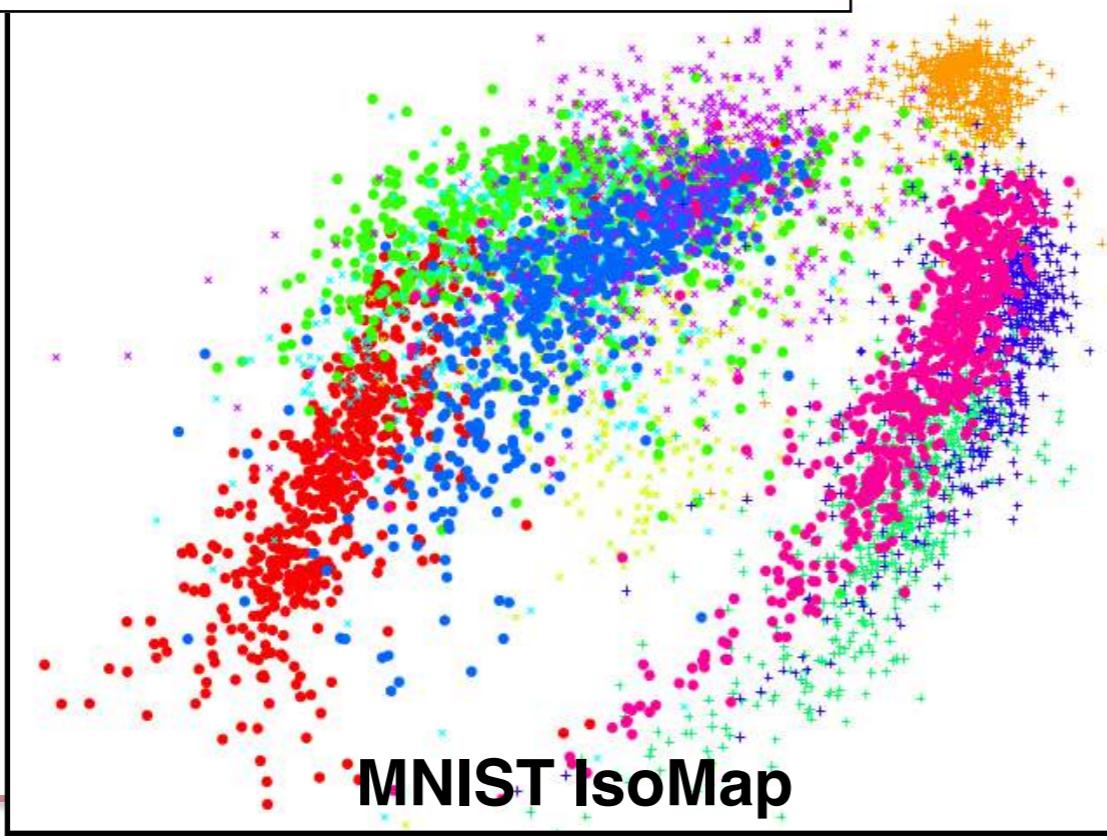
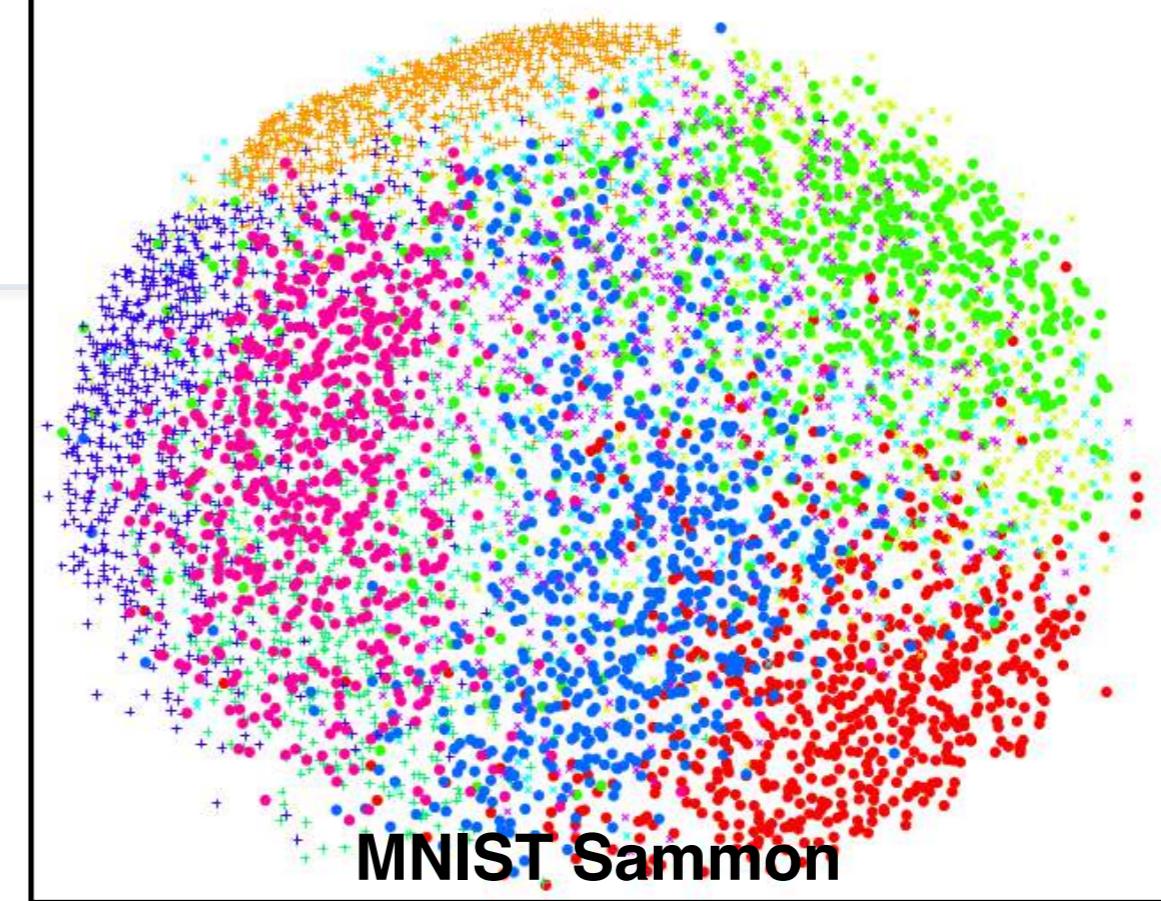


t-SNE mainly preserves local similarity structure of data

- gradient:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}$$

9



# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

Supervised!

You know the corresponding class for each data point!

given  $n$  data samples  $\{x_1, x_2, \dots, x_n\} \in \mathbb{R}^D$

assume they belong to 2 classes:  $\mathcal{C}_1$  and  $\mathcal{C}_2$

and  $|\mathcal{C}_1| = n_1, |\mathcal{C}_2| = n_2, n_1 + n_2 = n$

we want to project data onto a line parameterized by a unit vector  $w$ :  $y = w^\top x$   
such that projected data of  $\mathcal{C}_1$  is maximally separated from projected data of  $\mathcal{C}_2$   
if we use the distance between centers of classes as measure:

$$\mathbf{m}_j = \frac{1}{n_j} \sum_{i \in \mathcal{C}_j} x_i, \quad j = 1, 2$$

$$m_j = \frac{1}{n_j} \sum_{i \in \mathcal{C}_j} y_i = \sum_{i \in \mathcal{C}_j} w^\top x_i = w^\top \mathbf{m}_j, \quad j = 1, 2$$

$|m_2 - m_1| = |w^\top(\mathbf{m}_2 - \mathbf{m}_1)|$  is named as *between-class scatter*  
with  $\|w\| = 1$  (unit vector)

# Linear Discriminative Analysis (LDA)

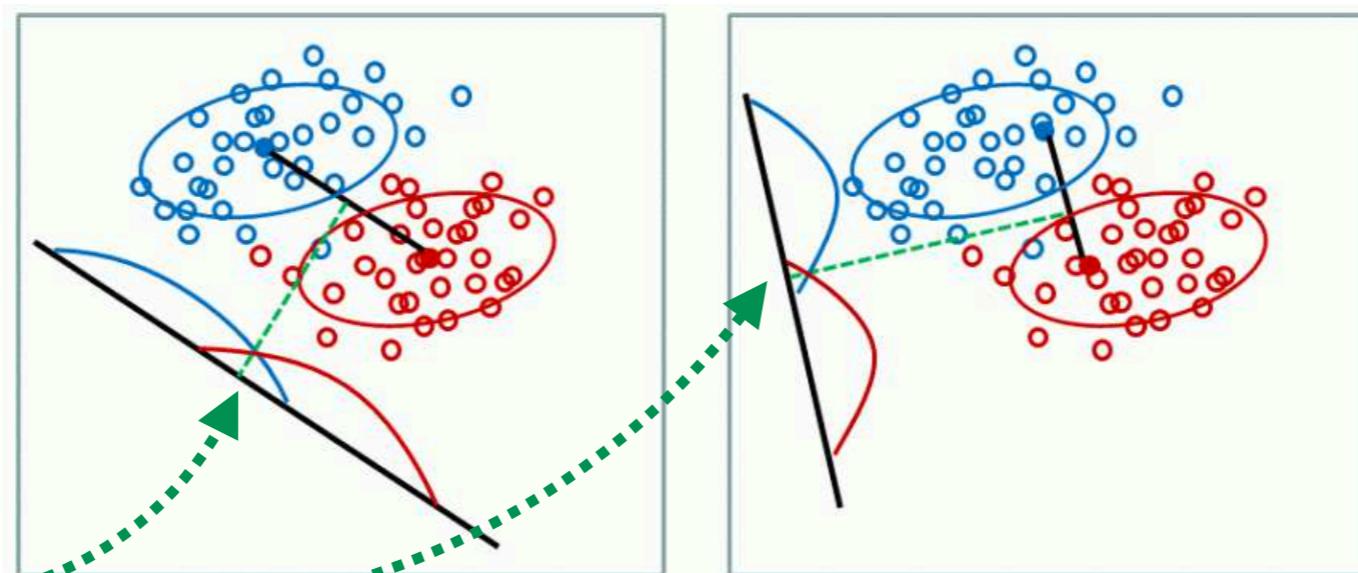
## - Fisher's discriminant analysis

maximize *between-class scatter*  $|m_2 - m_1| = |\mathbf{w}^\top(\mathbf{m}_2 - \mathbf{m}_1)|$  subject to  $\|\mathbf{w}\| = 1$

$$\begin{aligned} \text{use Lagrange } \Rightarrow L(\lambda, \mathbf{w}) &= (\mathbf{w}^\top(\mathbf{m}_2 - \mathbf{m}_1))^2 - \lambda(\mathbf{w}^\top \mathbf{w} - 1) \\ &= \mathbf{w}^\top(\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top \mathbf{w} - \lambda(\mathbf{w}^\top \mathbf{w} - 1) \end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{w}} = 2(\mathbf{m}_2 - \mathbf{m}_1)(\underline{(\mathbf{m}_2 - \mathbf{m}_1)^\top \mathbf{w}} - 2\lambda \mathbf{w})$$

**scalar**  
 $\rightarrow \mathbf{w} \propto \mathbf{m}_2 - \mathbf{m}_1$



**decision boundary  
after projection  
(perpendicular to  $\mathbf{w}$ )**

**not only want to maximize between-class scatter  
but also minimize variance of each projected class!  
 $\rightarrow$  minimize *within-class scatter***

# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

$$\text{within-class scatter: } s_j^2 = \sum_{i \in \mathcal{C}_j} (y_i - m_j)^2, \quad j = 1, 2$$

(if divid by  $(n_j - 1)$  then get variance for certain class)

then the within-class scatter for all data:  $s_1^2 + s_2^2$

we want maximize between-class scatter and minimize within-class scatter:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

$$\begin{aligned} s_j^2 &= \sum_{i \in \mathcal{C}_j} (\mathbf{w}^\top \mathbf{x}_i - \mathbf{w}^\top \mathbf{m}_j)^2 \\ &= \sum_{i \in \mathcal{C}_j} \mathbf{w}^\top (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^\top \mathbf{w} \\ &= \mathbf{w}^\top \left( \sum_{i \in \mathcal{C}_j} (\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^\top \right) \mathbf{w} \\ &= \mathbf{w}^\top S_j \mathbf{w} \\ \text{then } s_1^2 + s_2^2 &= \mathbf{w}^\top S_1 \mathbf{w} + \mathbf{w}^\top S_2 \mathbf{w} = \mathbf{w}^\top S_W \mathbf{w} \end{aligned}$$

$$\begin{aligned} (m_2 - m_1)^2 &= (\mathbf{w}^\top \mathbf{m}_2 - \mathbf{w}^\top \mathbf{m}_1)^2 \\ &= \mathbf{w}^\top (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top \mathbf{w} \\ &= \mathbf{w}^\top S_B \mathbf{w} \end{aligned}$$

$$\Rightarrow J(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}}$$

by Rayleigh quotient  $\Rightarrow S_B \mathbf{w} = \lambda S_W \mathbf{w}$

# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

$$\Rightarrow J(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}}$$

by Rayleigh quotient  $\Rightarrow S_B \mathbf{w} = \lambda S_W \mathbf{w}$

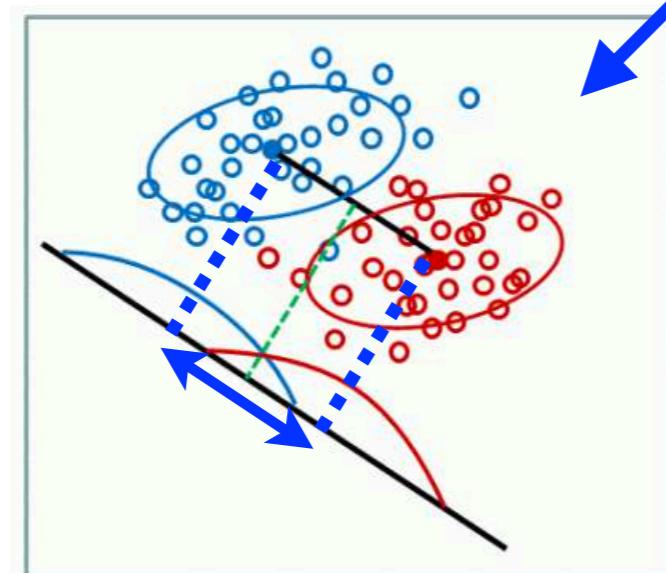
we assume  $S_W$  is invertible (yes if  $n > D$  which hosts most of time)

$$\Rightarrow S_W^{-1} \underline{S_B \mathbf{w}} = \lambda \mathbf{w}$$

as  $S_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^\top$

then  $\underline{S_B \mathbf{w}} = (\mathbf{m}_2 - \mathbf{m}_1) \underline{(\mathbf{m}_2 - \mathbf{m}_1)^\top \mathbf{w}} = (\mathbf{m}_2 - \mathbf{m}_1) * \underline{\lambda_w}$

since  $S_B$  is the outer product of two vectors, its rank is at most 1



$$\Rightarrow S_W^{-1} S_B \mathbf{w} = S_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1) * \lambda_w = \lambda_w$$

since it won't matter if  $\mathbf{w}$  is scaled, then:

$$\Rightarrow \mathbf{w} = S_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$$

# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

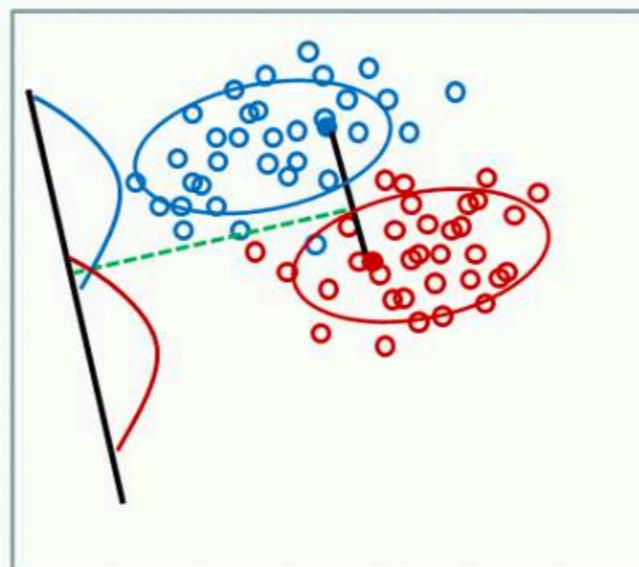
$$\mathbf{w} = S_w^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

we can choose  $w_0$  as a threshold to separate the data, for instance:

$$\Rightarrow x_i \in \mathcal{C}_1 \text{ if } \mathbf{w}^\top \mathbf{x}_i \leq -w_0 ; x_i \in \mathcal{C}_2 \text{ if } \mathbf{w}^\top \mathbf{x}_i \geq -w_0$$

There is no general rule for the threshold  $w_0$

However, if projections of points from both classes exhibit approximately the same distributions, (i.e remove within-class scatter by having  $S_W^{-1}$ )  
a good choice would be the hyperplane between projections of the two means



$$w_0 = \mathbf{w}^\top \frac{1}{2}(\mathbf{m}_2 + \mathbf{m}_1) = \frac{1}{2}\mathbf{m}_2^\top s_2^{-2}\mathbf{m}_2 - \frac{1}{2}\mathbf{m}_1^\top s_1^{-2}\mathbf{m}_1$$

# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

if now we are dealing with multi-class cases  $(\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k)$ :

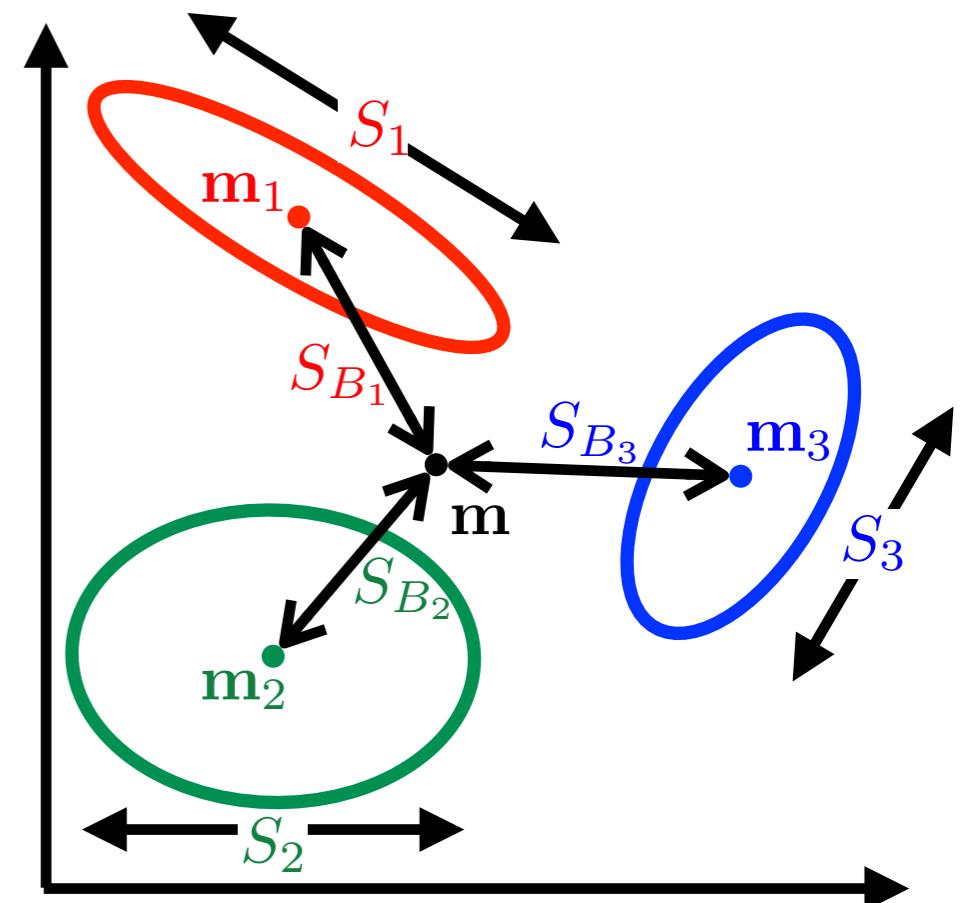
*within-class scatter:*  $S_W = \sum_{j=1}^k S_j$ , where  $S_j = \sum_{i \in \mathcal{C}_j} (x_i - \mathbf{m}_j)(x_i - \mathbf{m}_j)^\top$

$$\text{and } \mathbf{m}_j = \frac{1}{n_j} \sum_{i \in \mathcal{C}_j} x_i$$

*between-class scatter:*

$$S_B = \sum_{j=1}^k S_{B_j} = \sum_{j=1}^k n_j (\mathbf{m}_j - \mathbf{m})(\mathbf{m}_j - \mathbf{m})^\top$$

$$\text{where } \mathbf{m} = \frac{1}{n} \sum x$$



# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

we would like to project data into  $q$  space:  $w_1, w_2, \dots, w_q$

$$y_l = w_l^\top x, \quad l = 1, \dots, q,$$

$$\text{or } y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_q \end{bmatrix} = \begin{bmatrix} w_1^\top x \\ w_2^\top x \\ \vdots \\ w_q^\top x \end{bmatrix} = [w_1 \ w_2 \ \dots \ w_q] x = \underline{W}^\top x$$

**$d \times q$  matrix**

$$\rightarrow \tilde{S}_W = \sum_{j=1}^k \sum_{i \in \mathcal{C}_j} (y_i - \tilde{\mathbf{m}}_j)(y_i - \tilde{\mathbf{m}}_j)^\top = W^\top S_W W$$

$$\tilde{S}_B = \sum_{j=1}^k n_j (\tilde{\mathbf{m}}_j - \tilde{\mathbf{m}})(\tilde{\mathbf{m}}_j - \tilde{\mathbf{m}})^\top = W^\top S_B W$$

where  $\tilde{\mathbf{m}}_j = W^\top \mathbf{m}_j$  and  $\tilde{\mathbf{m}} = W^\top \mathbf{m}$

$$\rightarrow J_1(W) = \text{trace}(\tilde{S}_W^{-1} \tilde{S}_B) = \text{trace}((W^\top S_W W)^{-1} (W^\top S_B W))$$

$$J_2(W) = \frac{\det \tilde{S}_B}{\det \tilde{S}_W} = \frac{\det (W^\top S_B W)}{\det (W^\top S_W W)}$$

**both  $J_1$  &  $J_2$  relate to**  
 $\rightarrow S_B w_l = \lambda_l S_W w_l$

# Linear Discriminative Analysis (LDA)

## - Fisher's discriminant analysis

we would like to project data into  $q$  space:  $w_1, w_2, \dots, w_q$

$$y_l = w_l^\top x, \quad l = 1, \dots, q,$$

$$\text{or } y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_q \end{bmatrix} = \begin{bmatrix} w_1^\top x \\ w_2^\top x \\ \vdots \\ w_q^\top x \end{bmatrix} = [w_1 \ w_2 \ \dots \ w_q] x = \underline{W}^\top x$$

*d x q matrix*

**practical issue: how if  $n < D$ ?  $S_W$  becomes invertible**

→ use pseudo inverse or shrinkage estimator of covariance matrix  
(ref: [https://en.wikipedia.org/wiki/Linear\\_discriminant\\_analysis](https://en.wikipedia.org/wiki/Linear_discriminant_analysis))

get first  $q$  largest eigenvectors of  $S_W^{-1} S_B$  as  $W$

note that  $q \leq k$  since  $\text{rank}(S_B) = \dim \text{span}\{\mathbf{m}_1 - \mathbf{m}, \dots, \mathbf{m}_k - \mathbf{m}\} \leq k - 1$

**$S_B$  is sum of  $k$  matrices of rank 1 or less**

**and the mean vectors are constrained by  $\mathbf{m} = (1/k) \sum \mathbf{m}_k$**

**Therefore,  $S_B$  will be of rank ( $k-1$ ) or less**



**both  $J_1$  &  $J_2$  relate to**  
→  $S_B w_l = \lambda_l S_W w_l$

# Linear Discriminative Analysis (LDA)

in binary case, LDA assumes both  $p(x|\mathcal{C}_1)$  and  $p(x|\mathcal{C}_2)$  follow normal distribution  
they have different means  $\mathbf{m}_1$  and  $\mathbf{m}_2$ , **but identical covariance  $\Sigma$**   
given a data point  $x$ , how to classify it into  $\mathcal{C}_1$  or  $\mathcal{C}_2$ ? **Bayes's theorem!**

$$\Rightarrow P(\mathcal{C}_j|x) = \frac{p(x|\mathcal{C}_j)P(\mathcal{C}_j)}{p(x)}, \text{ where } \begin{cases} P(\mathcal{C}_j): \text{ prior usually by frequencies } \frac{n_j}{n} \\ p(x|\mathcal{C}_j): \text{ likelihood} \\ p(x): \text{ evidence } p(x) = p(x|\mathcal{C}_1)P(\mathcal{C}_1) + p(x|\mathcal{C}_2)P(\mathcal{C}_2) \\ P(\mathcal{C}_j|x): \text{ posterior} \end{cases}$$

$\Rightarrow$  if  $P(\mathcal{C}_1|x) > P(\mathcal{C}_2|x)$  then  $x \in \mathcal{C}_1$  otherwise  $x \in \mathcal{C}_2$

$$\text{since } \mathcal{N}(x|\mathbf{m}_j, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mathbf{m}_j)^\top \Sigma^{-1} (x - \mathbf{m}_j)\right\}$$

$$P(\mathcal{C}_1|x) > P(\mathcal{C}_2|x) \Rightarrow \log P(\mathcal{C}_1|x) > \log P(\mathcal{C}_2|x)$$

$$\Rightarrow \log p(x|\mathcal{C}_1) + \log P(\mathcal{C}_1) > \log p(x|\mathcal{C}_2) + \log P(\mathcal{C}_2)$$

$$\Rightarrow -\frac{1}{2}(x - \mathbf{m}_1)^\top \Sigma^{-1} (x - \mathbf{m}_1) + \log P(\mathcal{C}_1) > -\frac{1}{2}(x - \mathbf{m}_2)^\top \Sigma^{-1} (x - \mathbf{m}_2) + \log P(\mathcal{C}_2)$$

$$\Rightarrow (\mathbf{m}_2 - \mathbf{m}_1)^\top \Sigma^{-1} x < \frac{1}{2}\mathbf{m}_2^\top \Sigma^{-1} \mathbf{m}_2 - \frac{1}{2}\mathbf{m}_1^\top \Sigma^{-1} \mathbf{m}_1 - \log P(\mathcal{C}_2) + \log P(\mathcal{C}_1)$$

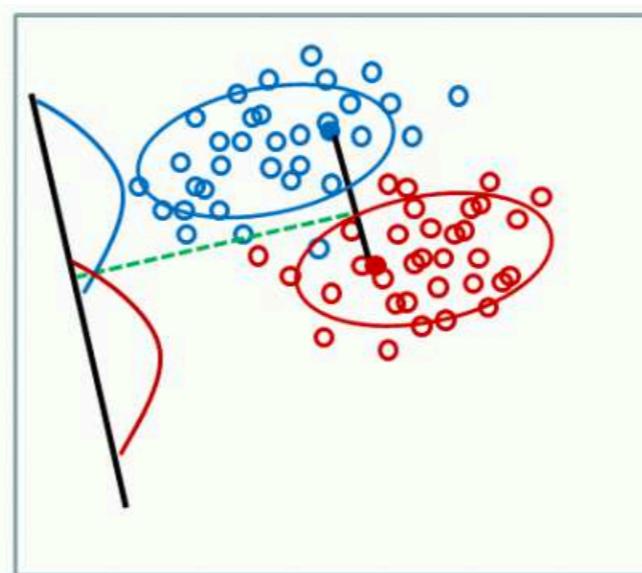
# Linear Discriminative Analysis (LDA)

$$\frac{(\mathbf{m}_2 - \mathbf{m}_1)^\top \Sigma^{-1} x}{\mathbf{w}^\top} < \frac{\frac{1}{2}\mathbf{m}_2^\top \Sigma^{-1} \mathbf{m}_2 - \frac{1}{2}\mathbf{m}_1^\top \Sigma^{-1} \mathbf{m}_1 - \log P(\mathcal{C}_2) + \log P(\mathcal{C}_1)}{\mathbf{w}_0}$$

$$S = \left(\frac{n_1}{n}\right) \frac{1}{n_1} \sum_{i \in \mathcal{C}_1} (x_i - \mathbf{m}_1)(x_i - \mathbf{m}_1)^\top + \left(\frac{n_2}{n}\right) \frac{1}{n_2} \sum_{i \in \mathcal{C}_2} (x_i - \mathbf{m}_2)(x_i - \mathbf{m}_2)^\top$$

$$= \frac{1}{n}(S_1 + S_2) = \frac{1}{n}S_W$$

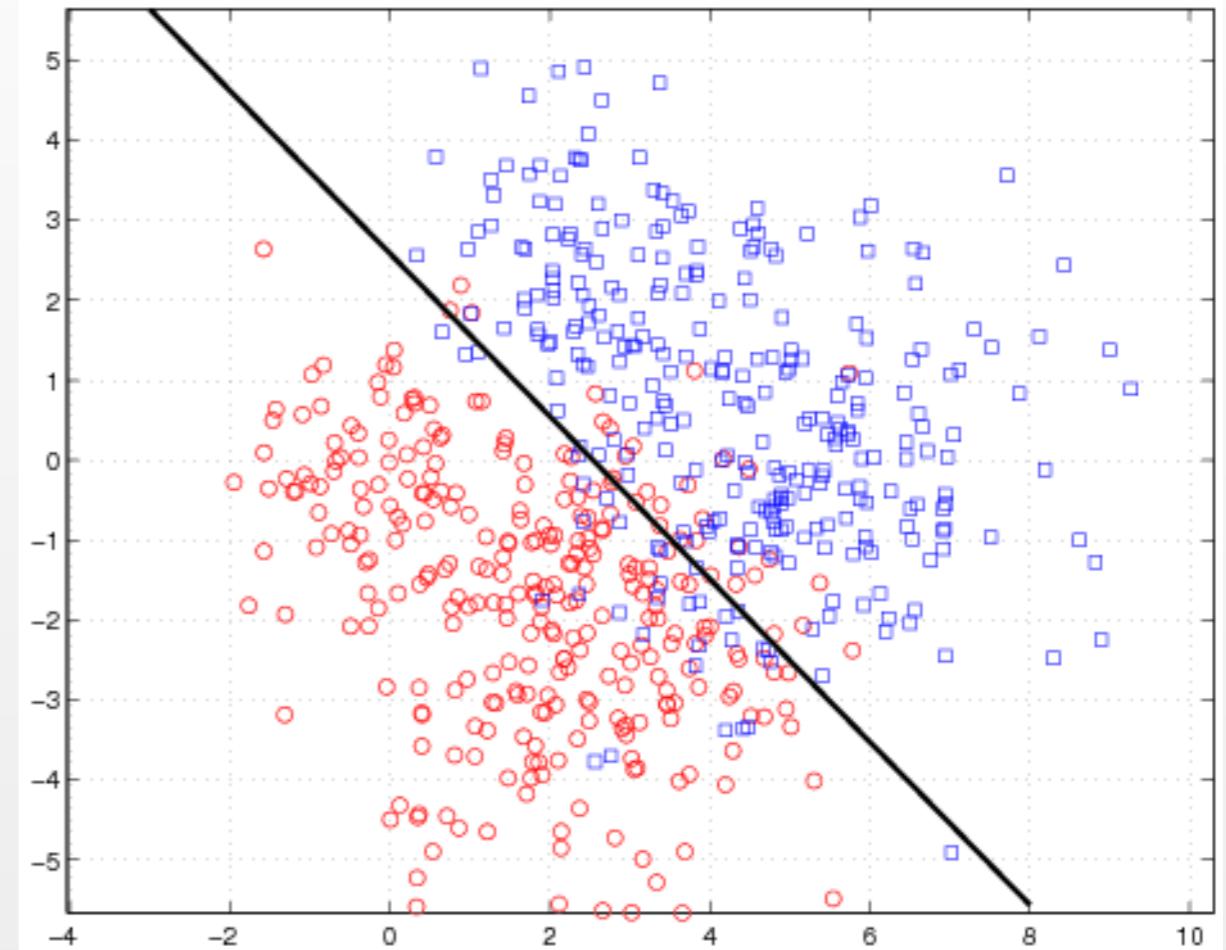
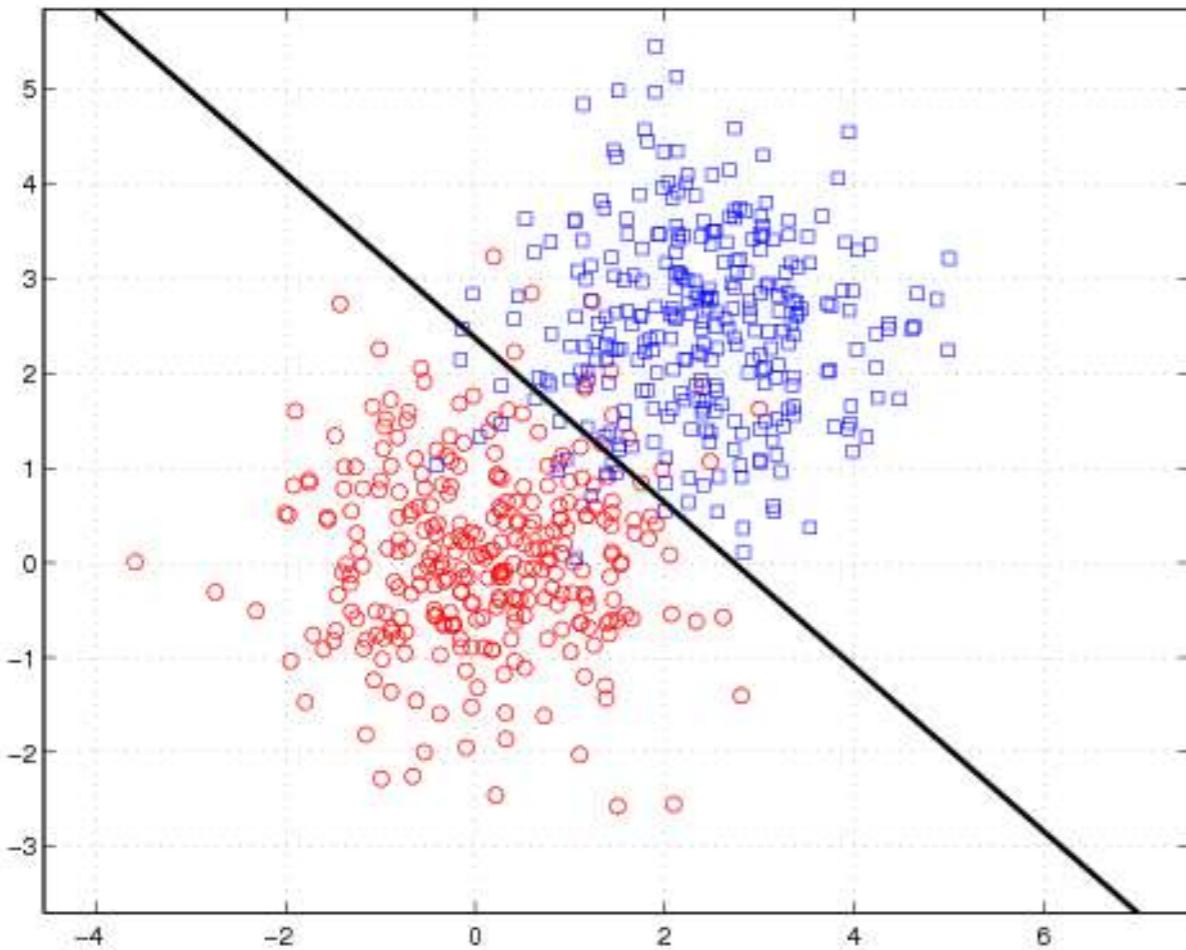
get  $\mathbf{w} = nS_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$  and  $\mathbf{w}_0 = -\frac{n}{2}\mathbf{m}_2^\top S_W^{-1}\mathbf{m}_2 + \frac{n}{2}\mathbf{m}_1^\top S_W^{-1}\mathbf{m}_1 + \log \frac{n_2}{n} - \log \frac{n_1}{n}$



Fisher's  
LDA

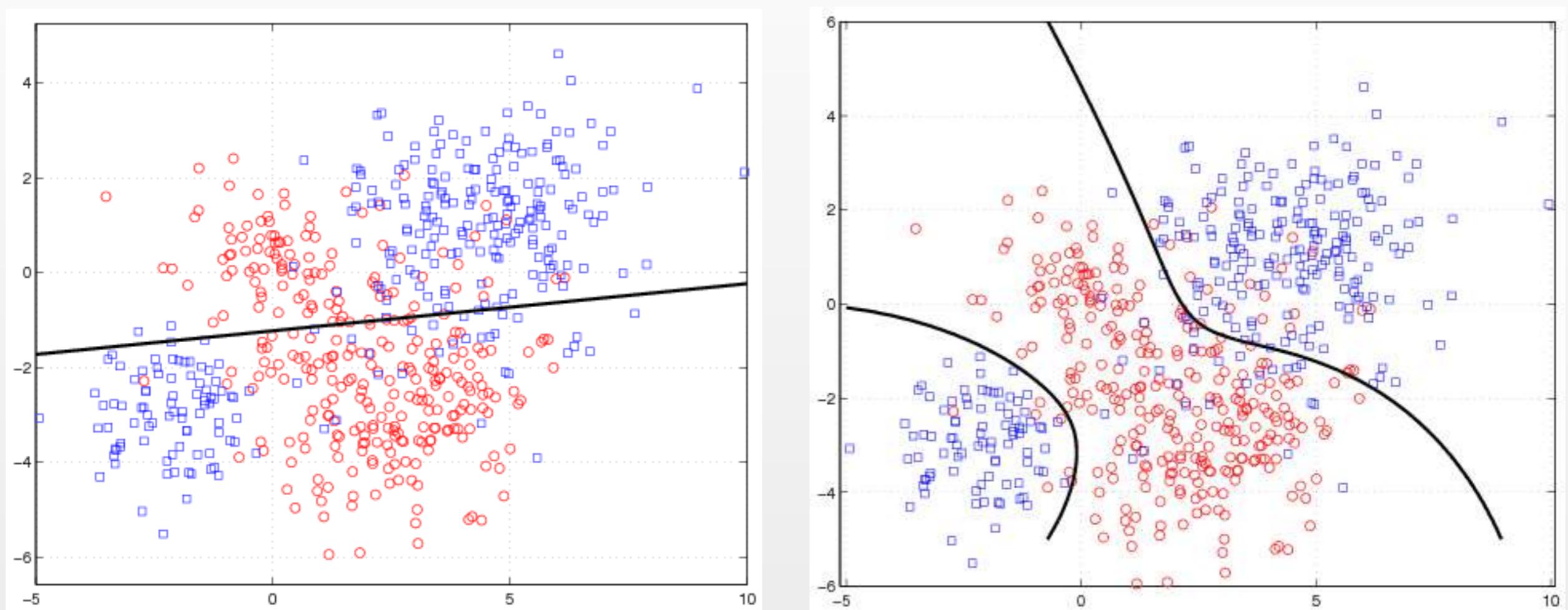
$$\mathbf{w} = S_w^{-1}(\mathbf{m}_2 - \mathbf{m}_1) \quad \mathbf{w}_0 = \mathbf{w}^\top \frac{1}{2}(\mathbf{m}_2 + \mathbf{m}_1) = \frac{1}{2}\mathbf{m}_2^\top s_2^{-2}\mathbf{m}_2 - \frac{1}{2}\mathbf{m}_1^\top s_1^{-2}\mathbf{m}_1$$

# Linear Discriminative Analysis (LDA)



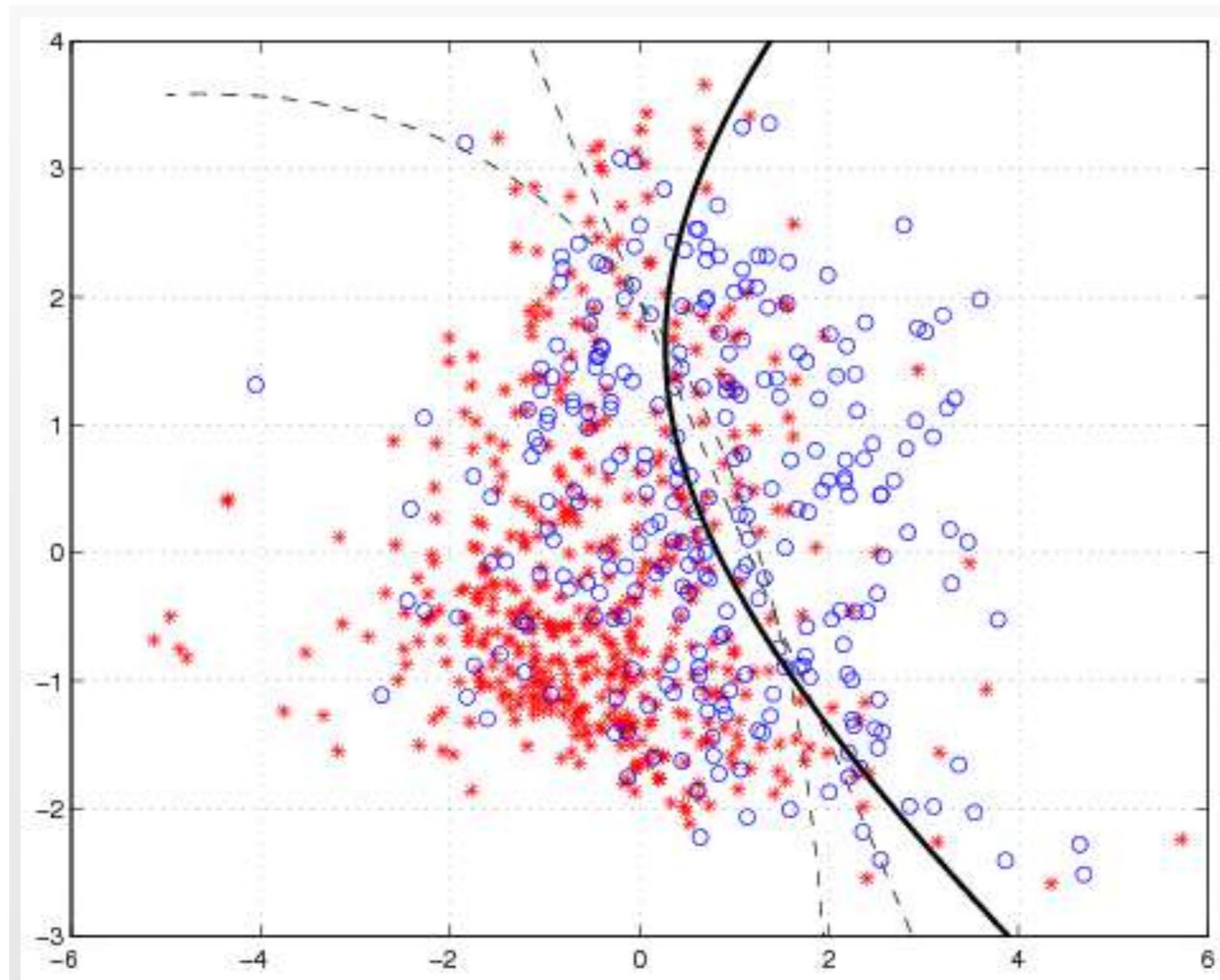
LDA applied to simulated data sets. Left: The true within class densities are Gaussian with identical covariance matrices across classes. Right: The true within class densities are mixtures of two Gaussians.

# Linear Discriminative Analysis (LDA)



Left: Decision boundaries by LDA. Right: Decision boundaries obtained by modeling each class by a mixture of two Gaussians.

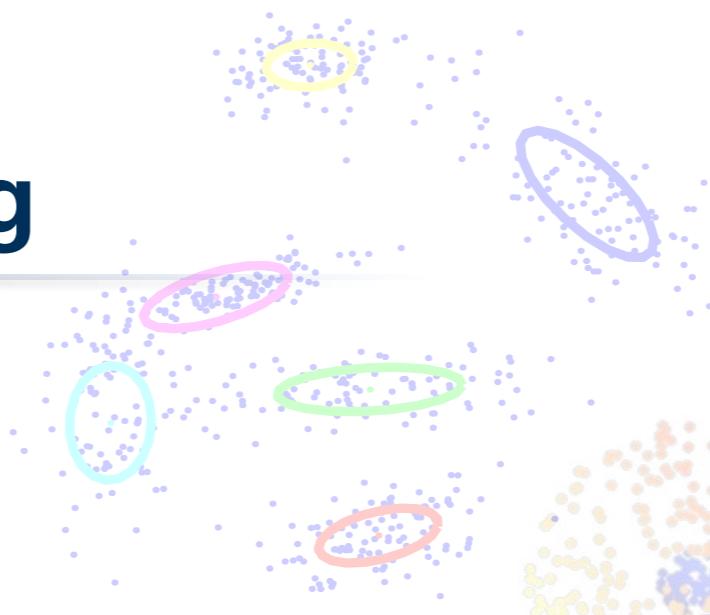
# Linear Discriminative Analysis (LDA)



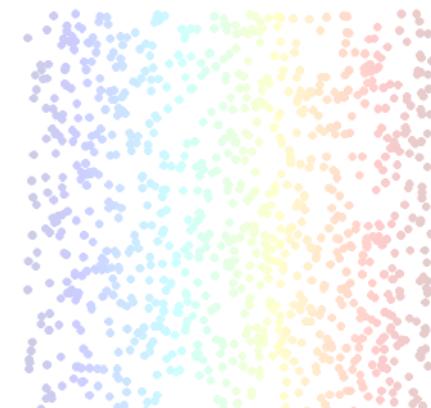
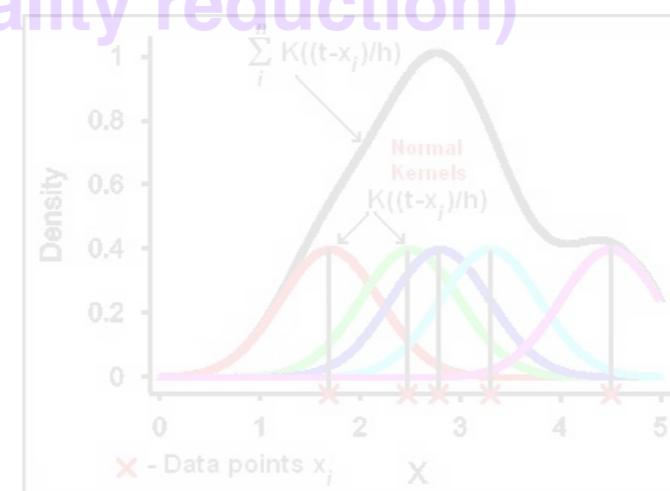
yes, you can also use kernel trick to have nonlinear boundary

# Unsupervised Learning

✓ clustering



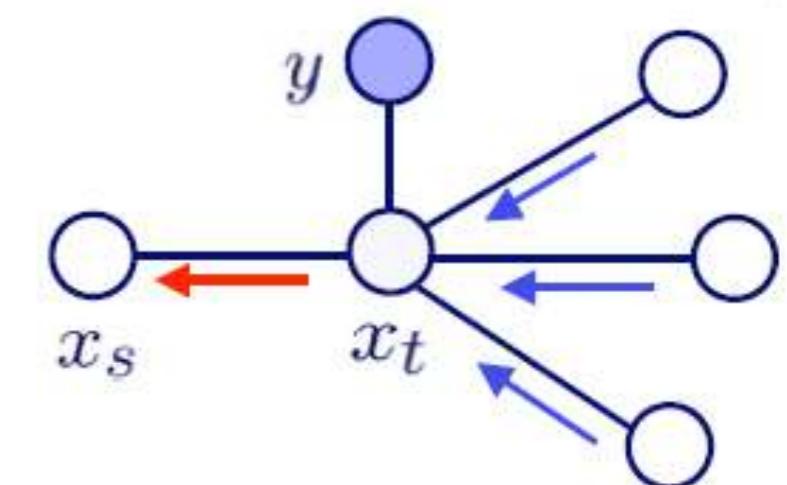
✓ embedding (dimensionality reduction)



Actually, 2-D

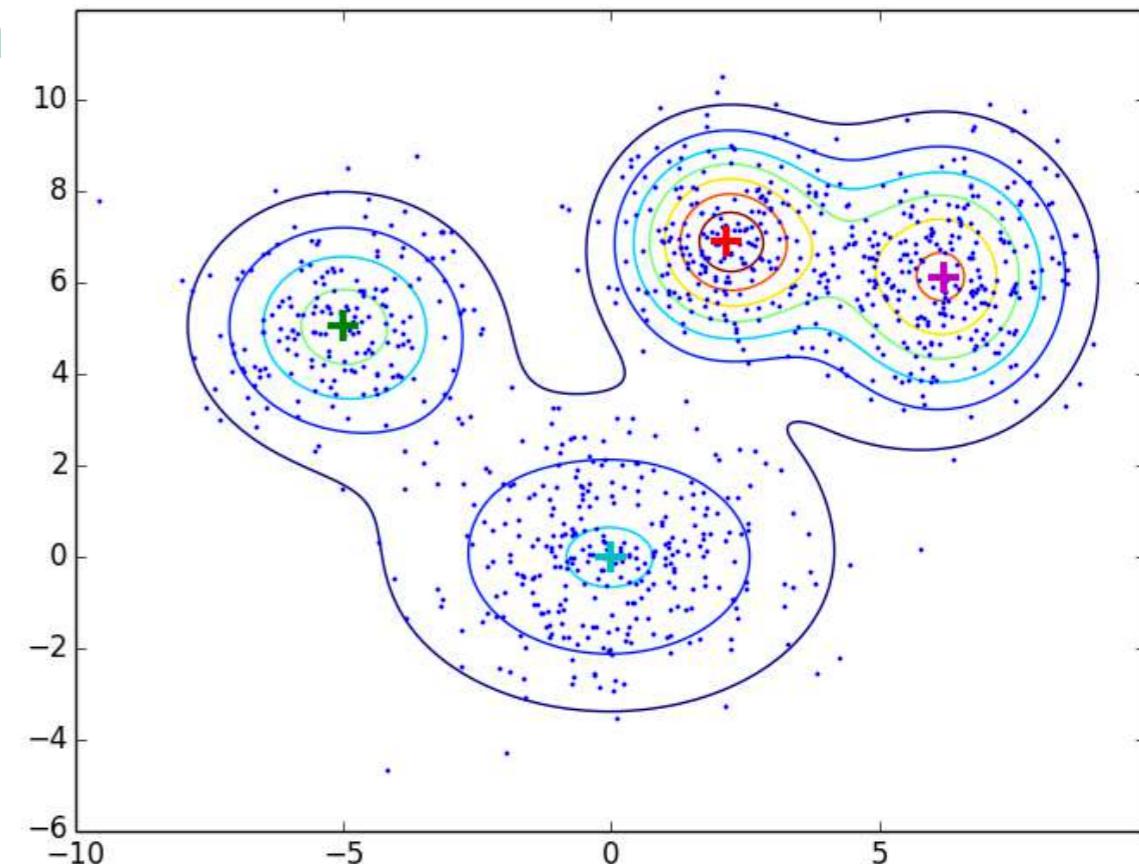
✓ density estimation

✓ finding good explanations  
(hidden causes) of data



# Probabilistic Graphical Models - Classic Example of Gaussian Mixture Model (GMM)

- Define **Generative Procedures**
  - $N$  Data in 2D space distributed as  $K$  Gaussians
  - Choose one from  $K$  Gaussians:  $\mathcal{N}(\mu_k, \Sigma_k)$
  - For each data, sample its position from the corresponding Gaussian:
$$(x_n, y_n) \sim \mathcal{N}(\mu_k, \Sigma_k)$$
- Perform **Inference**
  - We only have  $\{x_n, y_n\}_{n=1\dots N}$
  - Infer **hidden/latent variables**
$$\{\mu_k, \Sigma_k\}_{k=1\dots K}$$
  - Can be done by Expectation-Maximization algorithm: K-means



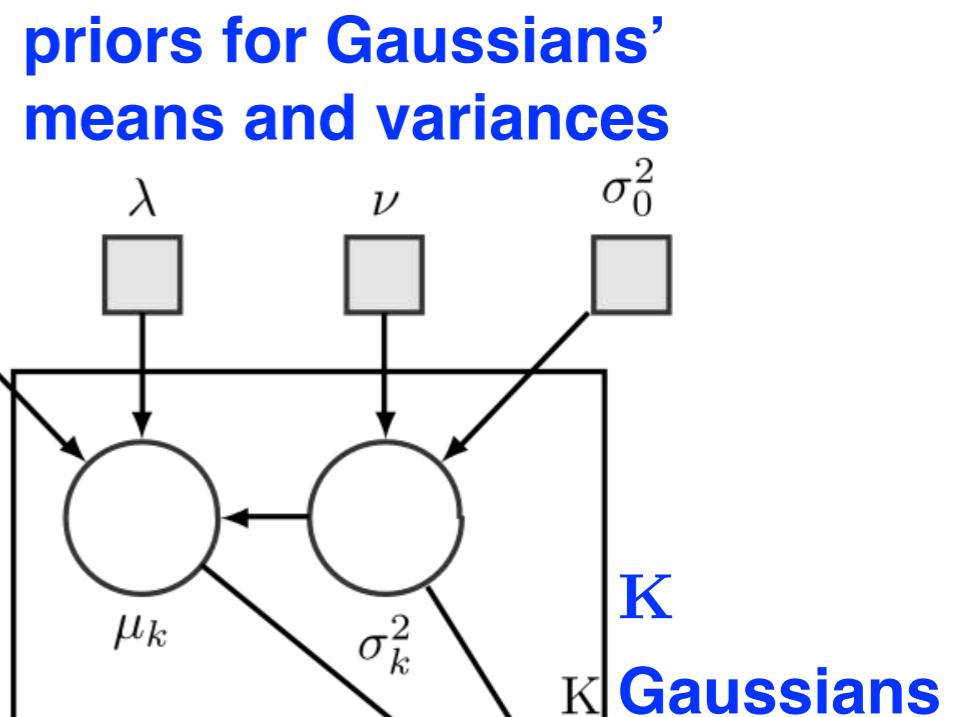
# Probabilistic Graphical Models - Classic Example of Gaussian Mixture Model (GMM)

- Define **Generative Procedures**
  - $N$  Data in 2D space distributed as  $K$  Gaussians
  - Choose one from  $K$  Gaussians:  $\mathcal{N}(\mu_k, \Sigma_k)$
  - For each data, sample its position from the corresponding Gaussian:

$$(x_n, y_n) \sim \mathcal{N}(\mu_k, \Sigma_k)$$

- Can be shown by **Graphical Models**
  - Flexible modeling technique
  - Easy to understand
  - Know how to approximately compute

- Inference would related to clustering, dimension reduction, etc.



**N data samples, each has assignment  $z_i$**

# Probabilistic Graphical Models - Modeling Your Knowledge

- Events (random variables) - notation:  $(X, Y, Z)$ 
  - e.g. it rained, the street is wet, you are older than 23
  - they may affect each other
  - they may be (conditionally) independent
- We will use graphs to encode this information
  - event is a vertex; dependence is an edge
  - each variable corresponds to a node in the graph; links in the graph denote probabilistic relations between variable
- This leads to a “graphical model” that captures and expresses relations among variables
  - Graphical representation of a probabilistic model

# Probabilistic Graphical Models -

## Why do we need graphical models?

- Graphs are an intuitive way of representing and visualising the relationships between many variables. (Examples: family trees, electric circuit diagrams, neural networks)
- A graph allows us to abstract out the conditional independence relationships between the variables from the details of their parametric forms. Thus we can ask questions like: “Is A dependent on B given that we know the value of C ?” just by looking at the graph.
- Graphical models allow us to define general message-passing algorithms that implement Bayesian inference efficiently. Thus we can answer queries like “What is  $P(A|C = c)$ ?” without enumerating all settings of all variables in the model.

# Probabilistic Graphical Models - Some Notations First

- Random variables  $X, Y, Z$

Chain Rule

$$p(X, Y) = p(X|Y)p(Y)$$

$$\begin{aligned} p(X, Y, Z) &= p(X|Y, Z)p(Y, Z) \\ &= p(X|Y, Z)p(Y|Z)p(Z) \end{aligned}$$

Bayes' Theorem

$$p(X|Y) = \frac{p(X, Y)}{p(Y)} = \frac{p(Y|X)p(X)}{p(Y)}$$

Independence

$X$  and  $Y$  are independent if

$$p(X, Y) = p(X)p(Y)$$

since  $p(X, Y) = p(X|Y)p(Y)$   
provided  $p(X) \neq 0, p(Y) \neq 0$ , equivalent  
 $p(X|Y) = p(X) \leftrightarrow p(Y|X) = p(Y)$

Conditional independence

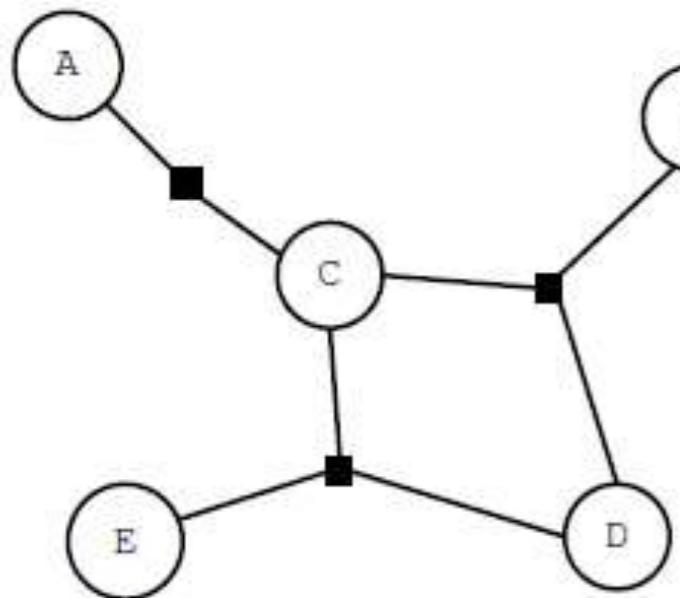
$\mathcal{X}$  and  $\mathcal{Y}$  are independent provided we know the state of  $\mathcal{Z}$  if  
 $p(\mathcal{X}, \mathcal{Y} | \mathcal{Z}) = p(\mathcal{X} | \mathcal{Z})p(\mathcal{Y} | \mathcal{Z})$  for all states of  $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ .

They are **conditionally independent** given  $\mathcal{Z}$

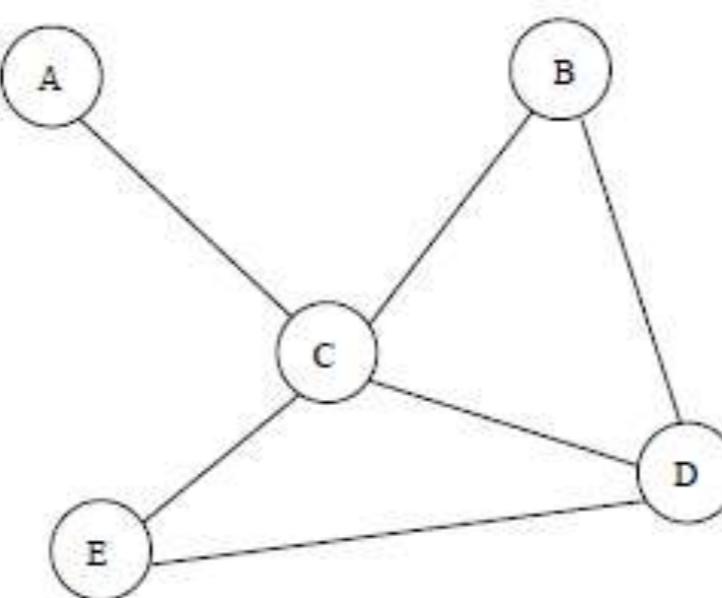
**conditional independence**  $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \mathcal{Z}$

**independence**  $\mathcal{X} \perp\!\!\!\perp \mathcal{Y} | \emptyset$  or  $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}$

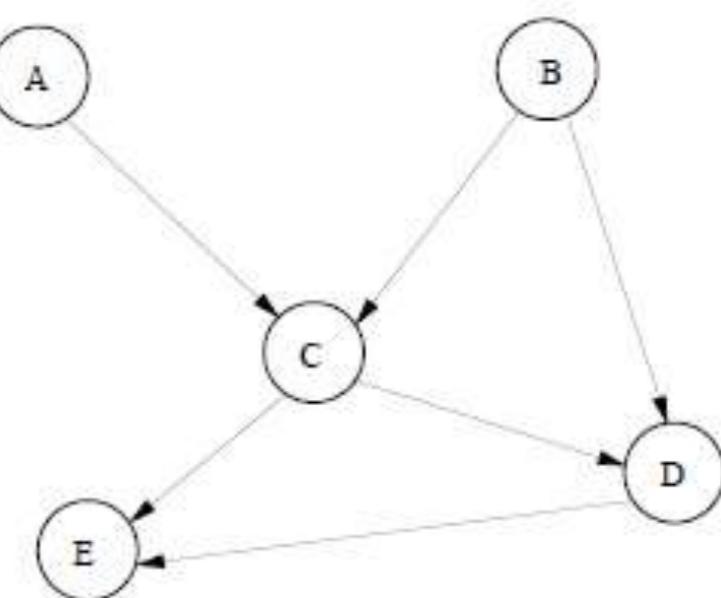
# Probabilistic Graphical Models - Three kinds of Graphical Models



factor graph



undirected graph  
or Markov random fields



directed graph  
or Bayesian Networks

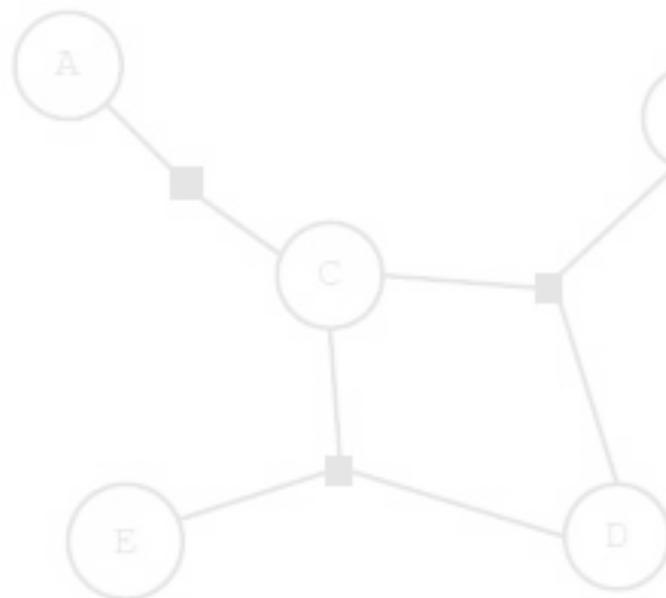
convenient for  
solving inference  
problems

useful to express soft  
constraints between  
variables

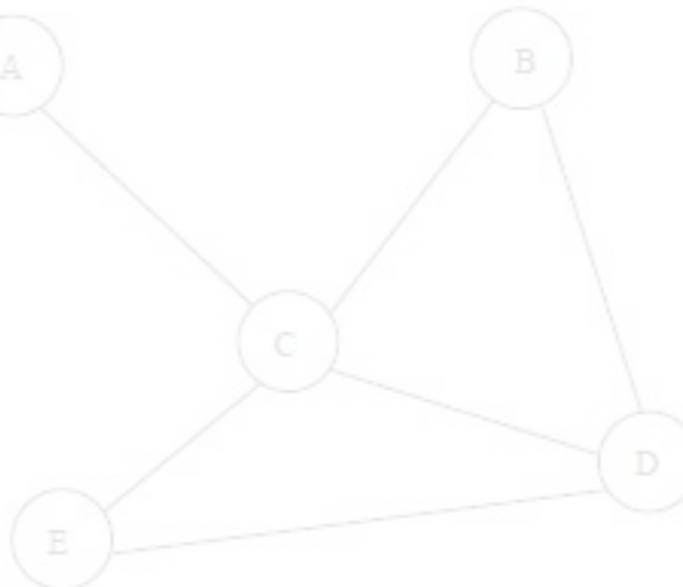
useful to express causal  
relationships between  
variables

**computing functions of the distribution,  
e.g. mean, mode, marginal, conditional**

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

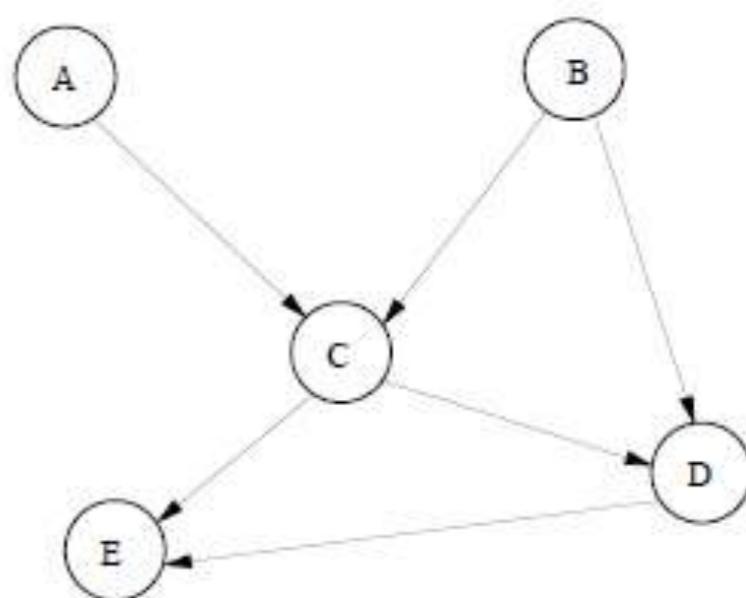


factor graph



undirected graph

or Markov random fields



directed graph

or Bayesian Networks

convenient for  
solving inference  
problems

useful to express soft  
constraints between  
variables

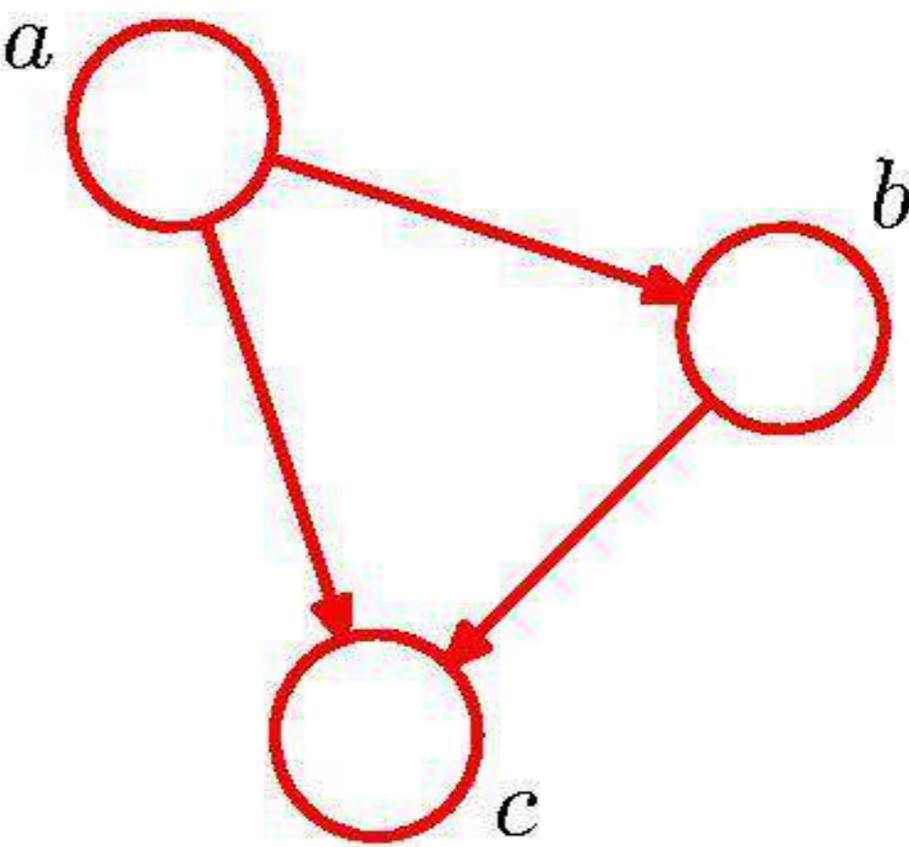
useful to express causal  
relationships between  
variables

computing functions of the distribution,  
e.g. mean, mode, marginal, conditional

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

## Directed Acyclic Graph (DAG)

by following a directed path  
of vertices, no path revisits a vertex



$$p(a, b, c) = p(c|a, b)p(a, b) = p(c|a, b)p(b|a)p(a)$$

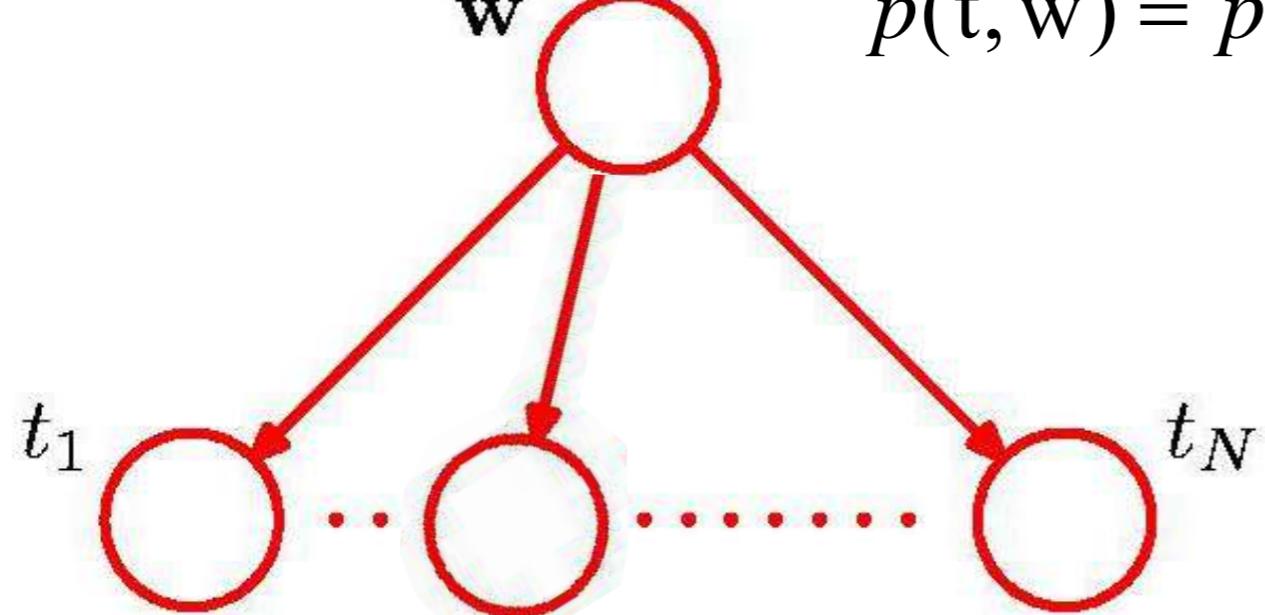
Generalization to  $D$  variables:

$$p(x_1, \dots, x_D) = p(x_D|x_1, \dots, x_{D-1}) \cdots p(x_2|x_1)p(x_1)$$

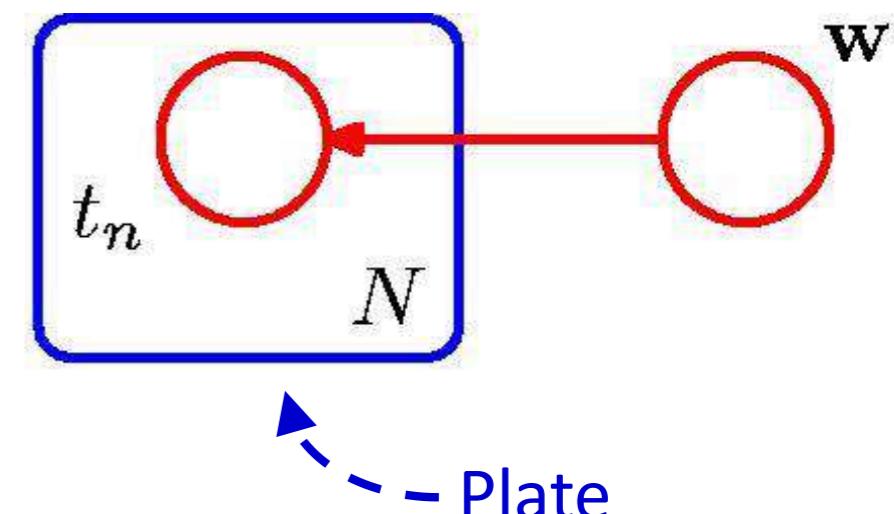
- The associated graph is fully connected.
- The absence of links conveys important information.

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

- Some notations:

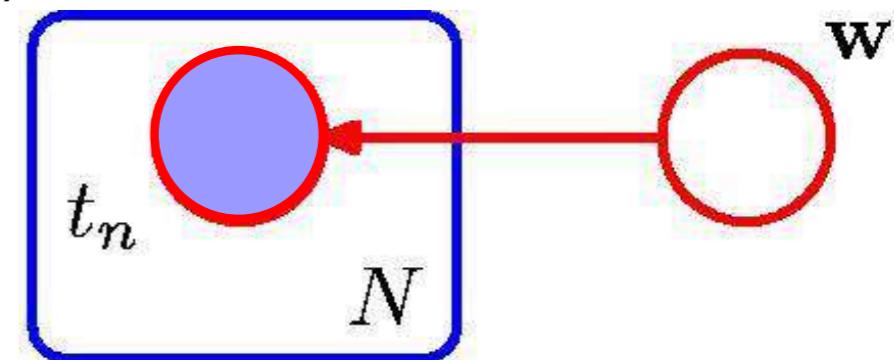


$$p(\mathbf{t}, \mathbf{w}) = p(\mathbf{w}) \prod_{n=1}^N p(t_n | \mathbf{w})$$

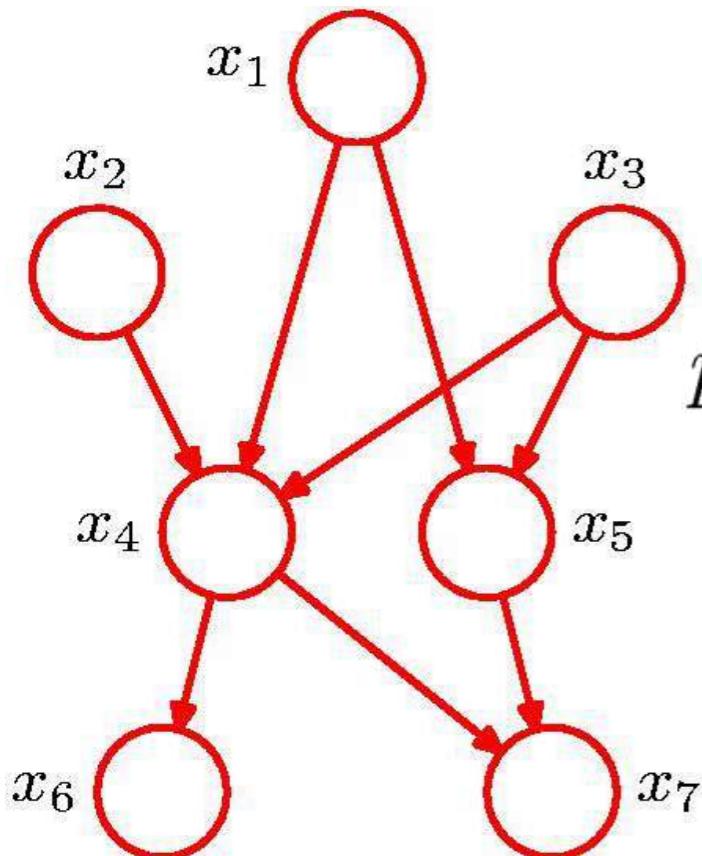


- Condition on data (shaded as observation):

$$p(\mathbf{w}|\mathbf{t}) \propto p(\mathbf{w}) \prod_{n=1}^N p(t_n|\mathbf{w})$$



# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks



$$p(x_1, \dots, x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3) \\ p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5)$$

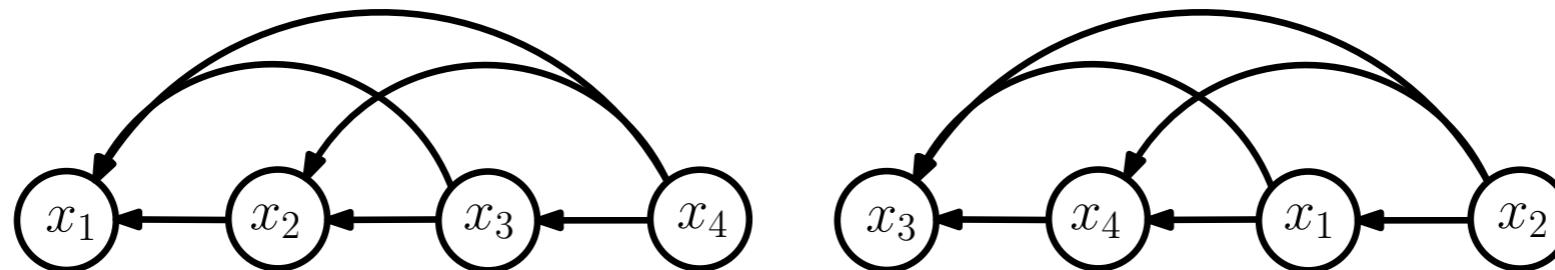
Belief network

A belief network is a distribution of the form

$$p(x_1, \dots, x_D) = \prod_{i=1}^D p(x_i | pa(x_i)),$$

where  $pa(x)$  denotes the parental variables of  $x$

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks



- ▶ Two factorizations of four variables:

$$\begin{aligned} p(x_1, x_2, x_3, x_4) &= p(x_1 \mid x_2, x_3, x_4)p(x_2 \mid x_3, x_4)p(x_3 \mid x_4)p(x_4) \\ p(x_1, x_2, x_3, x_4) &= p(x_3 \mid x_1, x_2, x_4)p(x_4 \mid x_1, x_2)p(x_1 \mid x_2)p(x_2) \end{aligned}$$

- ▶ Any distribution can be written in such a cascade form as a belief network (using chain rule)
- ▶ With independence assumptions the factorization often becomes simpler
- ▶ Structure of the DAG corresponds to a set of conditional independence assumptions
  - ▶ for completeness we need to specify all  $p(x \mid pa(x))$  **causality**

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

- ▶ Two factorizations of four variables:

$$p(x_1, x_2, x_3, x_4) = p(x_1 | x_2, x_3, x_4)p(x_2 | x_3, x_4)$$

- ▶ This does **not** mean non-parental variables have no influence:

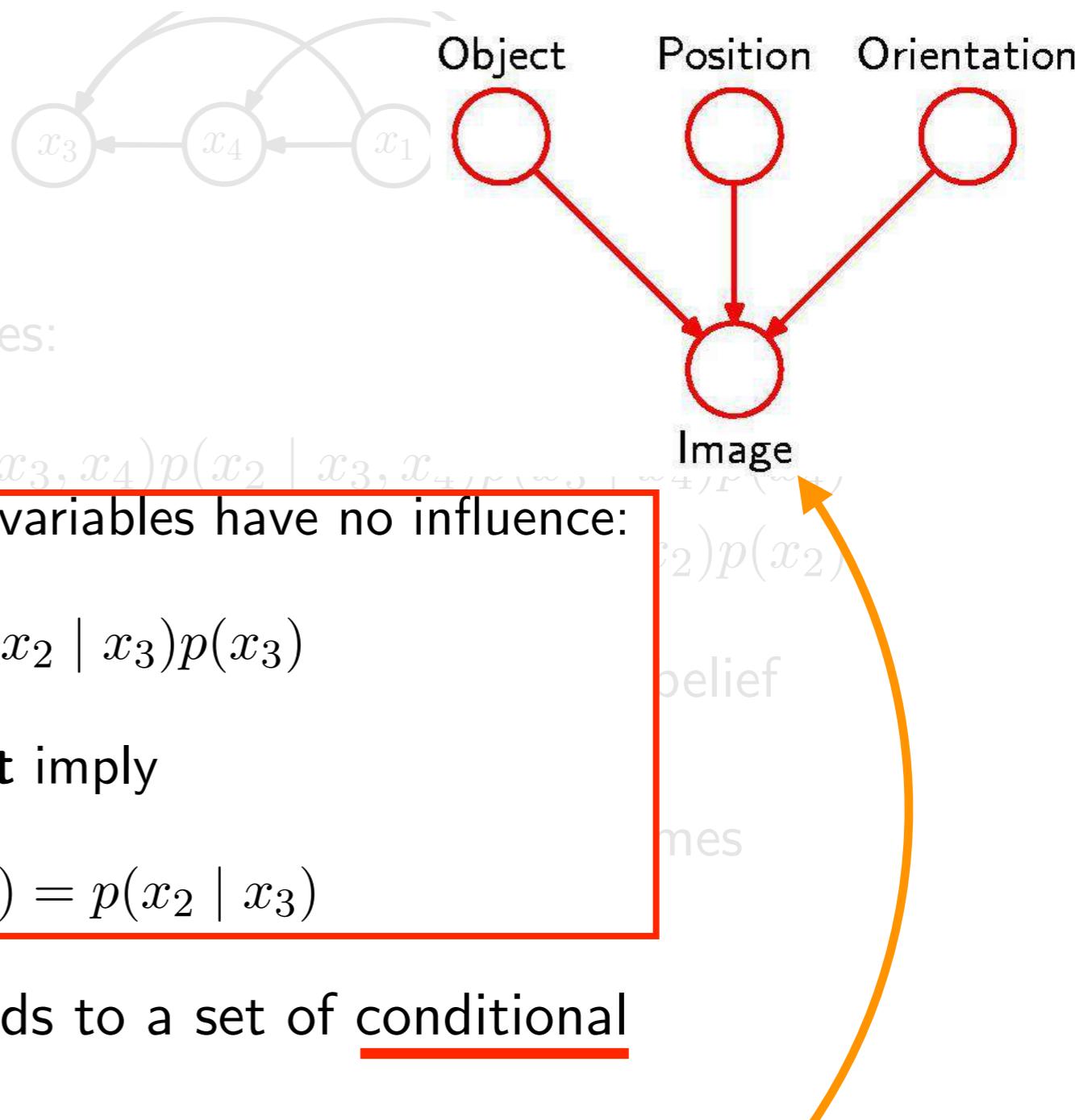
$$p(x_1 | x_2)p(x_2 | x_3)p(x_3)$$

with DAG  $x_1 \leftarrow x_2 \leftarrow x_3$  does **not** imply

$$p(x_2 | x_1, x_3) = p(x_2 | x_3)$$

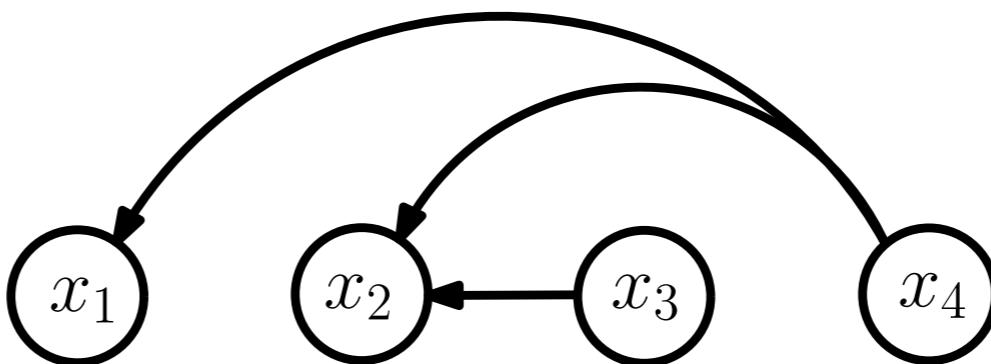
- ▶ Structure of the DAG corresponds to a set of conditional independence assumptions
  - ▶ for completeness we need to specify all  $p(x | pa(x))$  **causality**

Causal process for generating images



# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

- ▶ Important task:
  - ▶ given graph, read off conditional independence statements
- ▶ Question:
  - ▶ are  $x_1$  and  $x_2$  conditionally independent given  $x_4$  ( $x_1 \perp\!\!\!\perp x_2 | x_4$ )?
  - ▶ and what about  $x_1 \perp\!\!\!\perp x_2 | x_3$  ?



- ▶ how to **automate?**

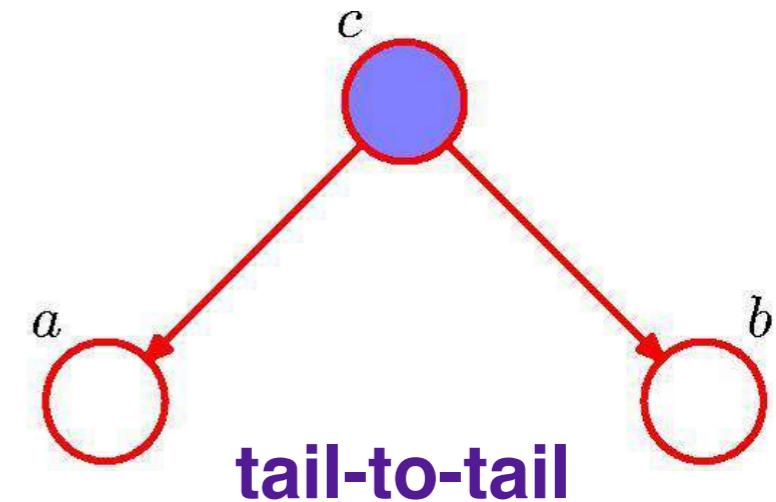
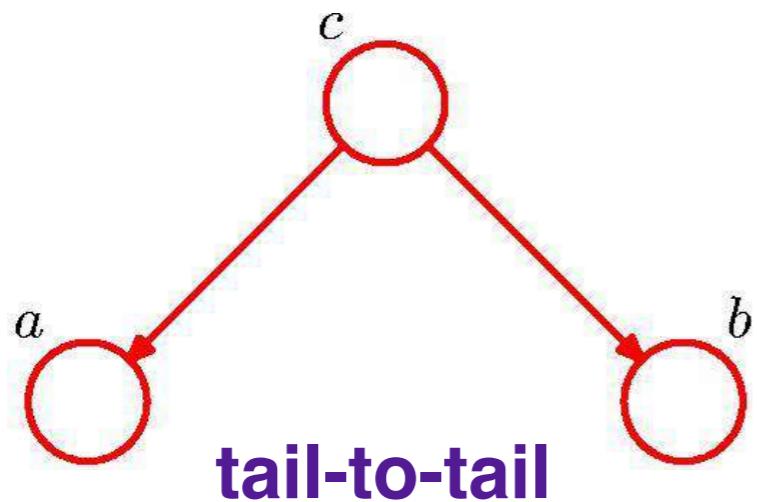
# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

$a$  independent of  $b$  given  $c$ :  $p(a|b, c) = p(a|c)$

equivalently:  $p(a, b|c) = p(a|b, c)p(b|c) = p(a|c)p(b|c)$

notation:  $a \perp\!\!\!\perp b | c$

## example#1



$$p(a, b, c) = p(a|c)p(b|c)p(c)$$

$\Rightarrow$  marginalize w.r.t.  $c$

$$\Rightarrow p(a, b) = \sum_c p(a|c)p(b|c)p(c)$$

$$\rightarrow a \perp\!\!\!\perp b | \emptyset$$

**as it doesn't factorize  
into  $p(a)p(b)$  in general**

$$p(a, b|c) = \frac{p(a, b, c)}{p(c)}$$

$$= \frac{p(a|c)p(b|c)p(c)}{p(c)}$$

$$= p(a|c)p(b|c)$$

$$\rightarrow a \perp\!\!\!\perp b | c$$

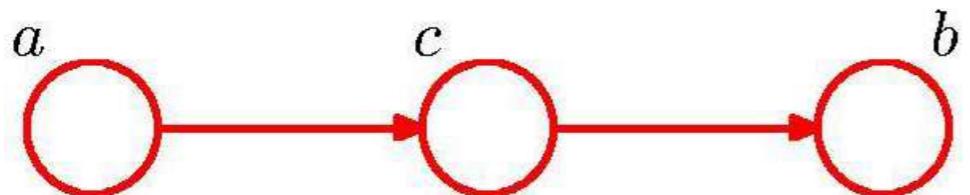
# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

$a$  independent of  $b$  given  $c$ :  $p(a|b, c) = p(a|c)$

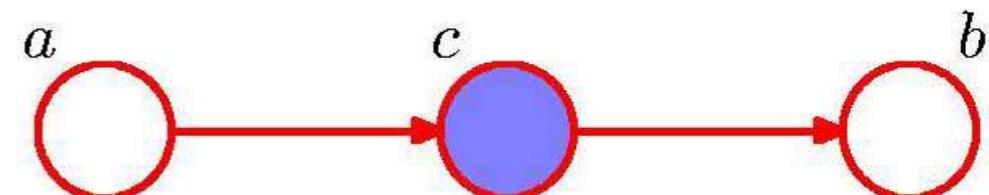
equivalently:  $p(a, b|c) = p(a|b, c)p(b|c) = p(a|c)p(b|c)$

notation:  $a \perp\!\!\!\perp b | c$

## example#2



**head-to-tail**



**head-to-tail**

$$p(a, b, c) = p(b|c)p(c|a)p(a)$$

$\Rightarrow$  marginalize w.r.t.  $c$

$$\Rightarrow p(a, b) = \sum_c p(a) \sum_c p(b|c)p(c|a)$$

$$\begin{aligned} p(a, b|c) &= \frac{p(a, b, c)}{p(c)} \\ &= \frac{p(b|c)p(c|a)p(a)}{p(c)} \\ &= \frac{p(b|c)p(c, a)}{p(c)} \\ &= p(b|c)p(a|c) \end{aligned}$$

  $a \perp\!\!\!\perp b | \emptyset$   
**as it doesn't factorize  
into  $p(a)p(b)$  in general**

  $a \perp\!\!\!\perp b | c$

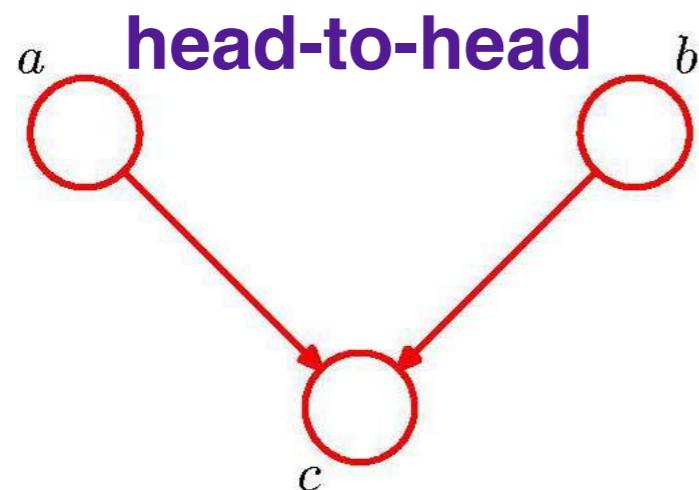
# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

$a$  independent of  $b$  given  $c$ :  $p(a|b, c) = p(a|c)$

equivalently:  $p(a, b|c) = p(a|b, c)p(b|c) = p(a|c)p(b|c)$

notation:  $a \perp\!\!\!\perp b | c$

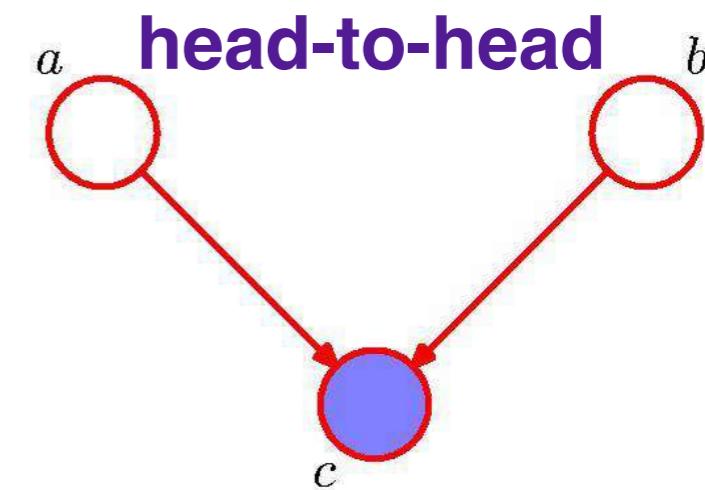
## example#3



$$p(a, b, c) = p(c|a, b)p(b)p(a)$$

$\Rightarrow$  marginalize w.r.t.  $c$

$$\Rightarrow p(a, b) = p(a)p(b)$$



$$p(a, b|c) = \frac{p(a, b, c)}{p(c)}$$

$$= \frac{p(c|a, b)p(b)p(a)}{p(c)}$$

→  $a \perp\!\!\!\perp b | 0$

opposite to example#1 and #2!

→  $a \not\perp\!\!\!\perp b | c$

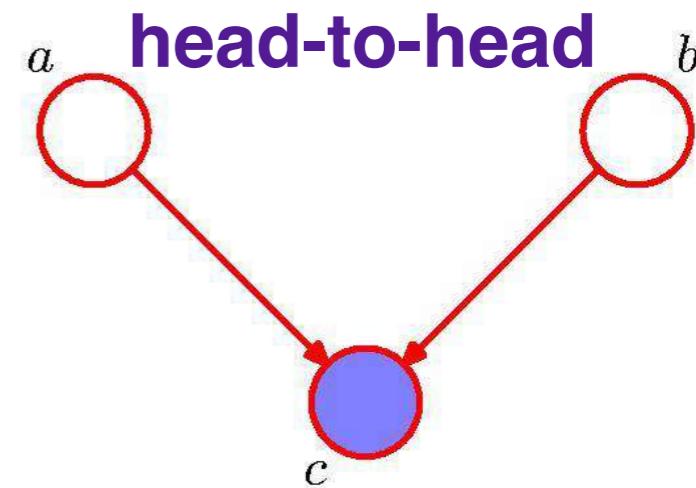
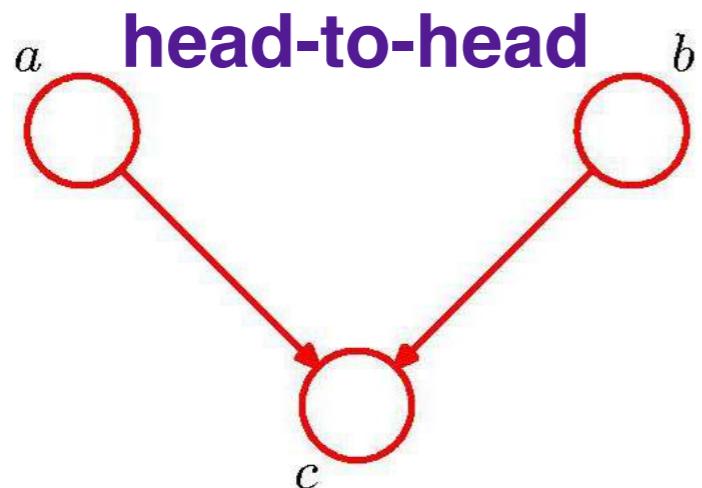
# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

$a$  independent of  $b$  given  $c$ :  $p(a|b, c) = p(a|c)$

equivalently:  $p(a, b|c) = p(a|b, c)p(b|c) = p(a|c)p(b|c)$

notation:  $a \perp\!\!\!\perp b | c$

## example#3

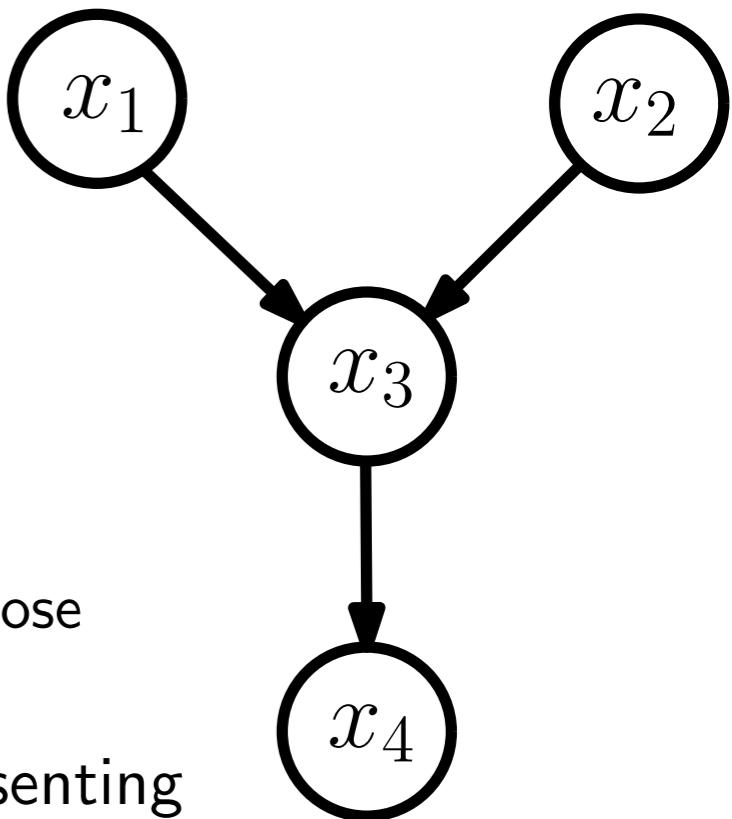


## Collision

Given a path from node  $x$  to  $y$ , a **collider** is a node  $c$  for which there are two nodes  $a, b$  in the path pointing *towards*  $c$ . ( $a \rightarrow c \leftarrow b$ )

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

- ▶  $x_3$  a collider ? yes ( $x_1 \rightarrow x_2$ ), no ( $x_1 \rightarrow x_4$ )
- ▶  $x_1 \perp\!\!\!\perp x_2 | x_3$  ? no
- ▶  $x_1 \perp\!\!\!\perp x_2 | x_4$  ? maybe
- ▶  $x_1$  and  $x_2$  are “graphically” dependent on  $x_4$ 
  - ▶ There are distributions with this DAG with  $x_1 \perp\!\!\!\perp x_2 | x_4$  and those with  $\underline{x_1 \perp\!\!\!\perp x_2 | x_4}$  **both possible**
- ▶ BN good for representing independence but not good for representing dependence!



## Collision

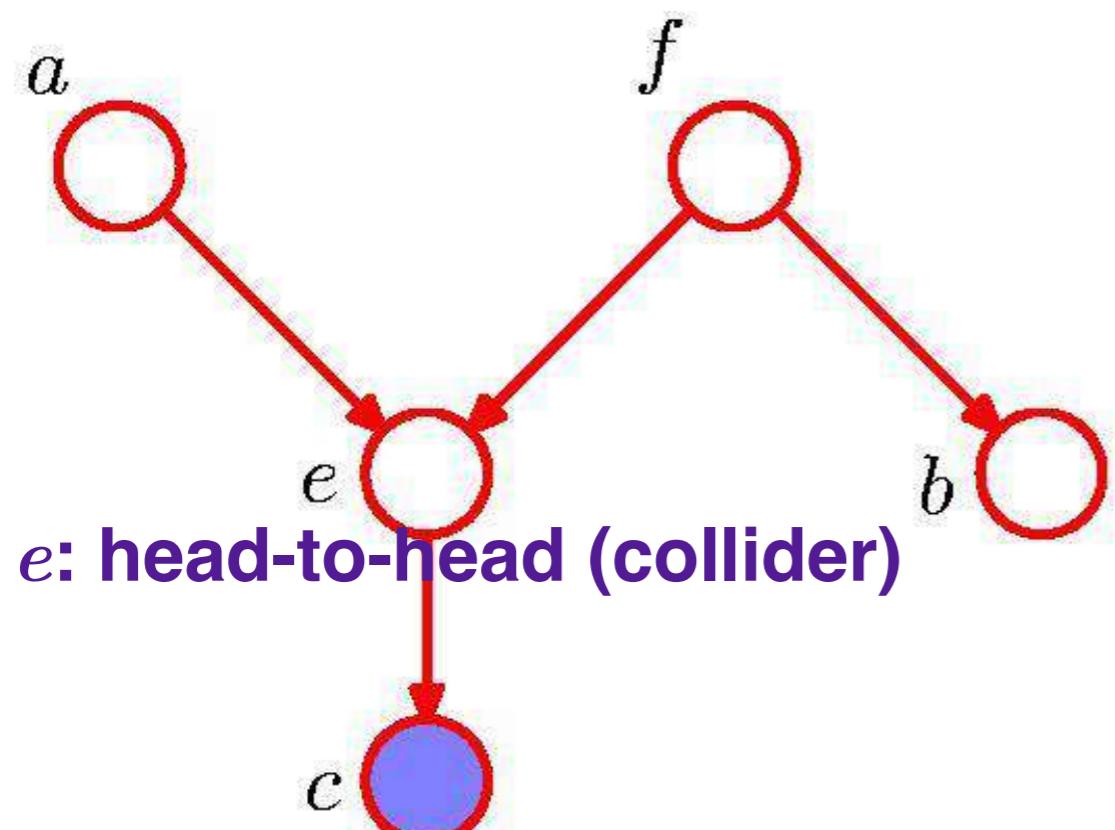
Given a path from node  $x$  to  $y$ , a **collider** is a node  $c$  for which there are two nodes  $a, b$  in the path pointing towards  $c$ . ( $a \rightarrow c \leftarrow b$ )

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

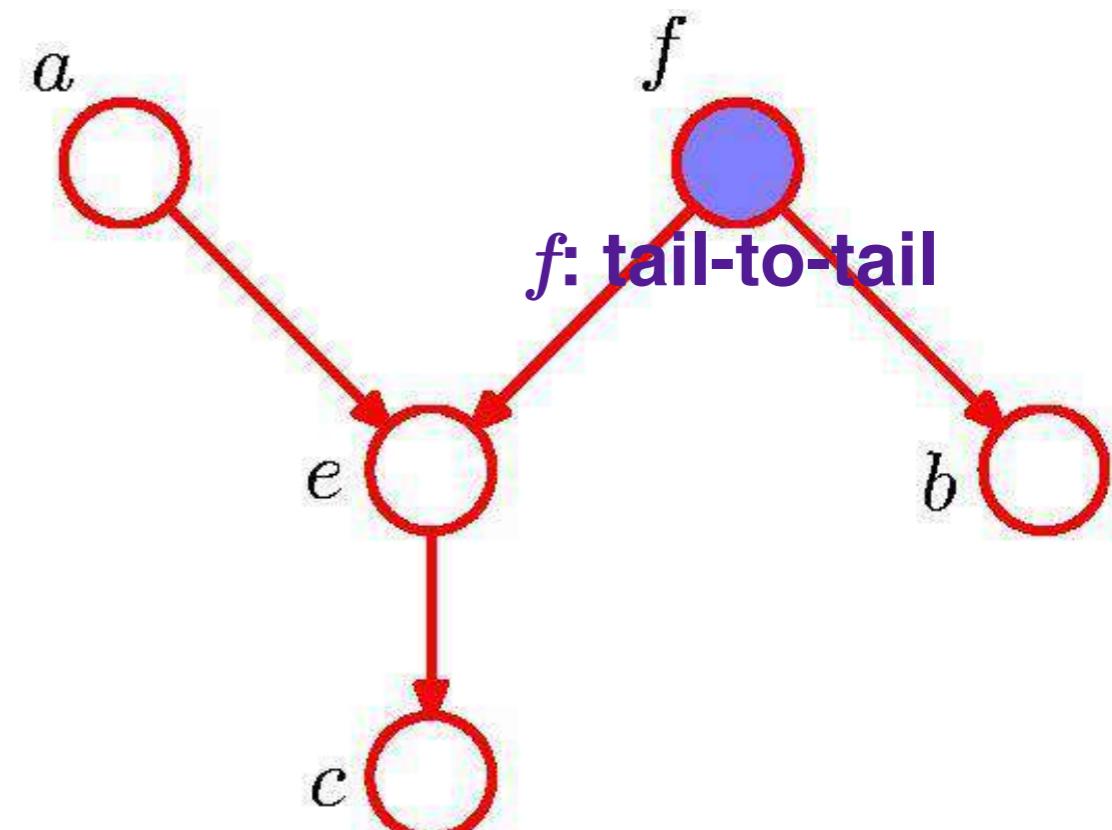
- **D-Separation** helps you easily answer question of independence!
  - $A, B$ , and  $C$  are non-intersecting subsets of nodes in a directed graph.
  - A path from  $A$  to  $B$  is blocked if it contains a node such that either
    - a) the arrows on the path meet either head-to-tail or tail-to-tail at the node, and the node is in the set  $C$ , or
    - b) the arrows meet head-to-head at the node, and neither the node, nor any of its descendants, are in the set  $C$ . **example#3:  $C$  or any of its descendant is not in set  $C$ !**
  - If all paths from  $A$  to  $B$  are blocked,  $A$  is said to be d-separated from  $B$  by  $C$ .
  - If  $A$  is d-separated from  $B$  by  $C$ , the joint distribution over all variables in the graph satisfies  $A \perp\!\!\!\perp B | C$ .

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

- **D-Separation** helps you easily answer question of independence!



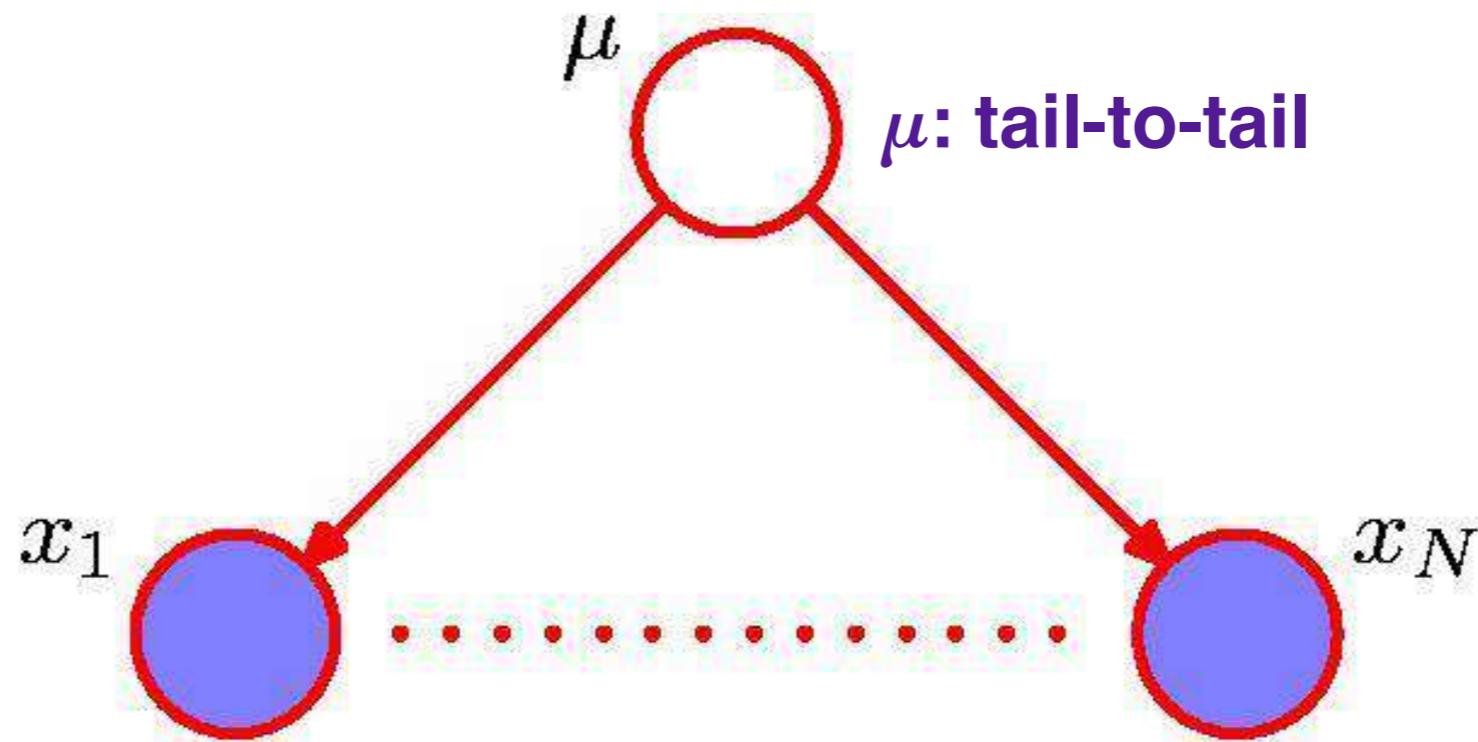
$$\rightarrow a \not\perp\!\!\!\perp b \mid c$$



$$\rightarrow a \perp\!\!\!\perp b \mid f$$

# Probabilistic Graphical Models - Directed Graph / Belief Networks / Bayesian Networks

- **D-Separation** helps you easily answer question of independence!

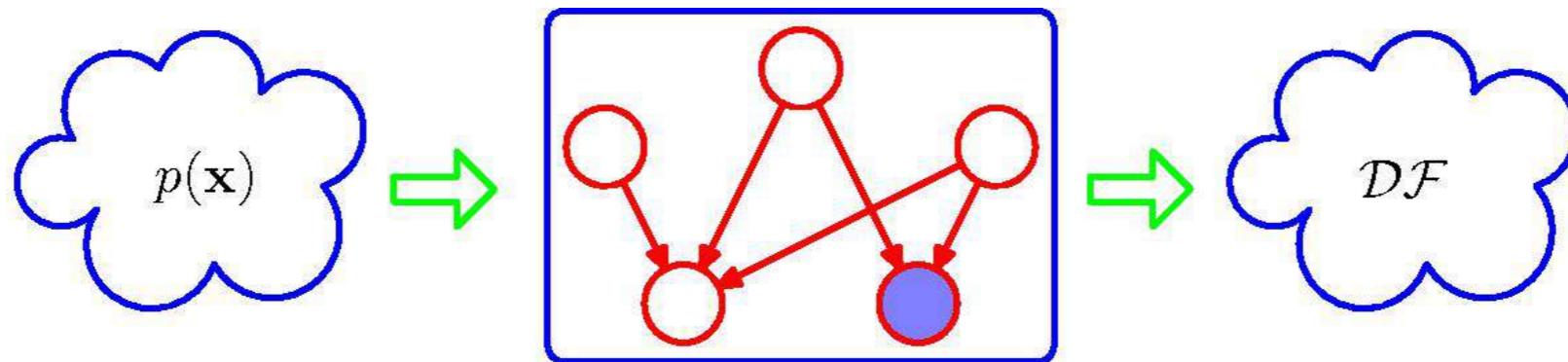


$$p(\mathcal{D}|\mu) = \prod_{n=1}^N p(x_n|\mu)$$

$x_n$  are independent of each other given  $\mu$   
i.i.d. (identical independent distributed) data!

# Probabilistic Graphical Models - Some Notations Again

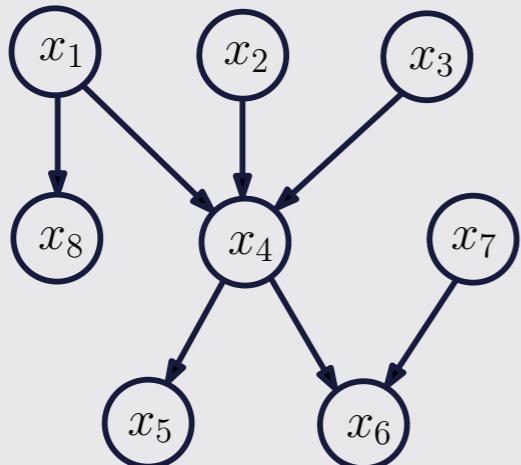
- Filter view of a graphical model



- ▶ Belief network (also undirected graph) implies a list of conditional independences
- ▶ Regard as filter:
  - ▶ only distributions that satisfy all conditional independences are allowed to pass
- ▶ One graph describes a whole family of probability distributions
- ▶ Extremes:
  - ▶ Fully connected, no constraints, all  $p$  pass
  - ▶ no connections, only product of marginals may pass

# Probabilistic Graphical Models - Some Notations Again

## The Family



The **parents** of  $x_4$  are  $pa(x_4) = \{x_1, x_2, x_3\}$ . The **children** of  $x_4$  are  $ch(x_4) = \{x_5, x_6\}$ . The **family** of  $x_4$  are the node itself, its parents and children. The **Markov blanket** is the node, its parents, the children and the parents of the children. In this case  $x_1, \dots, x_7$

## Path, Ancestor, Descendant

- A **path**  $A \rightarrow B$  is a sequence of vertices

$$A_0 = A, A_1, \dots, A_{N-1}, A_N = B \quad (12)$$

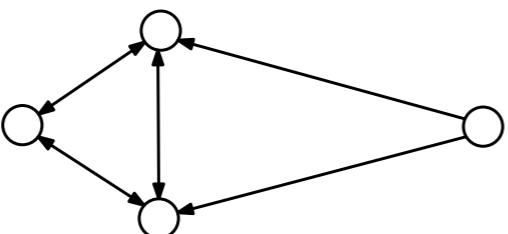
with  $(A_n, A_{n+1})$  an edge in the graph.

- In directed graphs, the vertices  $A$  such that  $A \rightarrow B$  and  $B \not\rightarrow A$  are the **ancestors** of  $B$ .
- Vertices  $B$  such that  $A \rightarrow B$  and  $B \not\rightarrow A$  are the **descendants** of  $A$ .

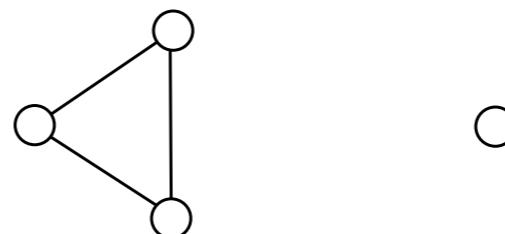
# Probabilistic Graphical Models - Some Notations Again

## Connected Graph

A graph is **connected** if there is a path between any two vertices.  
Otherwise there are **connected components**.



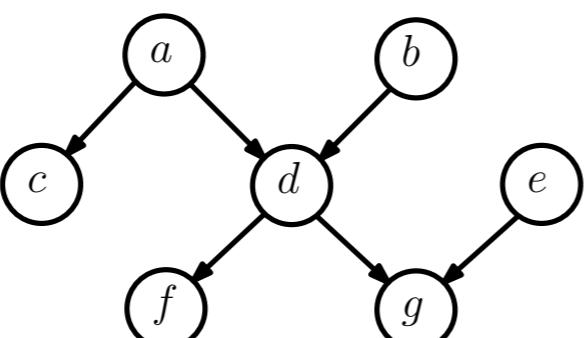
connected graph



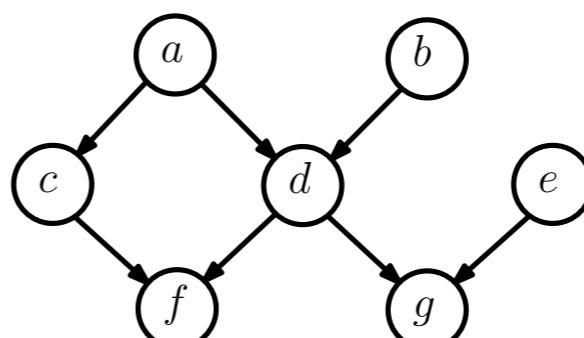
graph with  
two connected components

## Singly- and Multiply Connected

A graph is **singly-connected** if for any vertex  $a$  and  $b$  there exists not more than one path between them. Otherwise it is **multiply-connected**. Another name for a singly-connected graph is a **tree**. A multiply connected graph is also called **loopy**.



singly-connected

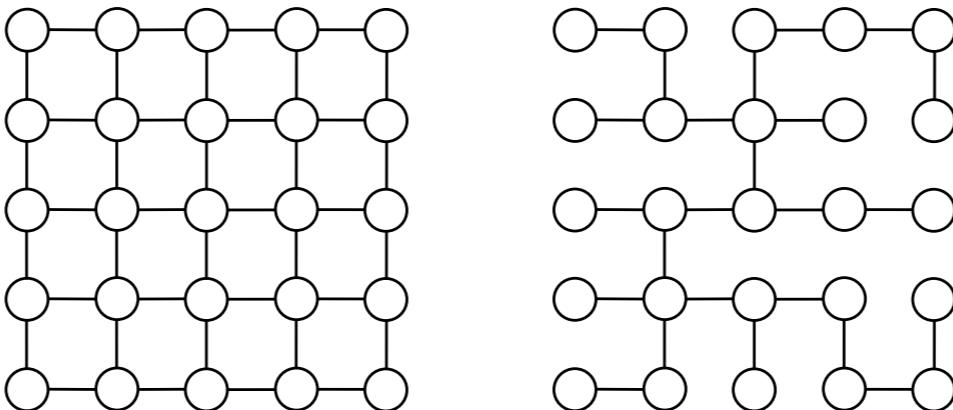


multiply-connected - Prof. Bernt Schiele

# Probabilistic Graphical Models - Some Notations Again

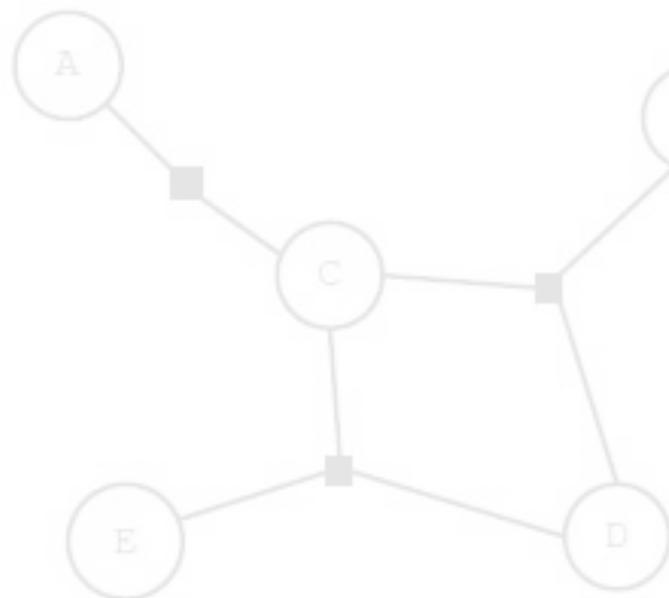
## Spanning Tree

A **spanning tree** of an undirected graph  $G$  is a singly-connected subset of the existing edges such that the resulting singly-connected graph covers all vertices of  $G$ . A **maximum (weight) spanning tree** is a spanning tree such that the sum of all weights on the edges is larger than for any other spanning tree of  $G$ .

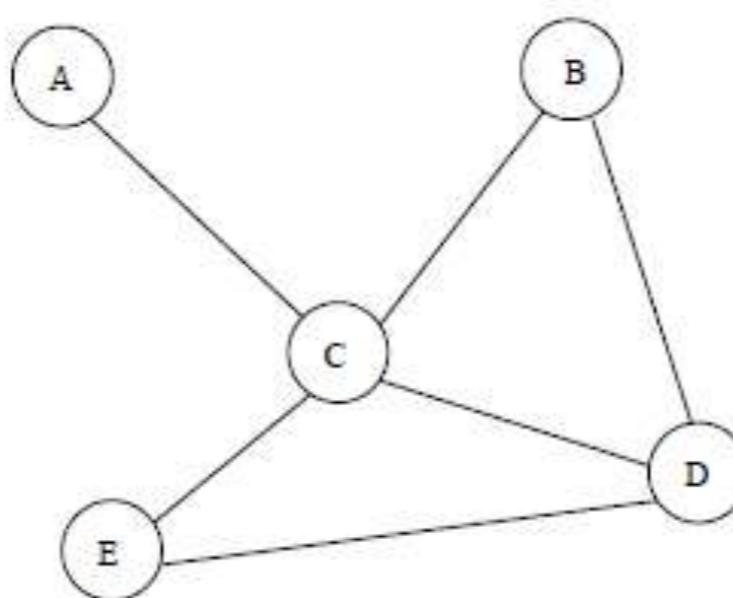


- ▶ There might be more than one maximum spanning tree.

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields



factor graph



undirected graph

or Markov random fields



directed graph

or Bayesian Networks

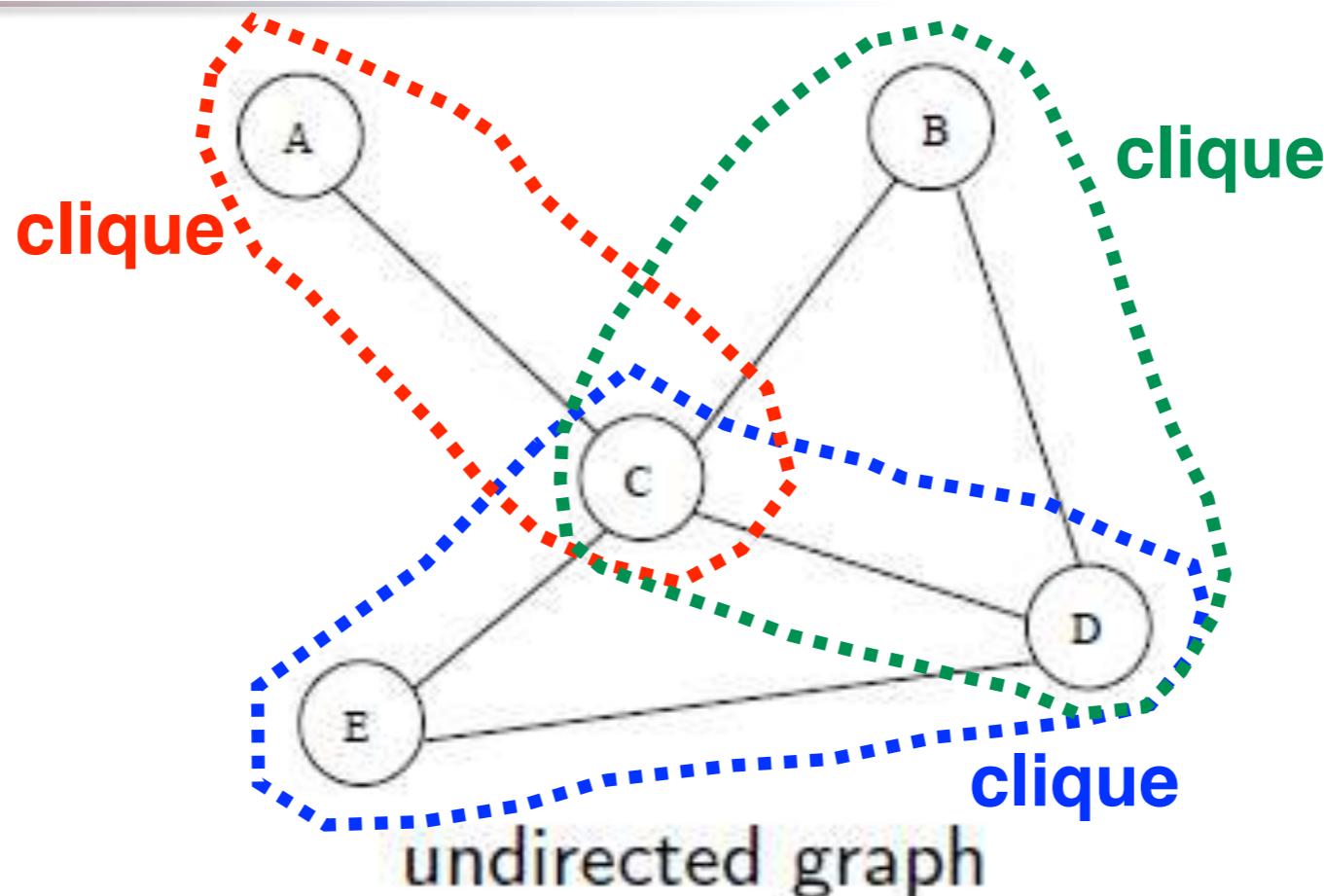
convenient for  
solving inference  
problems

useful to express soft  
constraints between  
variables

useful to express causal  
relationships between  
variables

computing functions of the distribution,  
e.g. mean, mode, marginal, conditional

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields



## Neighbour

In an undirected graph a **neighbour** of  $x$  are all vertices that share an edge with  $x$ .

## Clique

Given an undirected graph a **clique** is a subset of fully connected vertices. All members of the clique are neighbours, there is no larger clique that contains the clique. **maximal clique: clique that cannot be enlarged**

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields

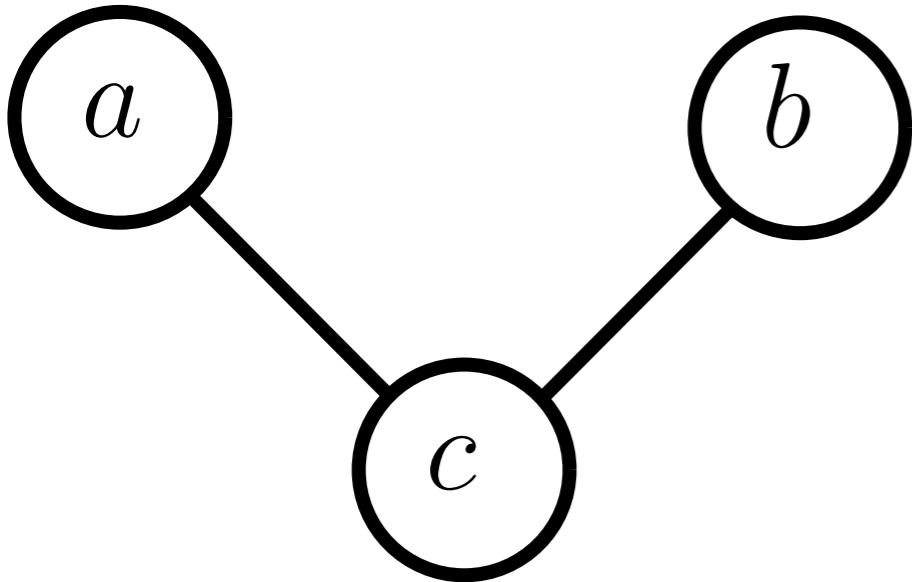
1. Joint distribution can be factorized in terms of maximal cliques (later)  
(In *modeling* cliques describe variables that all depend on each other)
2. Functions defined on maximal cliques include subset of maximal cliques  
(In *inference* they describe sets of variables with no simpler structure to describe their relationships.  
For instance, if  $\{x_1, x_2, x_3\}$  is a maximal clique and we define an arbitrary function over this clique, then including another factor defined over a “subset” of these variables would be “redundant”!)



## Clique

Given an undirected graph a **clique** is a subset of fully connected vertices. All members of the clique are neighbours, there is no larger clique that contains the clique. **maximal clique: clique that cannot be enlarged**

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields



$$p(a, b, c) = \frac{1}{Z} \phi_{ac}(a, c) \phi_{bc}(b, c)$$



**functions over cliques**

Here  $Z$  normalization constant or **partition function**

$$Z = \sum_{a,b,c} \phi_{ac}(a, c) \phi_{bc}(b, c)$$

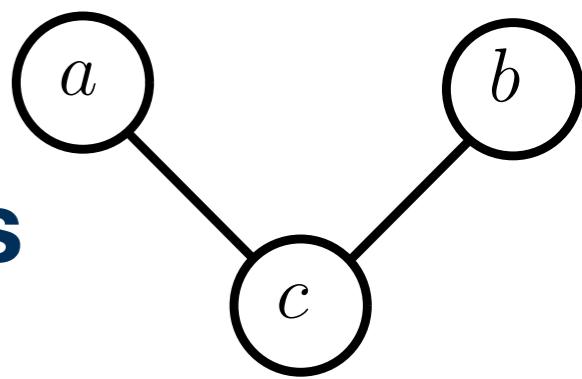
***potential: measures “compatibility” between settings of the variables***

## Potential

A **potential**  $\phi(x)$  is a non-negative function of the variable  $x$ . A **joint potential**  $\phi(x_1, \dots, x_D)$  is a non-negative function of the set of variables.

- Distribution (as in belief networks) is a special choice

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields



$$p(a, b, c) = \frac{1}{Z} \phi_{ac}(a, c) \phi_{bc}(b, c)$$

## Markov Network

For a set of variables  $\mathcal{X} = \{x_1, \dots, x_D\}$  a **Markov network** is defined as a product of potentials over the maximal cliques  $\mathcal{X}_c$  of the graph  $\mathcal{G}$

$$p(x_1, \dots, x_D) = \frac{1}{Z} \prod_{c=1}^C \phi_c(\mathcal{X}_c) \quad (16)$$

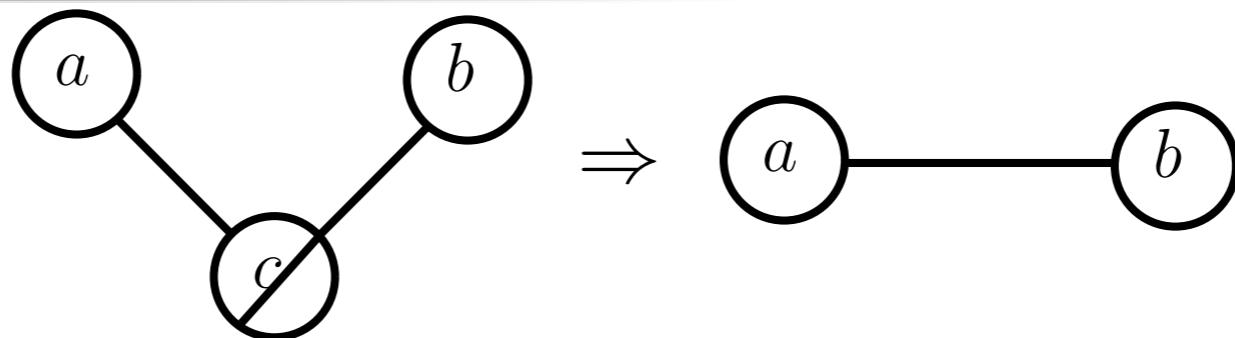
- ▶ Special case: cliques of size 2 – **pairwise Markov network**
- ▶ In case all potentials are strictly positive this is called a Gibbs distribution

$\phi_c(\mathcal{X}_c) = \exp(-E(\mathcal{X}_c))$  where  $E(\mathcal{X}_c)$  is called **energy function**  
and this exponential representation is called Boltzmann distribution

same  
thing  
actually

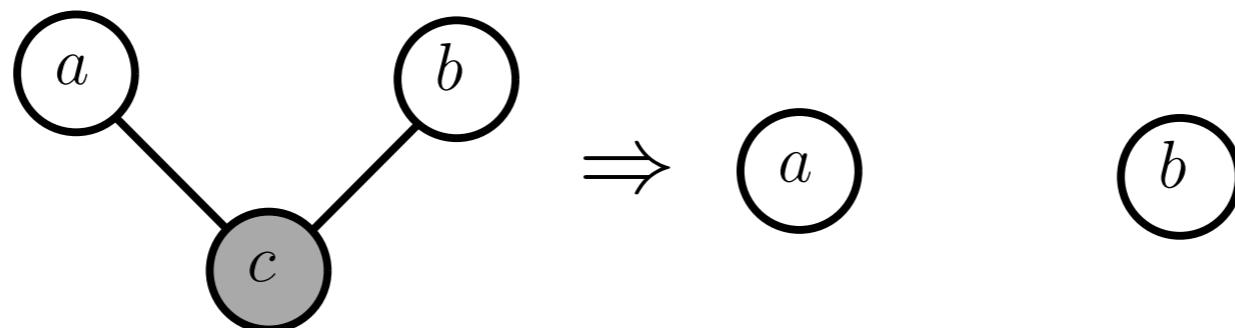
# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields

## properties of Markov Network



- ▶ Marginalizing over *c* makes *a* and *b* “graphically” dependent

$$p(a, b) = \sum_c \frac{1}{Z} \phi_{ac}(a, c) \phi_{bc}(b, c) = \frac{1}{Z} \phi_{ab}(a, b)$$



- ▶ Conditioning on *c* makes *a* and *b* independent

$$p(a, b | c) = p(a | c)p(b | c)$$

- ▶ This is opposite to the directed version  $a \rightarrow c \leftarrow b$  where conditioning introduced dependency

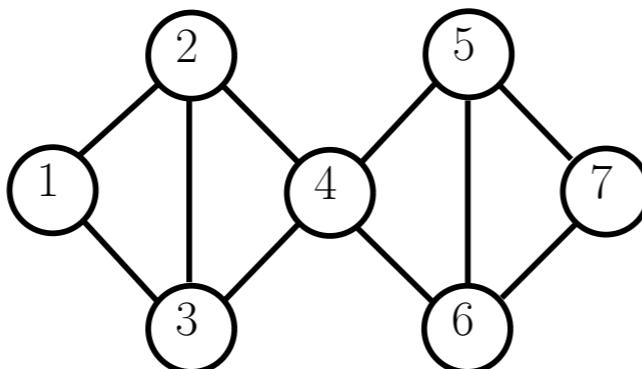
# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields

## properties of Markov Network

### Local Markov Property

$$p(x \mid \mathcal{X} \setminus \{x\}) = p(x \mid ne(x))$$

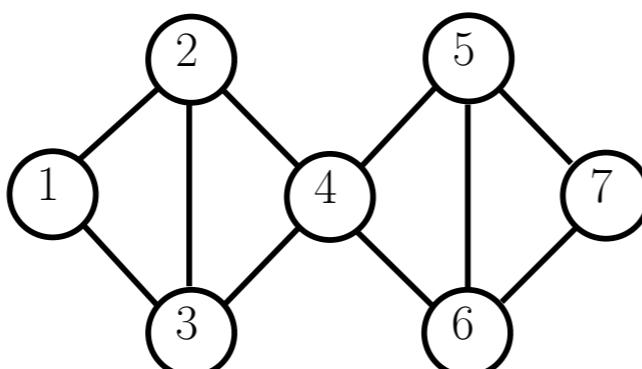
- ▶ Condition on neighbours independent on rest



$$x_4 \perp\!\!\!\perp \{x_1, x_7\} \mid \{x_2, x_3, x_5, x_6\}$$

### Global Markov Property

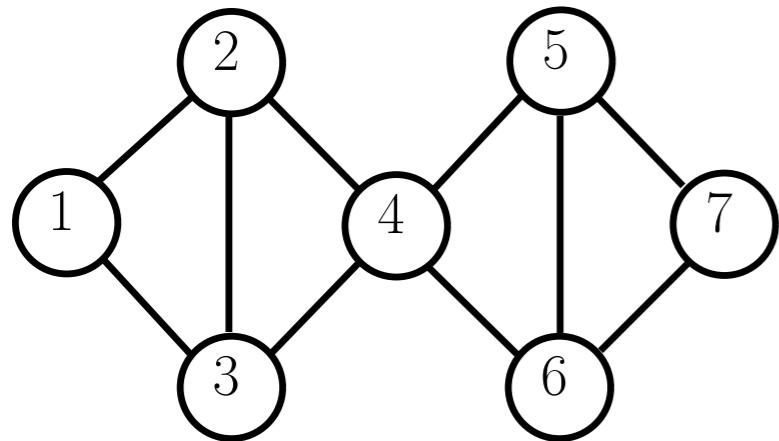
For disjoint sets of variables  $(\mathcal{A}, \mathcal{B}, \mathcal{S})$  where  $\mathcal{S}$  separates  $\mathcal{A}$  from  $\mathcal{B}$ , then  
 $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \mid \mathcal{S}$



$$x_1 \perp\!\!\!\perp x_7 \mid \{x_4\}$$

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields

- Finding the factorization



- ▶ Eliminate variable one by one

- ▶ Let's start with  $x_1$

$$p(x_1, \dots, x_7) = p(x_1 | x_2, x_3)p(x_2, \dots, x_7)$$

- ▶ Graph specifies:

$$p(x_1, x_2, x_3 | x_4, \dots, x_7) = p(x_1, x_2, x_3 | x_4)$$

$$\Rightarrow p(x_2, x_3 | x_4, \dots, x_7) = p(x_2, x_3 | x_4)$$

- ▶ Hence

$$p(x_1, \dots, x_7) = p(x_1 | x_2, x_3)p(x_2, x_3 | x_4)p(x_4, x_5, x_6, x_7)$$

- ▶ We continue to find

$$p(x_1, \dots, x_7) = p(x_1 | x_2, x_3)p(x_2, x_3 | x_4)$$

$$p(x_4 | x_5, x_6)p(x_5, x_6 | x_7)p(x_7)$$

- ▶ A factorization into clique potentials (maximal cliques)

$$p(x_1, \dots, x_7) = \frac{1}{Z} \phi(x_1, x_2, x_3)\phi(x_2, x_3, x_4)\phi(x_4, x_5, x_6)\phi(x_5, x_6, x_7)$$



**Markov conditions of graph  $G \Rightarrow$   
factorization  $F$  of clique potentials;  
and conversely:  $F \Rightarrow G$**

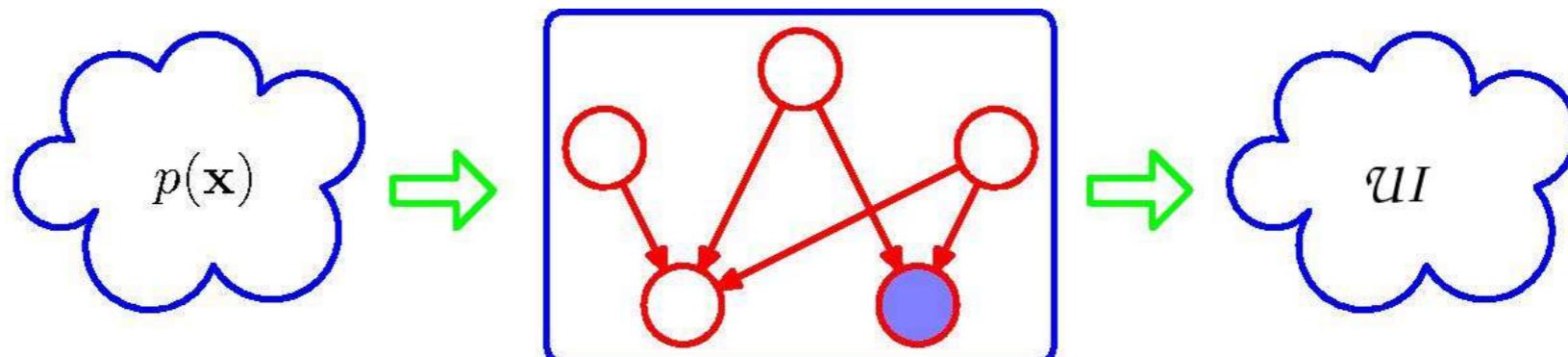
# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields

## Hammersley-Clifford

This factorization property  $G \Leftrightarrow F$  holds for any undirected graph provided that the potentials are positive

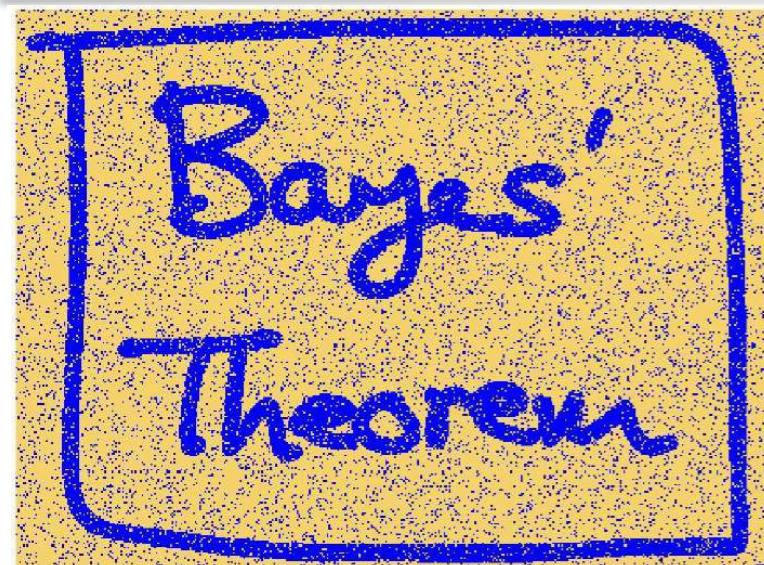
- ▶ Thus also loopy ones:  $x_1 - x_2 - x_3 - x_4 - x_1$
- ▶ Theorem says, distribution is of the form

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \phi_{12}(x_1, x_2) \phi_{23}(x_2, x_3) \phi_{34}(x_3, x_4) \phi_{41}(x_4, x_1)$$



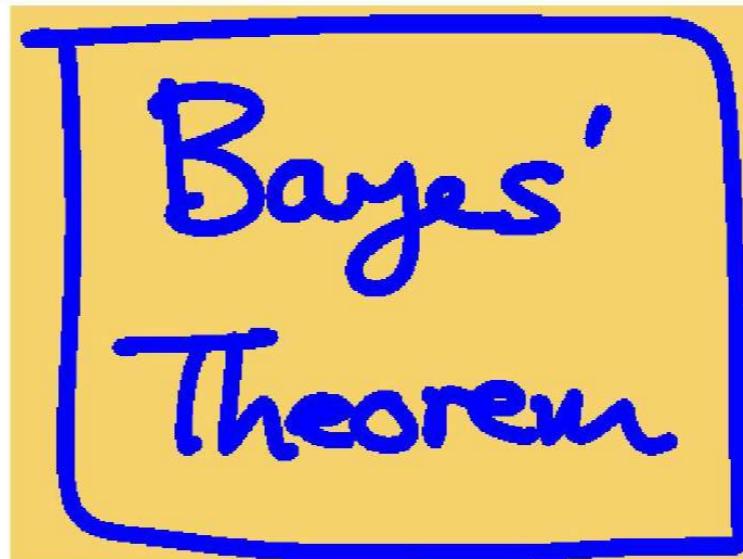
- ▶ Let  $\mathcal{UI}$  denote the distributions that can pass
  - ▶ those that satisfy all conditional independence statements
- ▶ Let  $\mathcal{UF}$  denote the distributions with factorization over cliques
- ▶ Hammersley-Clifford says :  $\mathcal{UI} = \mathcal{UF}$

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields



Noisy Image

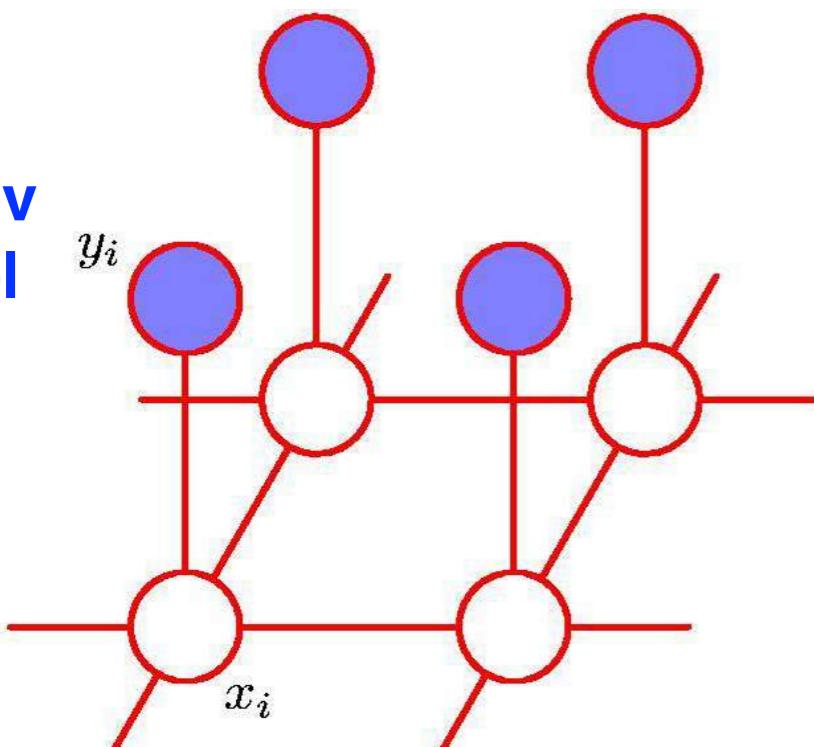
→  
de-noise  
hopefully



Original Image

- ▶ Noisy Image:  $y_i \in \{-1, +1\}$  where  $i$  runs over all the pixels
- ▶ Unknown Noise Free Image:  $x_i \in \{-1, +1\}$  **(2 colors only)**

Markov  
model



- strong correlation between  $x_i$  and  $y_i$ .
- neighbouring pixels  $x_i$  and  $x_j$  in an image are strongly correlated.

# Probabilistic Graphical Models - Undirected Graph / Markov Random Fields

Two types of cliques **(energy minimization)**

- ▶  $-\eta x_i y_i$ : giving a lower energy when  $x_i$  and  $y_i$  have the same sign and a higher energy when they have the opposite sign
- ▶  $-\beta x_i x_j$ : the energy is lower when the neighboring pixels have the same sign than when they have the opposite sign

$$E(\mathbf{x}, \mathbf{y}) = h \sum_i x_i - \beta \sum_{\{i,j\}} x_i x_j \quad \text{Ising model}$$

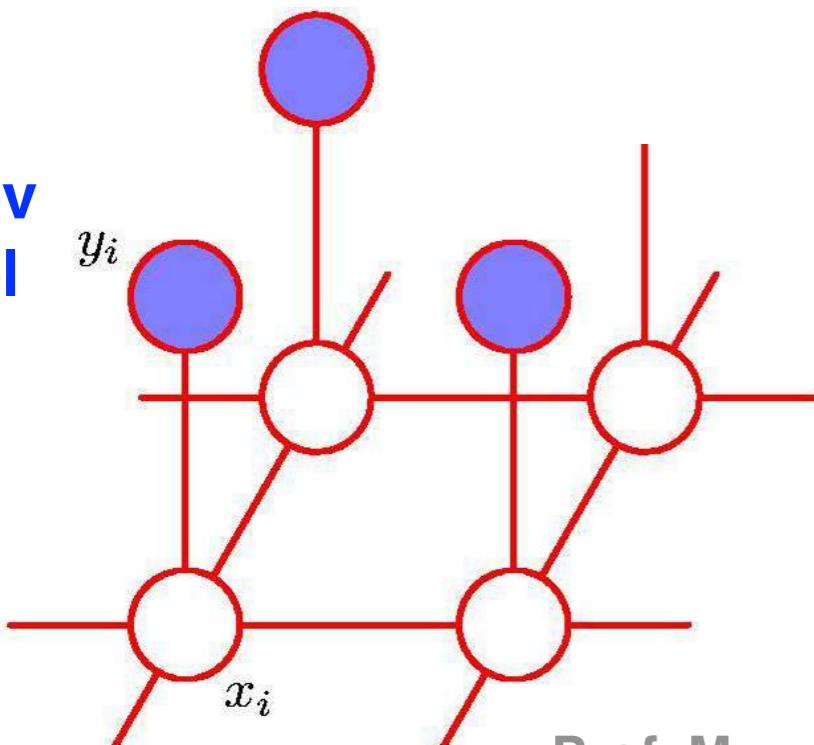
$-\eta \sum_i x_i y_i$  **biasing model towards values that have one particular sign in preference to the other**

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{y})\}$$

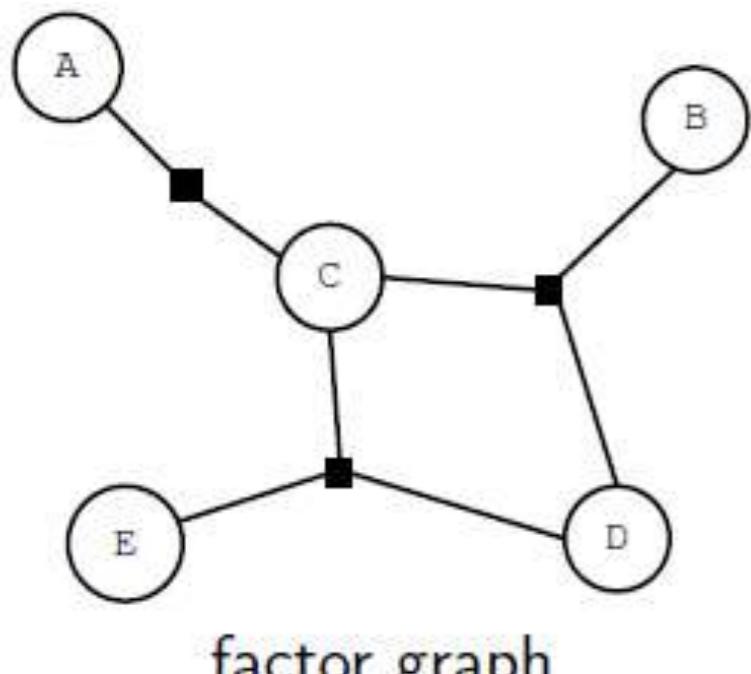
- initialize  $\mathbf{x}$  to  $\mathbf{y}$
- until convergence:

for each  $x_i$ :

$$x_i \leftarrow \operatorname{argmin}_{x_i} E(x_i, y_i)$$



# Probabilistic Graphical Models - Factor Graph



convenient for  
solving inference  
problems



**computing functions of the distribution,  
e.g. mean, mode, marginal, conditional**

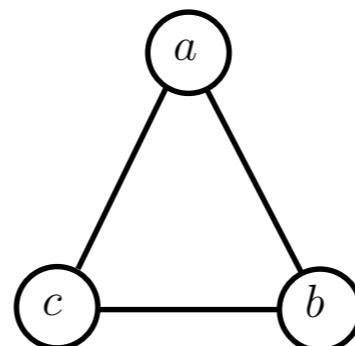
# Probabilistic Graphical Models - Factor Graph

## Relationship Potentials to Graphs

consider a factorization

$$p(a, b, c) = \frac{1}{Z} \phi(a, b) \phi(b, c) \phi(c, a)$$

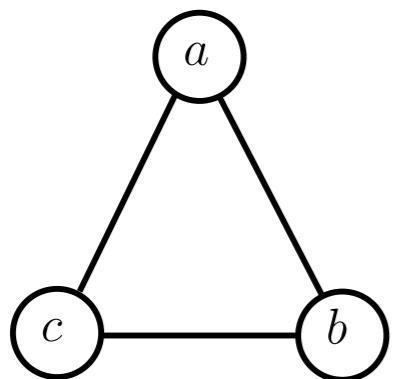
corresponding  
Markov network



other factorization  
represented by  
the same network

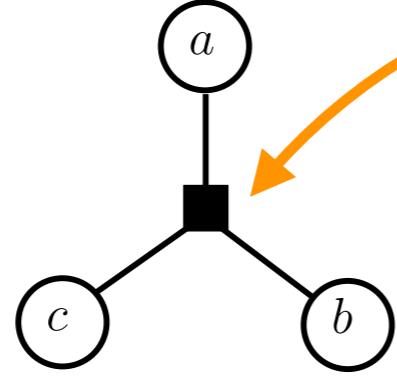
$$p(a, b, c) = \frac{1}{Z} \phi(a, b, c)$$

the factorization is not specified by the graph  $\Rightarrow$  the basic motivation of *Factor Graph*!



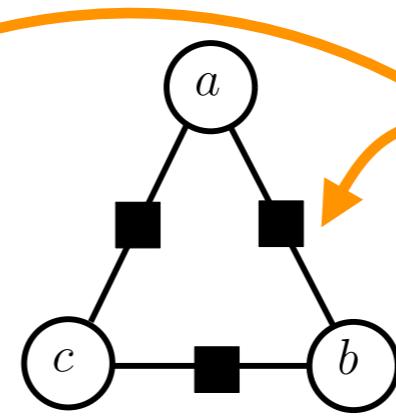
(1)

Markov  
network



(2)

factor graph  
representation of  
 $\phi(a, b, c)$



(3)

factor graph  
representation of  
 $\phi(a, b) \phi(b, c) \phi(c, a)$

we introduce an  
extra node (square)  
for each factor!



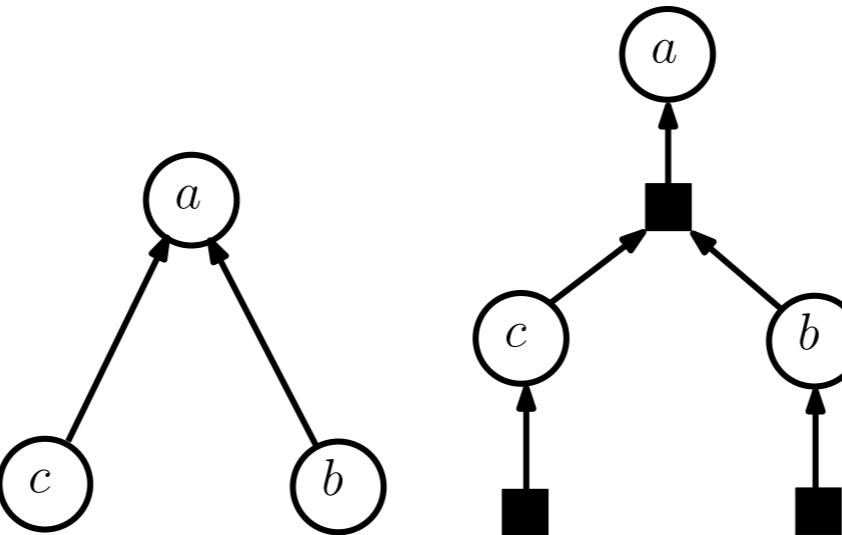
different factor graphs  
can have the same  
Markov network



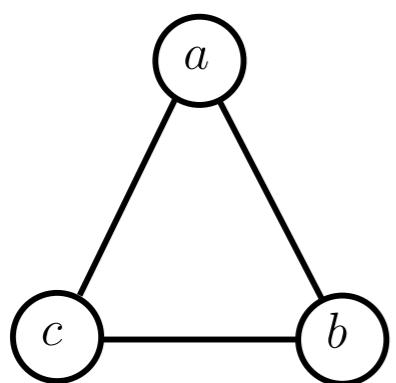
prefer to use  
*Factor Graph*!

# Probabilistic Graphical Models - Factor Graph

## Relationship Potentials to Graphs

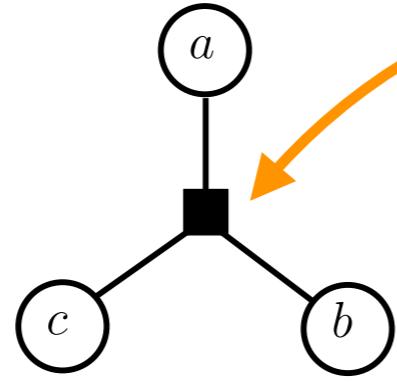


**Factor Graph** can be used for belief network as well



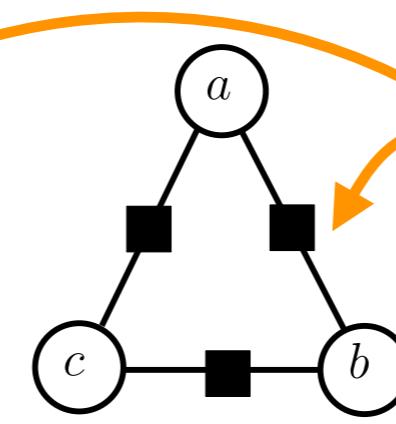
(1)

Markov  
network



(2)

factor graph  
representation of  
 $\phi(a, b, c)$



(3)

factor graph  
representation of  
 $\phi(a, b)\phi(b, c)\phi(c, a)$

we introduce an extra node (square) for each factor!



different factor graphs can have the same Markov network



prefer to use **Factor Graph!**

# Probabilistic Graphical Models - Factor Graph

## Factor Graph

Given a function

$$f(x_1, \dots, x_n) = \prod_i \psi_i(\mathcal{X}_i),$$

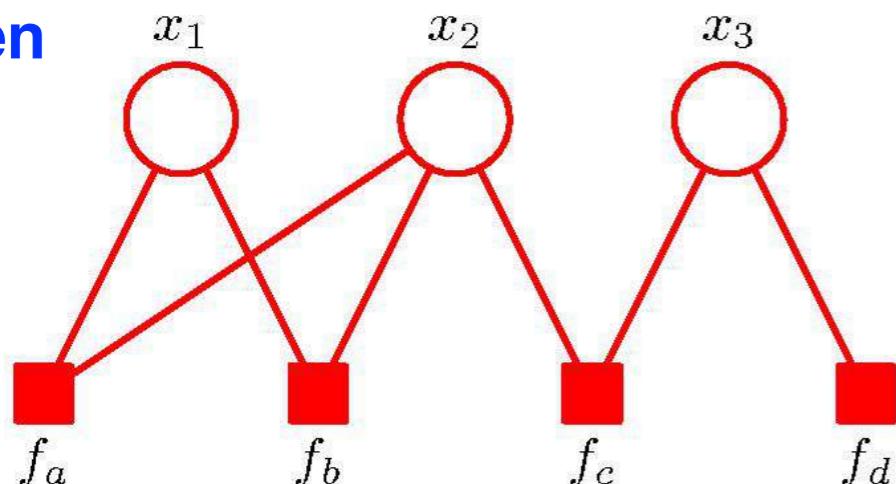
the **factor graph** (FG) has a node (represented by a square) for each factor  $\psi_i(\mathcal{X}_i)$  and a variable node (represented by a circle) for each variable  $x_j$ .

When used to represent a distribution

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_i \psi_i(\mathcal{X}_i),$$

a normalization constant is assumed.

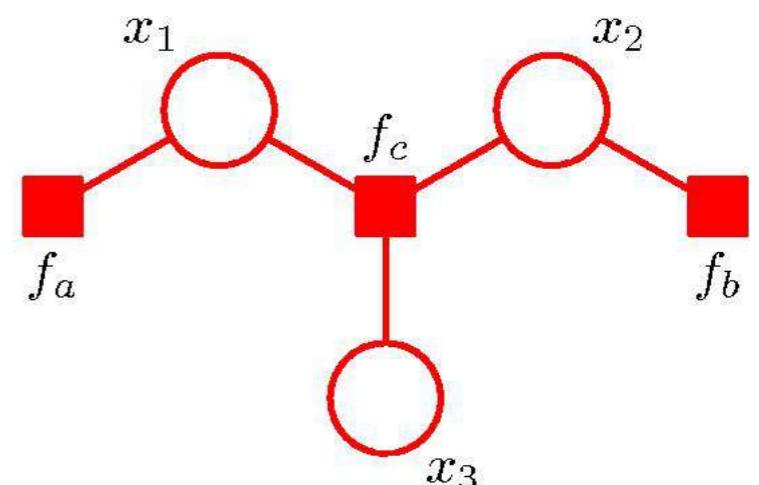
given



$$\rightarrow p(x) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

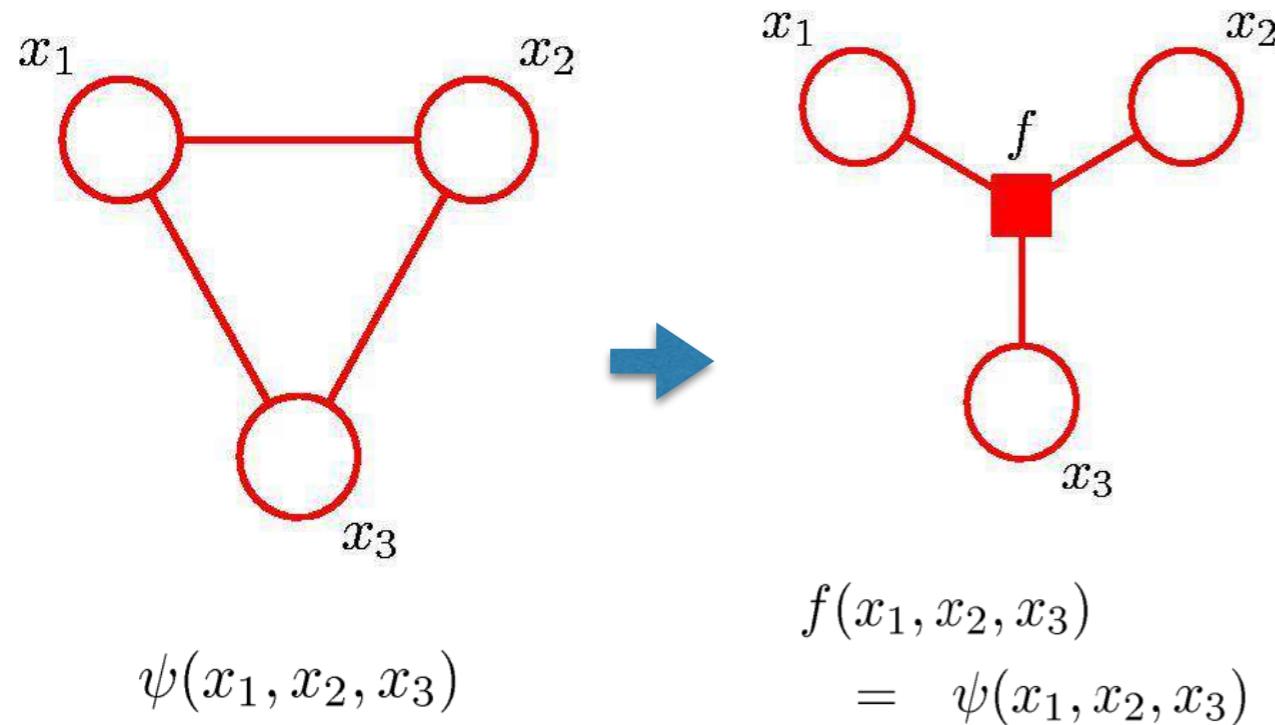
given

$$p(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3 | x_1, x_2)$$

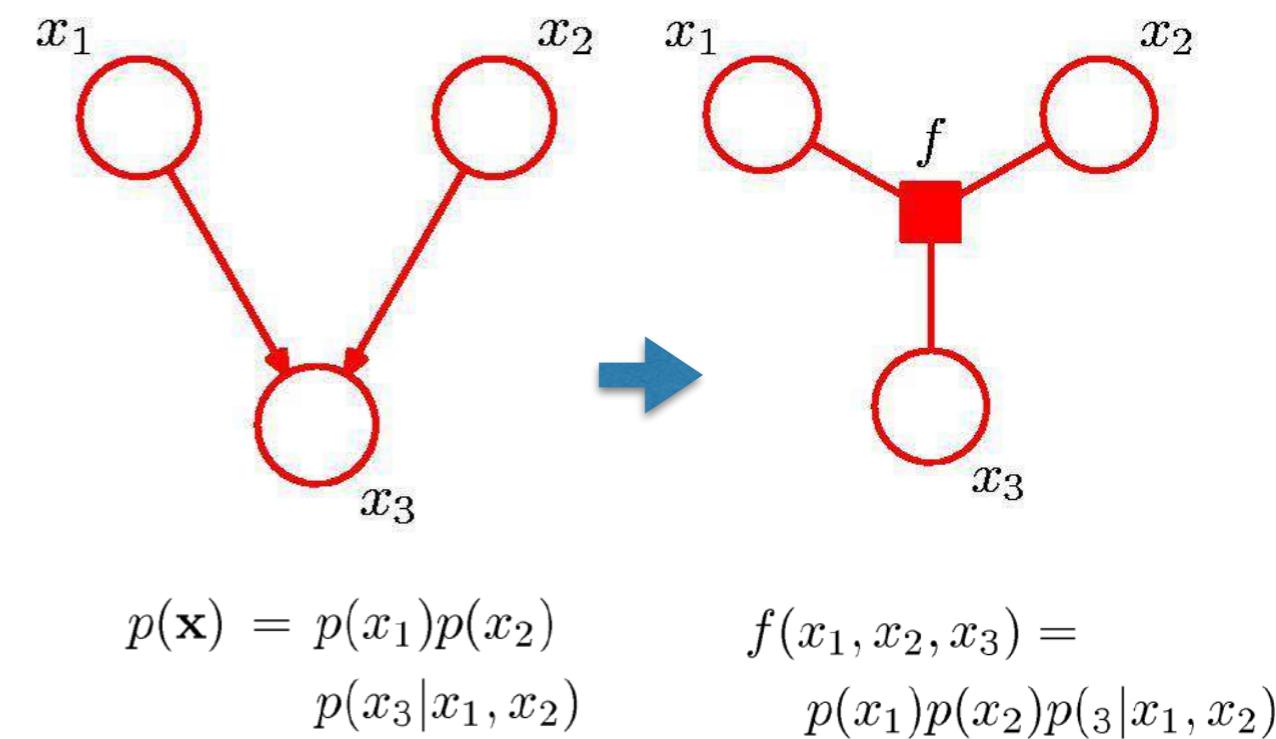


# Probabilistic Graphical Models - Factor Graph

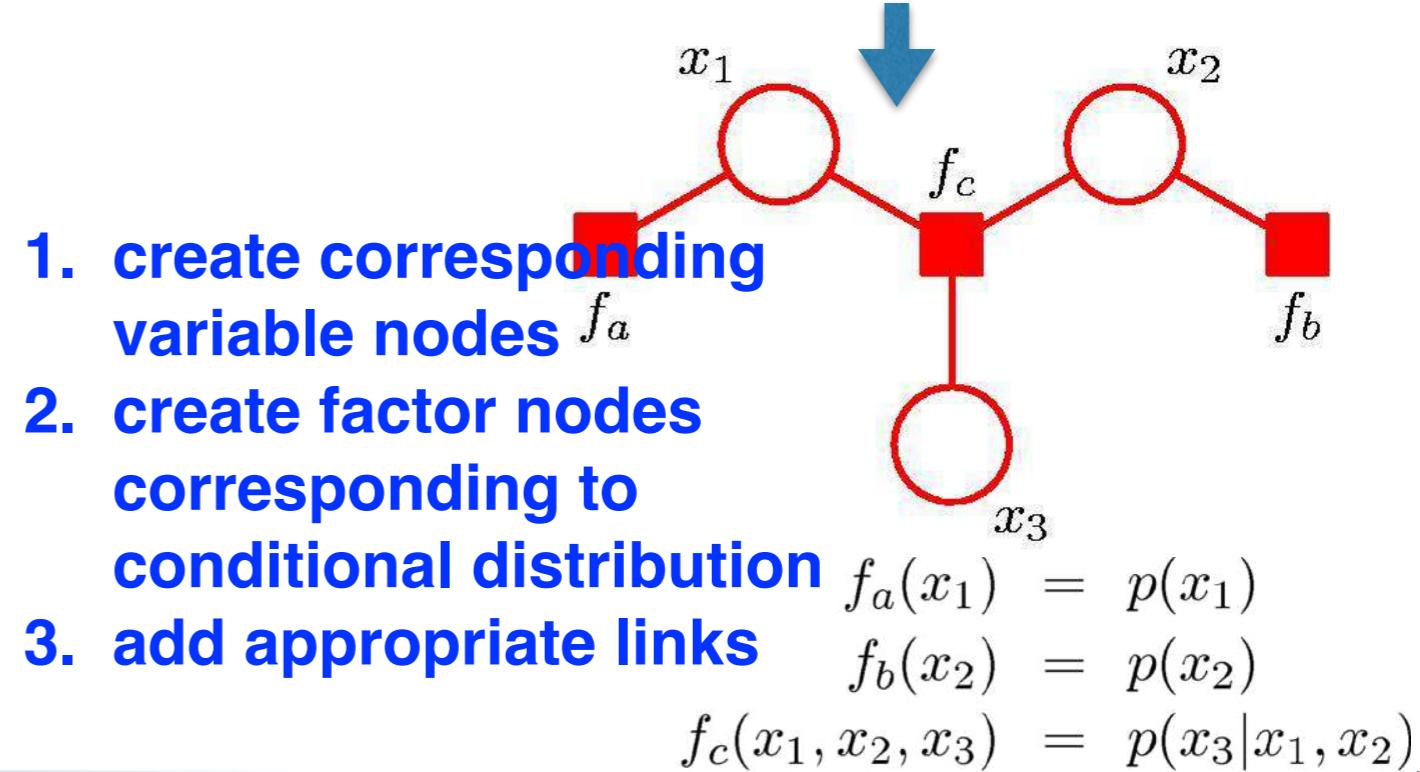
**factor graph from undirected graph**



**factor graph from directed graph**



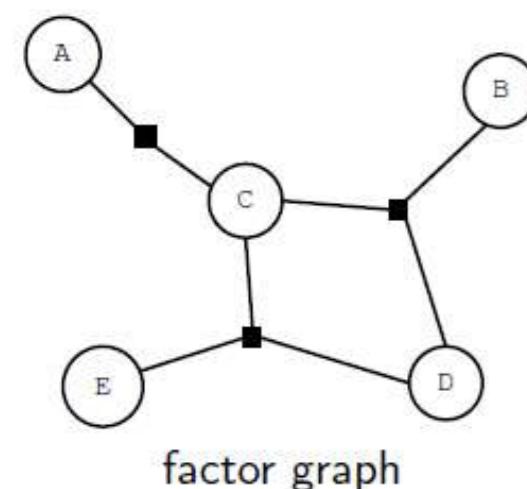
1. create corresponding variable nodes
2. create additional factor nodes corresponding to maximal cliques  $X_s$
3. The factors  $f_s(X_s)$  are set equal to the clique potentials



# Probabilistic Graphical Models - Factor Graph

**brief  
summary  
so far**

- ▶ With graphical models we represent probability distributions graphically
- ▶ Belief networks: directed graphs, causal dependency
- ▶ Markov networks: undirected, local cliques of dependent variables
- ▶ Factor graphs
  - ▶ Making the factorization explicit
  - ▶ Not a larger class of distributions, “just” a different way of drawing the graph
- ▶ Always think in terms of factor graphs

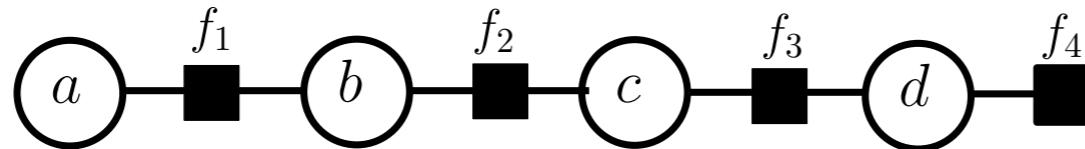


convenient for  
solving inference  
problems

$$\mathbb{E}_{p(x)}[x] = \sum_{x \in \mathcal{X}} xp(x)$$
$$x^* = \operatorname{argmax}_{x \in \mathcal{X}} p(x)$$
$$p(x_i, x_j \mid x_k, x_l)$$
$$x_i^* = \operatorname{argmax}_{x_i \in \mathcal{X}_i} \sum_{(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)} p(x)$$

mean  
mode  
condition  
max-marginal

# Probabilistic Graphical Models - Factor Graph



warm start from a simple chain structure

$$p(a, b, c, d) = \frac{1}{Z} f_1(a, b) f_2(b, c) f_3(c, d) f_4(d)$$

$$p(a, b, c) = \sum_d p(a, b, c, d) \text{ marginalize out } d$$

$$= \frac{1}{Z} f_1(a, b) f_2(b, c) \sum_d [f_3(c, d) f_4(d)]$$

move summation here  
since others doesn't  
depend on  $d$   
keep idea!

$$\mu_{d \rightarrow c}(c)$$

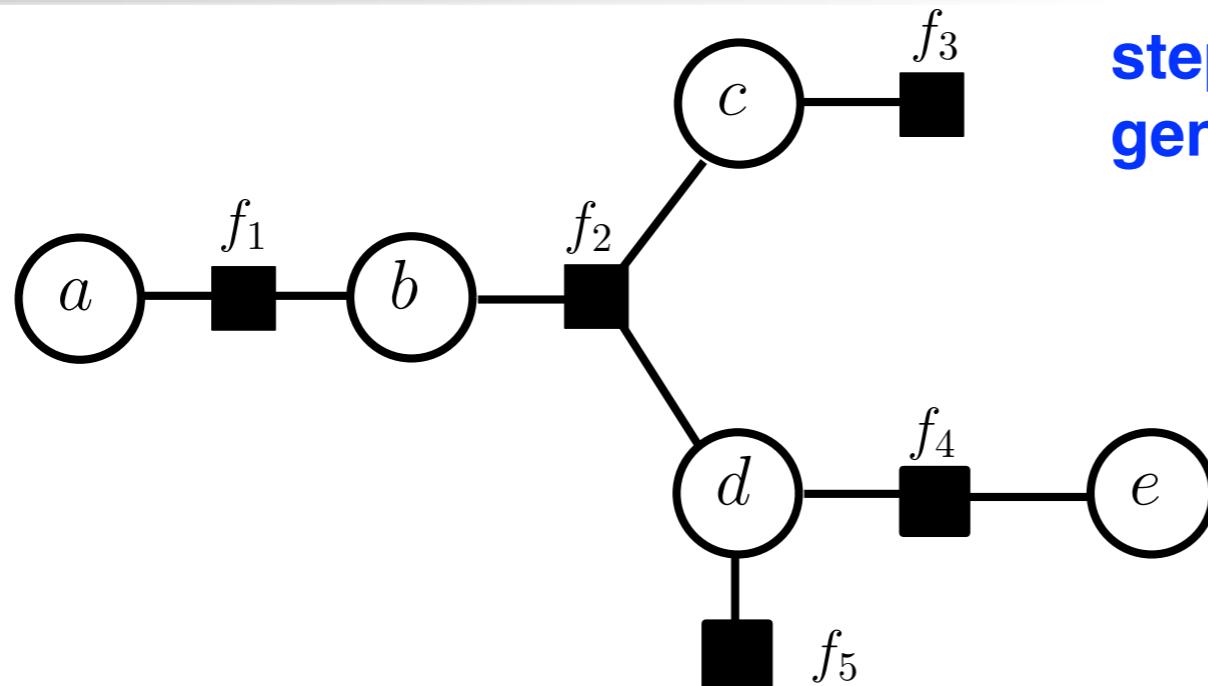
$$p(a, b) = \sum_c p(a, b, c) = \frac{1}{Z} f_1(a, b) \sum_c f_2(b, c) \mu_{d \rightarrow c}(c)$$

marginalize out  $c$

messages!

$$\mu_{c \rightarrow b}(b)$$

# Probabilistic Graphical Models - Factor Graph

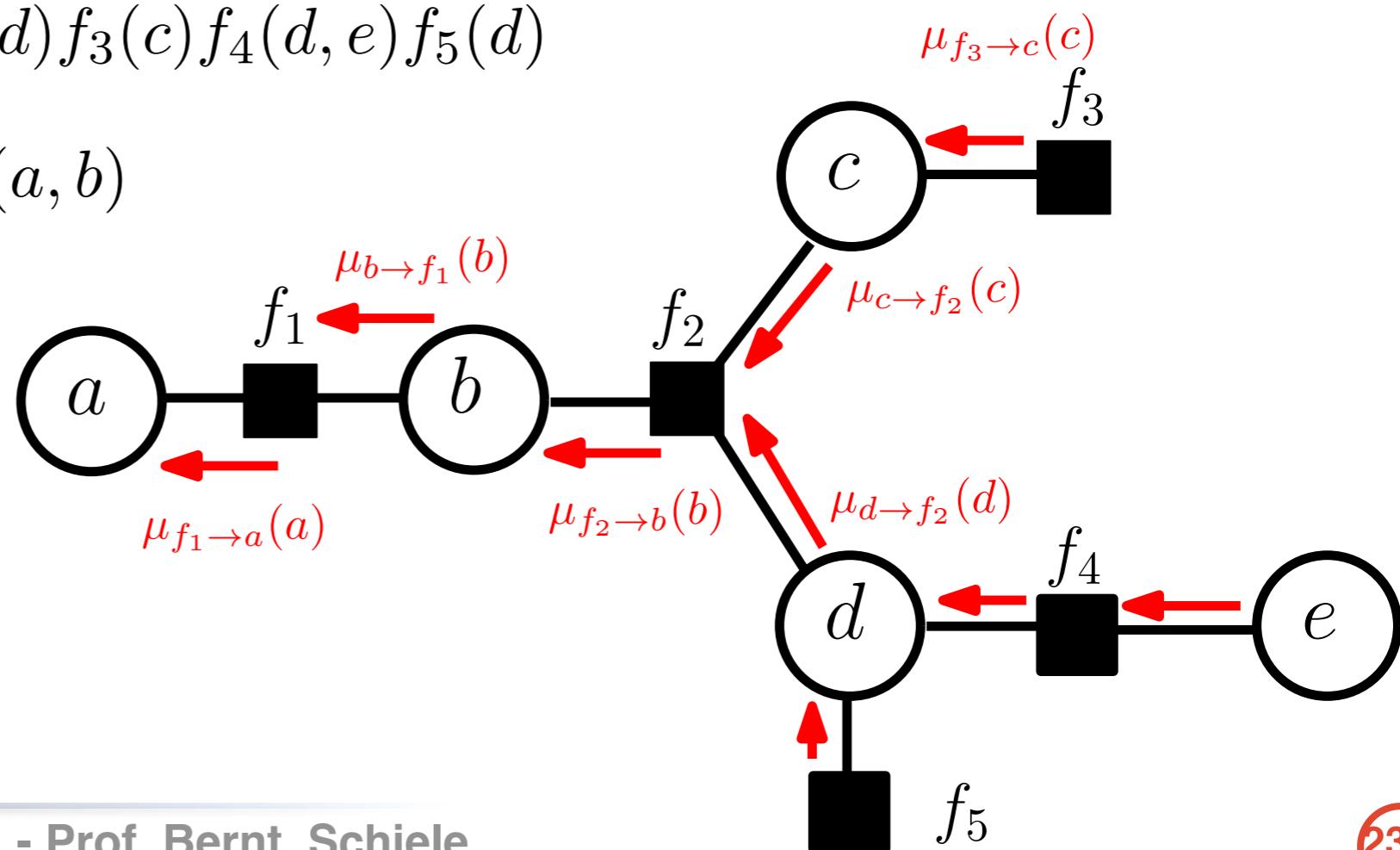


step further to consider  
general singly-connected factor graph

with factors

$$f_1(a, b) f_2(b, c, d) f_3(c) f_4(d, e) f_5(d)$$

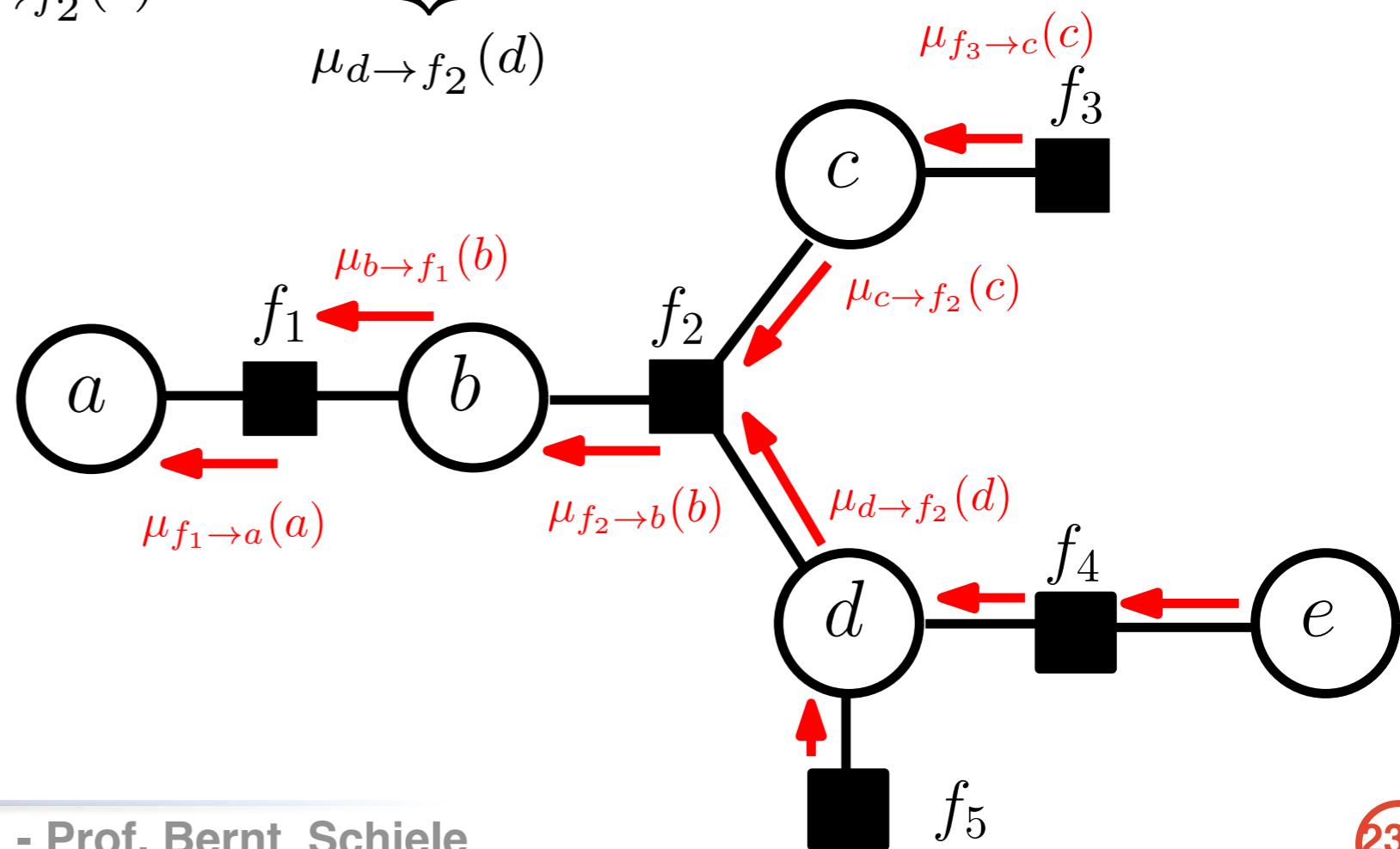
- ▶ For example: find marginal  $p(a, b)$
- ▶ Idea: compute messages



# Probabilistic Graphical Models - Factor Graph

$$p(a, b) = \frac{1}{Z} f_1(a, b) \underbrace{\sum_{c,d,e} f_2(b, c, d) f_3(c) f_5(d) f_4(d, e)}_{\mu_{f_2 \rightarrow b}(b)} \text{ marginalize out } c, d, e$$

$$\mu_{f_2 \rightarrow b}(b) = \sum_{c,d} f_2(b, c, d) \underbrace{f_3(c)}_{\mu_{c \rightarrow f_2}(c)} \underbrace{f_5(d) \sum_e f_4(d, e)}_{\mu_{d \rightarrow f_2}(d)}$$



# Probabilistic Graphical Models - Factor Graph

$$p(a, b) = \frac{1}{Z} f_1(a, b) \sum_{c,d,e} f_2(b, c, d) f_3(c) f_5(d) f_4(d, e)$$

$\underbrace{\hspace{10em}}_{\mu_{f_2 \rightarrow b}(b)}$

$$\mu_{f_2 \rightarrow b}(b) = \sum_{c,d} f_2(b, c, d) \underbrace{f_3(c)}_{\mu_{c \rightarrow f_2}(c)} \underbrace{f_5(d)}_{\mu_{d \rightarrow f_2}(d)} \sum_e f_4(d, e)$$

$\underbrace{\hspace{10em}}_{\mu_{d \rightarrow f_2}(d)}$

$$\mu_{d \rightarrow f_2}(d) = \mu_{f_5 \rightarrow d}(d) \mu_{f_4 \rightarrow d}(d)$$

**Variable-to-Factor  
Messages**



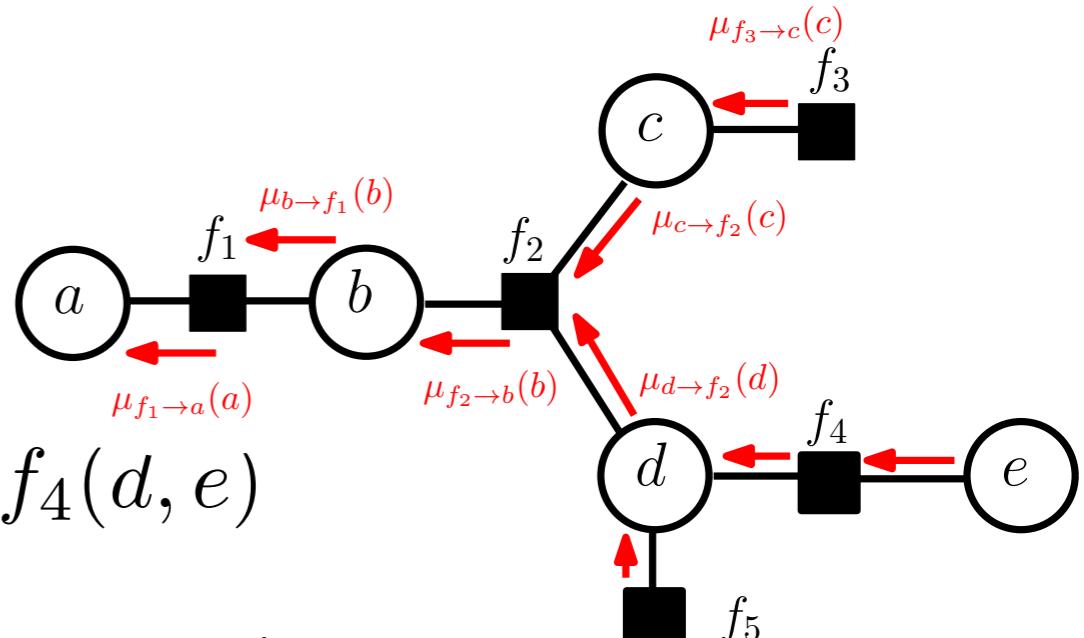
(like just passing)

- more general:

$$\mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}}$$

$\overbrace{\hspace{10em}}^{\text{product}}$

**other messages  
into the variable**



# Probabilistic Graphical Models - Factor Graph

$$p(a, b) = \frac{1}{Z} f_1(a, b) \underbrace{\sum_{c,d,e} f_2(b, c, d) f_3(c) f_5(d) f_4(d, e)}_{\mu_{f_2 \rightarrow b}(b)}$$

$$\mu_{f_2 \rightarrow b}(b) = \sum_{c,d} f_2(b, c, d) \underbrace{f_3(c)}_{\mu_{c \rightarrow f_2}(c)} \underbrace{f_5(d) \sum_e f_4(d, e)}_{\mu_{d \rightarrow f_2}(d)}$$

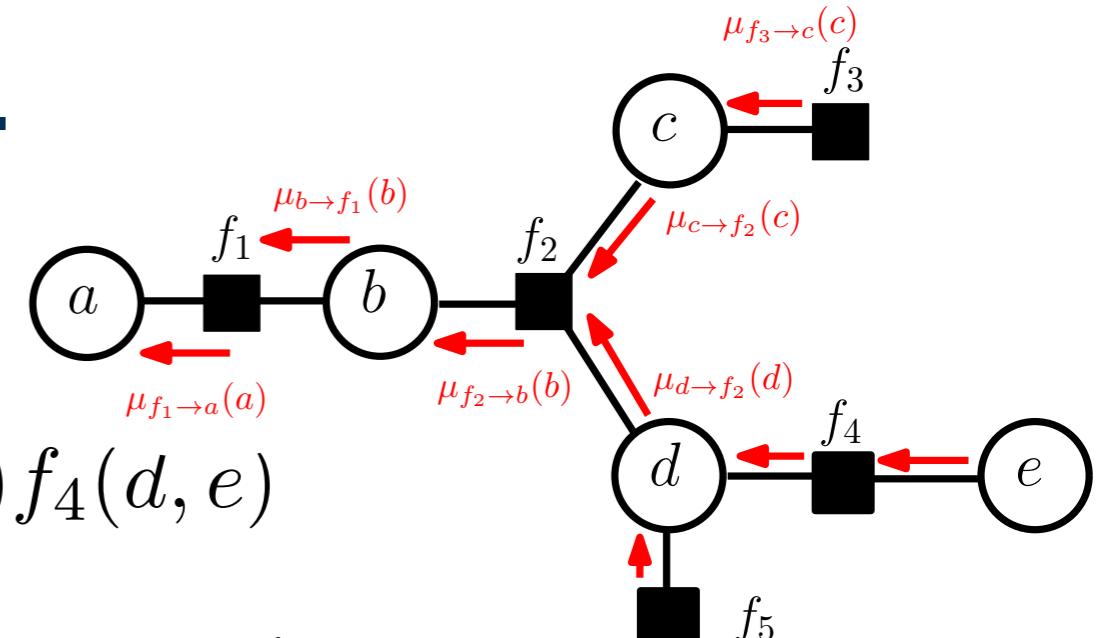
$$\mu_{f_2 \rightarrow b}(b) = \sum_{c,d} f_2(b, c, d) \mu_{c \rightarrow f_2}(c) \mu_{d \rightarrow f_2}(d)$$

**Factor-to-Variable  
Messages**

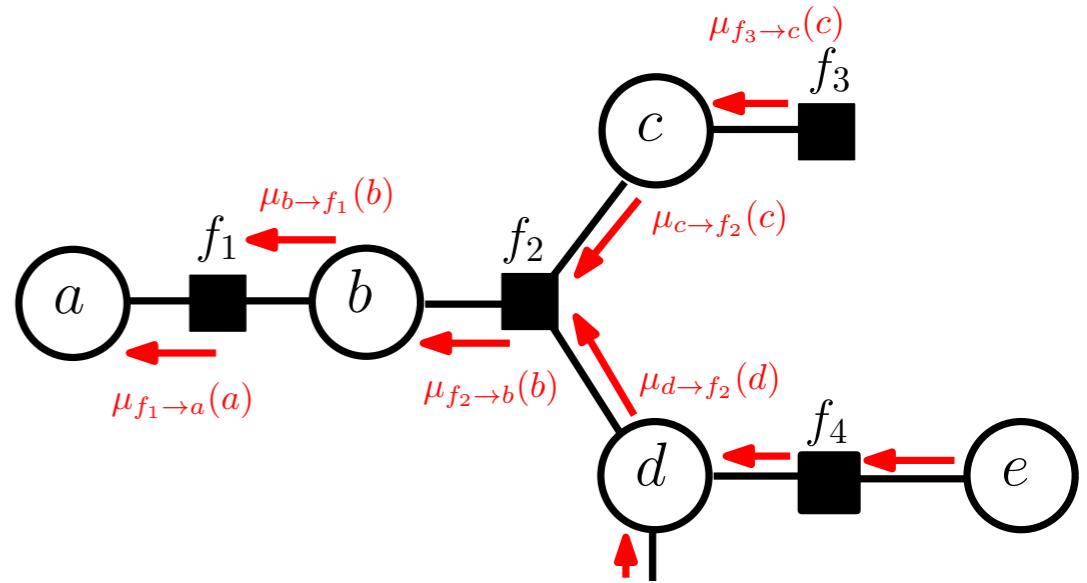
- more general:

$$\mu_{f \rightarrow x}(x) = \sum_{y \in \mathcal{X}_f \setminus x} \frac{\phi_f(\mathcal{X}_f)}{\text{factor}} \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f}(y)$$

summation      product  
marginalize itself      other messages into the factor  
out others      except target variable



# Probabilistic Graphical Models - Factor Graph



- If we want to compute the marginal  $p(a)$  (use factor-to-variable message):

$$p(a) = \frac{1}{Z} \mu_{f_1 \rightarrow a}(a) = \underbrace{\frac{1}{Z} \sum_b f_1(a, b) \mu_{b \rightarrow f_1}(b)}_{\mu_{f_1 \rightarrow a}(a)}$$

- which we could also view as

$$p(a) = \frac{1}{Z} \sum_b f_1(a, b) \underbrace{\mu_{b \rightarrow f_1}(b)}_{\mu_{f_2 \rightarrow b}(b)}$$

  
**variable  $b$**   
**just passing**

# Probabilistic Graphical Models - Factor Graph

- ▶ Many subscripts :)
- ▶ Once computed, messages can be re-used
- ▶ All marginals ( $p(c), p(d), p(c, d), \dots$ ) can be written as a function of messages
- ▶ The algorithm to compute all messages: Sum-Product algorithm



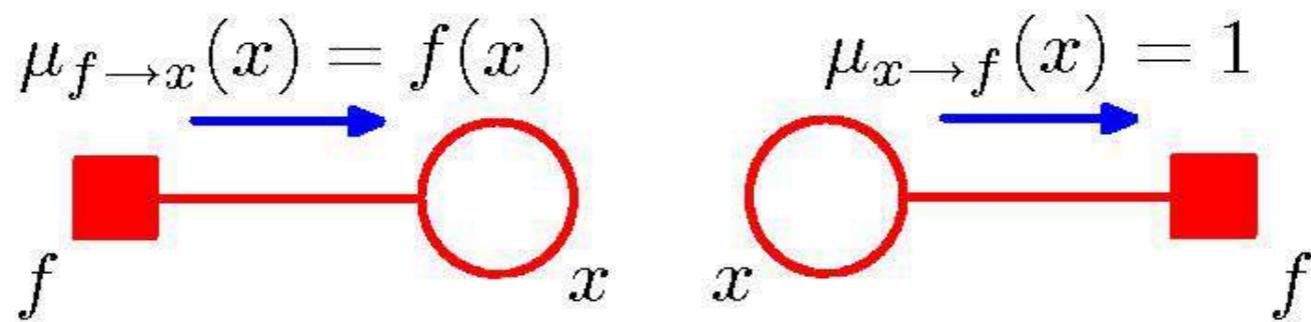
- ▶ Algorithm to compute all messages efficiently
- ▶ Assuming the graph is singly-connected (= tree)
  1. Initialization
  2. Variable to Factor message
  3. Factor to Variable message
- ▶ Then compute any desired marginals
- ▶ Also known as belief propagation

# Probabilistic Graphical Models - Factor Graph

Sum-Product algorithm

1. Initialization
2. Variable to Factor message
3. Factor to Variable message

- ▶ Messages from extremal (simplicial) node factors are initialized to the factor (left)
- ▶ Messages from extremal (simplicial) variable nodes are set to unity (right)

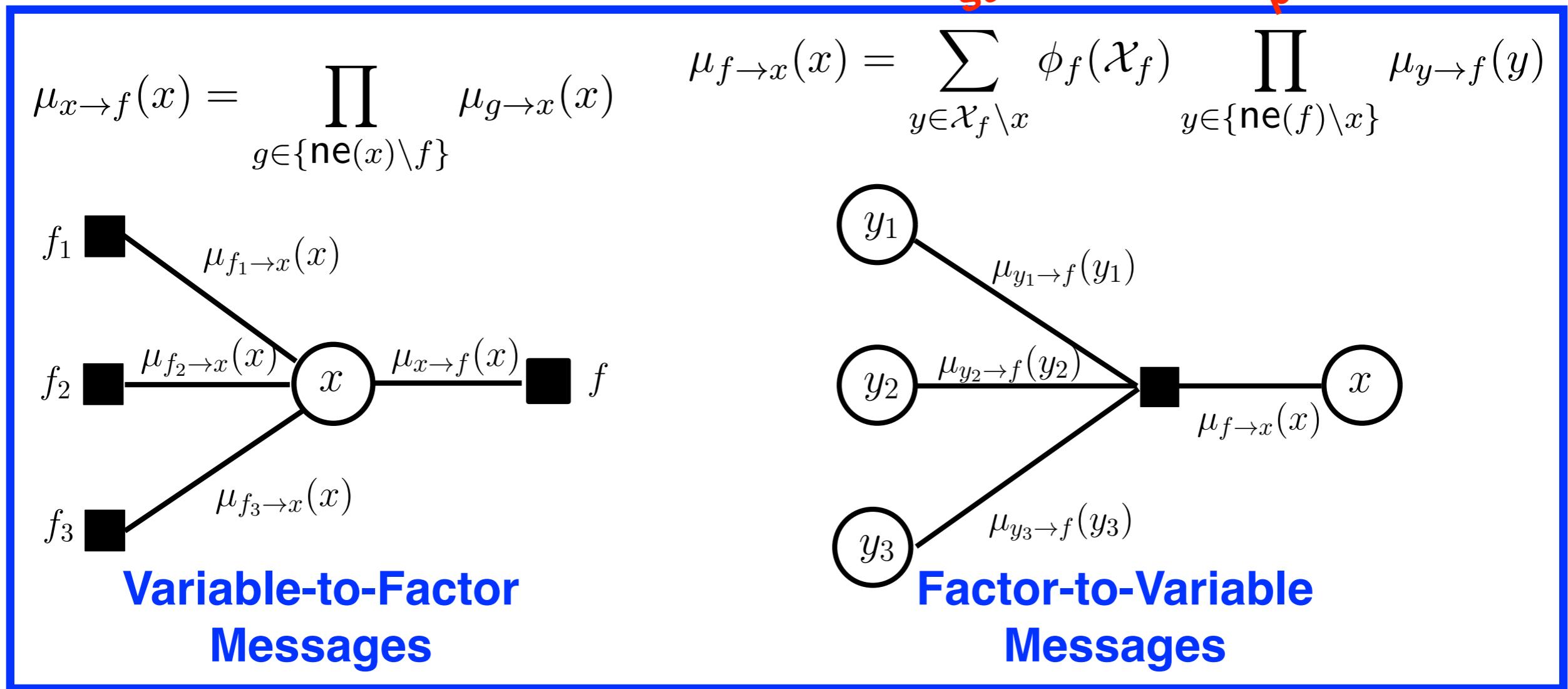


# Probabilistic Graphical Models - Factor Graph

Sum-Product algorithm

1. Initialization
2. Variable to Factor message
3. Factor to Variable message

explain the name of algorithm: sum-product



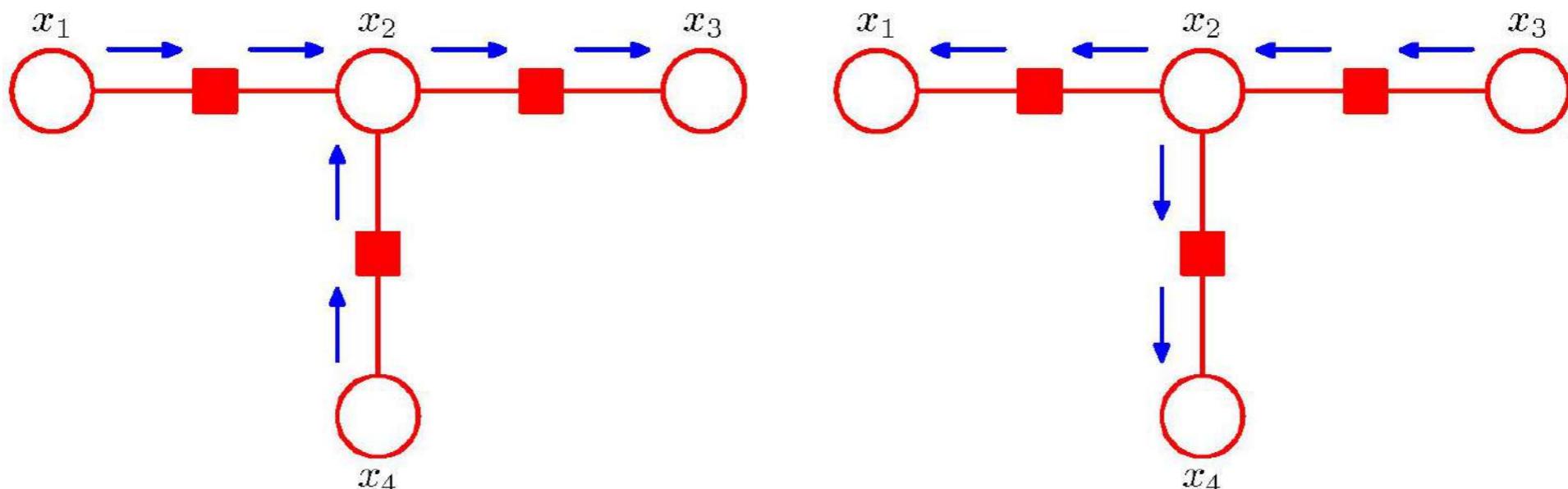
# Probabilistic Graphical Models - Factor Graph

Sum-Product algorithm

1. Initialization
2. Variable to Factor message
3. Factor to Variable message

► To compute all messages in the graph

1. leaf-to-root: (pick root node - here  $x_3$  - compute messages pointing towards root)
2. root-to-leave: (compute messages pointing away from root)  
**storing received messages at every node!**



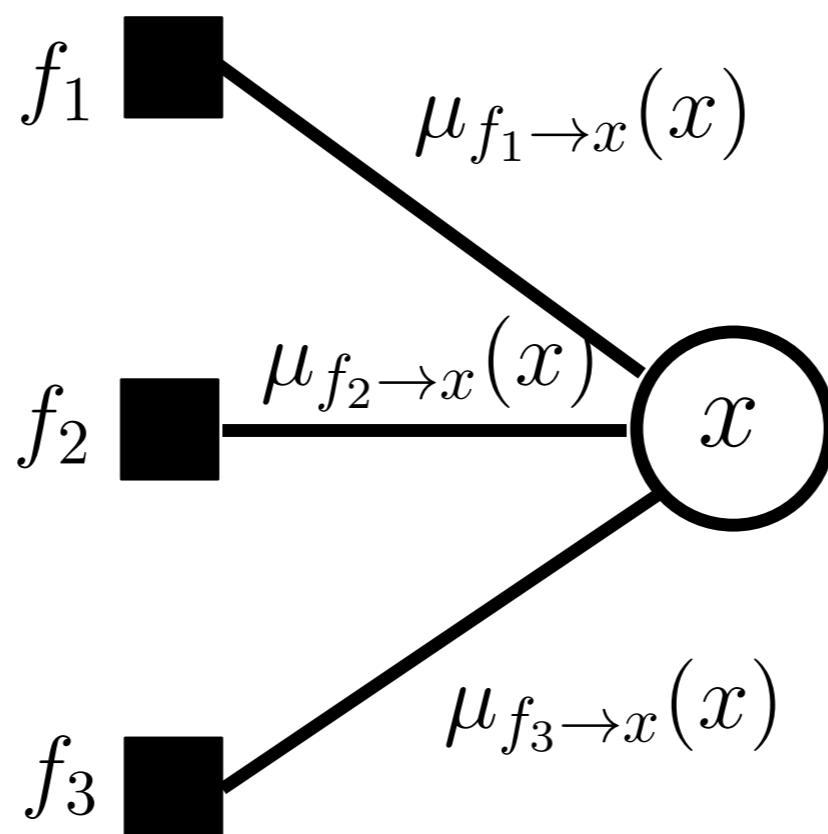
# Probabilistic Graphical Models - Factor Graph

Sum-Product algorithm

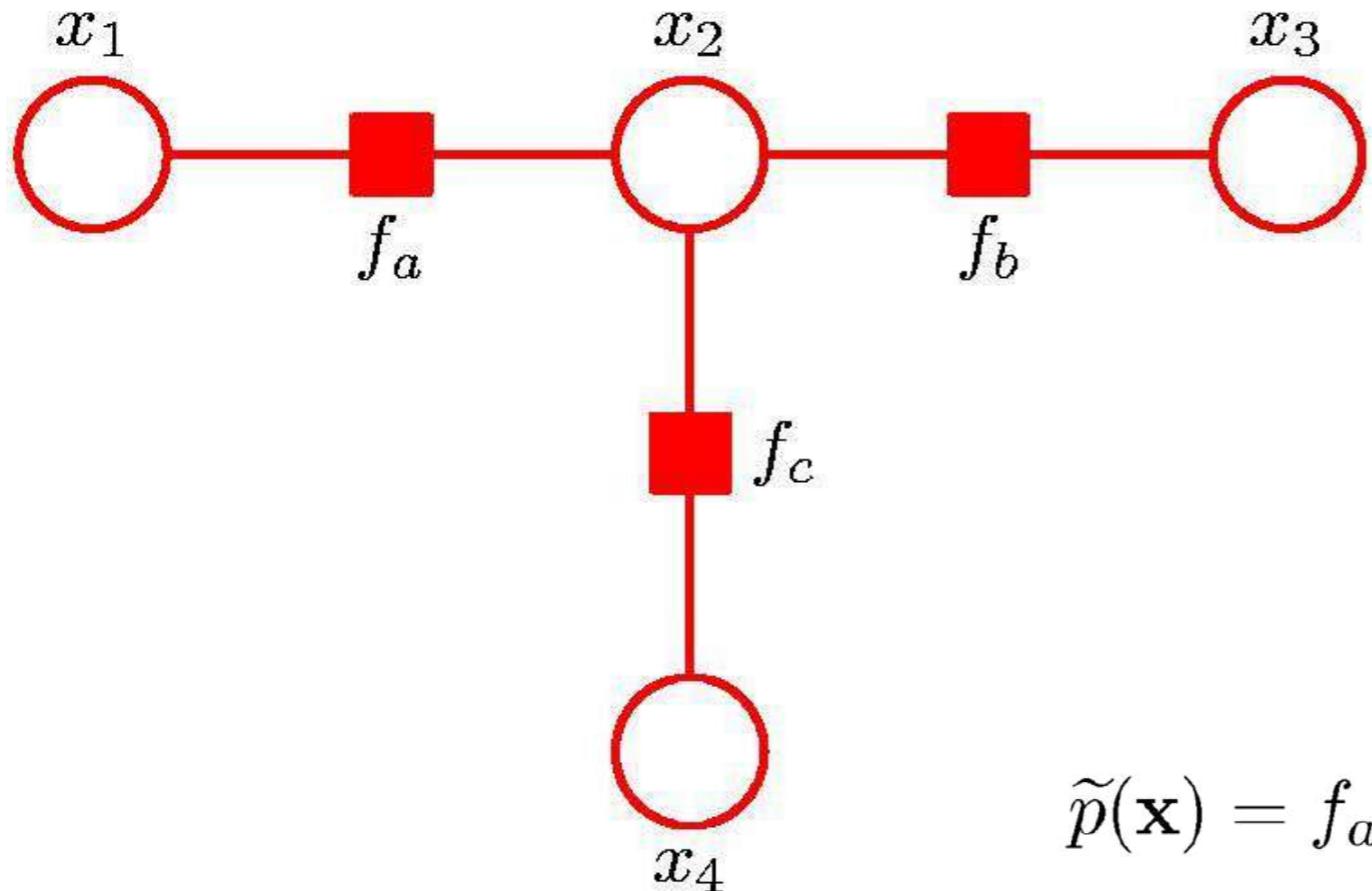
1. Initialization
2. Variable to Factor message
3. Factor to Variable message

$$p(x) \propto \prod_{f \in \text{ne}(x)} \mu_{f \rightarrow x}(x)$$

**then easy to  
have marginal  
Compute product of received  
messages at the node**

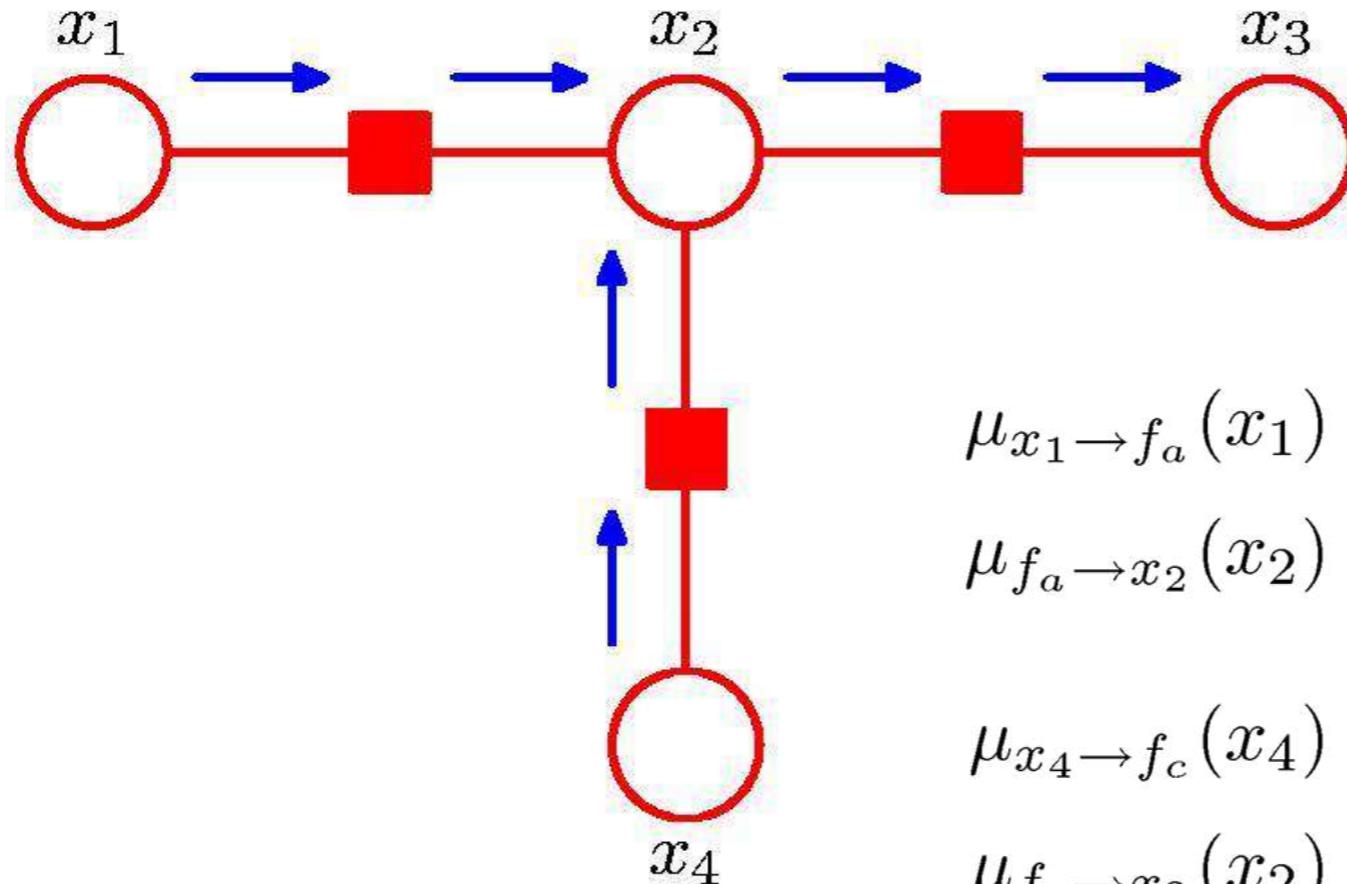


# Probabilistic Graphical Models - Factor Graph



$$\tilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

# Probabilistic Graphical Models - Factor Graph



$$\mu_{x_1 \rightarrow f_a}(x_1) = 1$$

$$\mu_{f_a \rightarrow x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2)$$

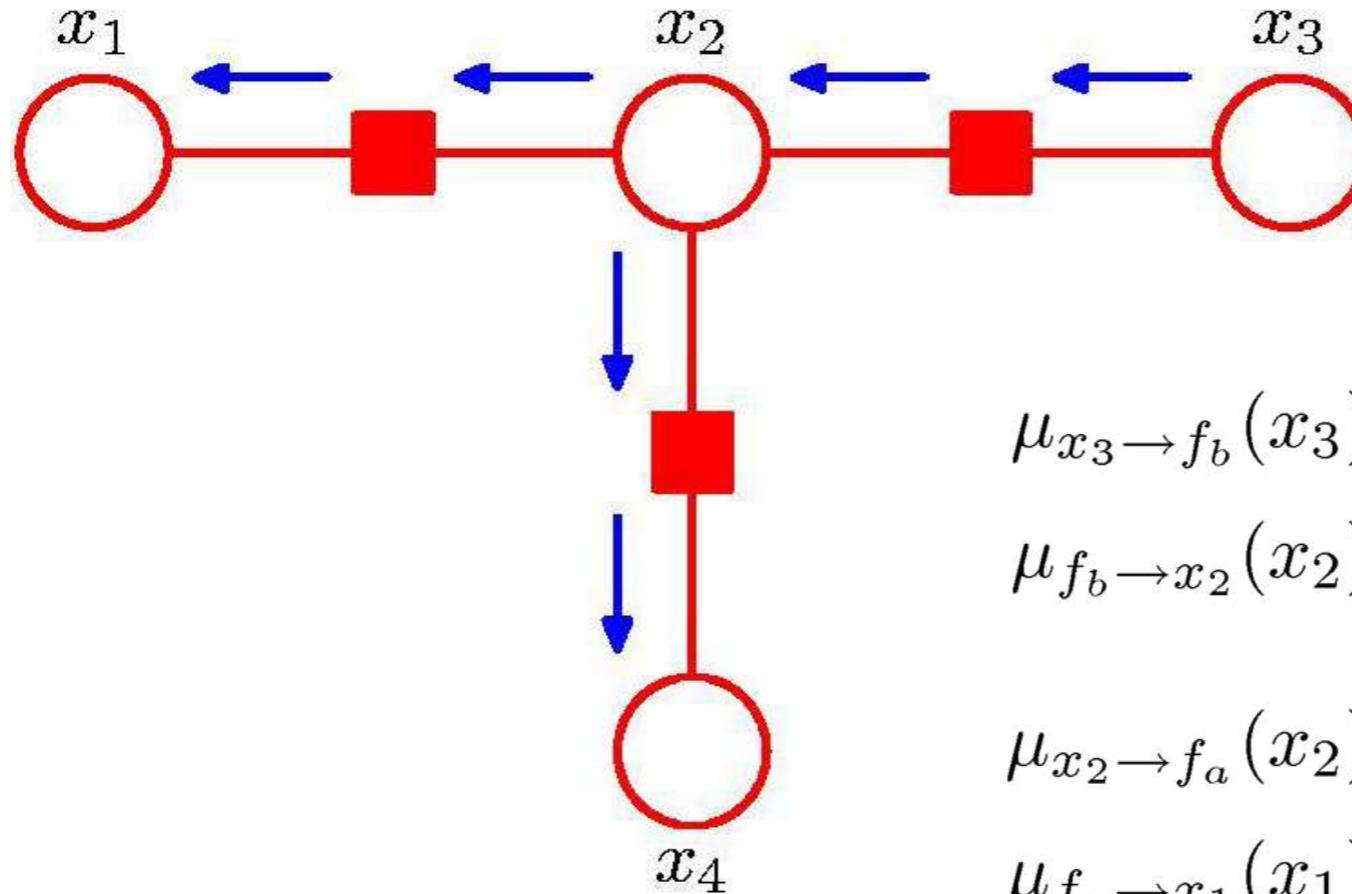
$$\mu_{x_4 \rightarrow f_c}(x_4) = 1$$

$$\mu_{f_c \rightarrow x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4)$$

$$\mu_{x_2 \rightarrow f_b}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2)$$

$$\mu_{f_b \rightarrow x_3}(x_3) = \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \rightarrow f_b}(x_2)$$

# Probabilistic Graphical Models - Factor Graph



$$\mu_{x_3 \rightarrow f_b}(x_3) = 1$$

$$\mu_{f_b \rightarrow x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3)$$

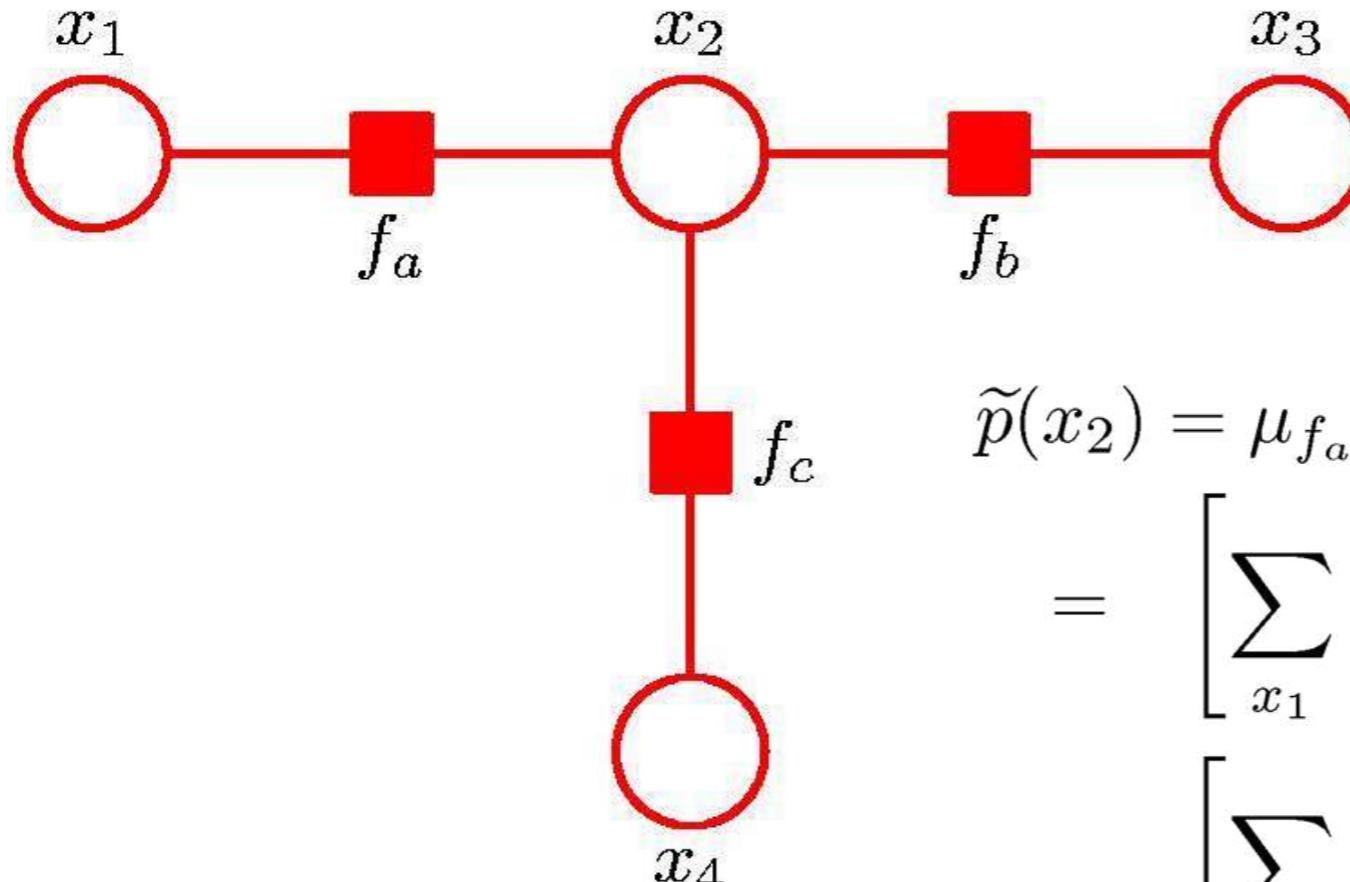
$$\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2)$$

$$\mu_{f_a \rightarrow x_1}(x_1) = \sum_{x_2} f_a(x_1, x_2) \mu_{x_2 \rightarrow f_a}(x_2)$$

$$\mu_{x_2 \rightarrow f_c}(x_2) = \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_b \rightarrow x_2}(x_2)$$

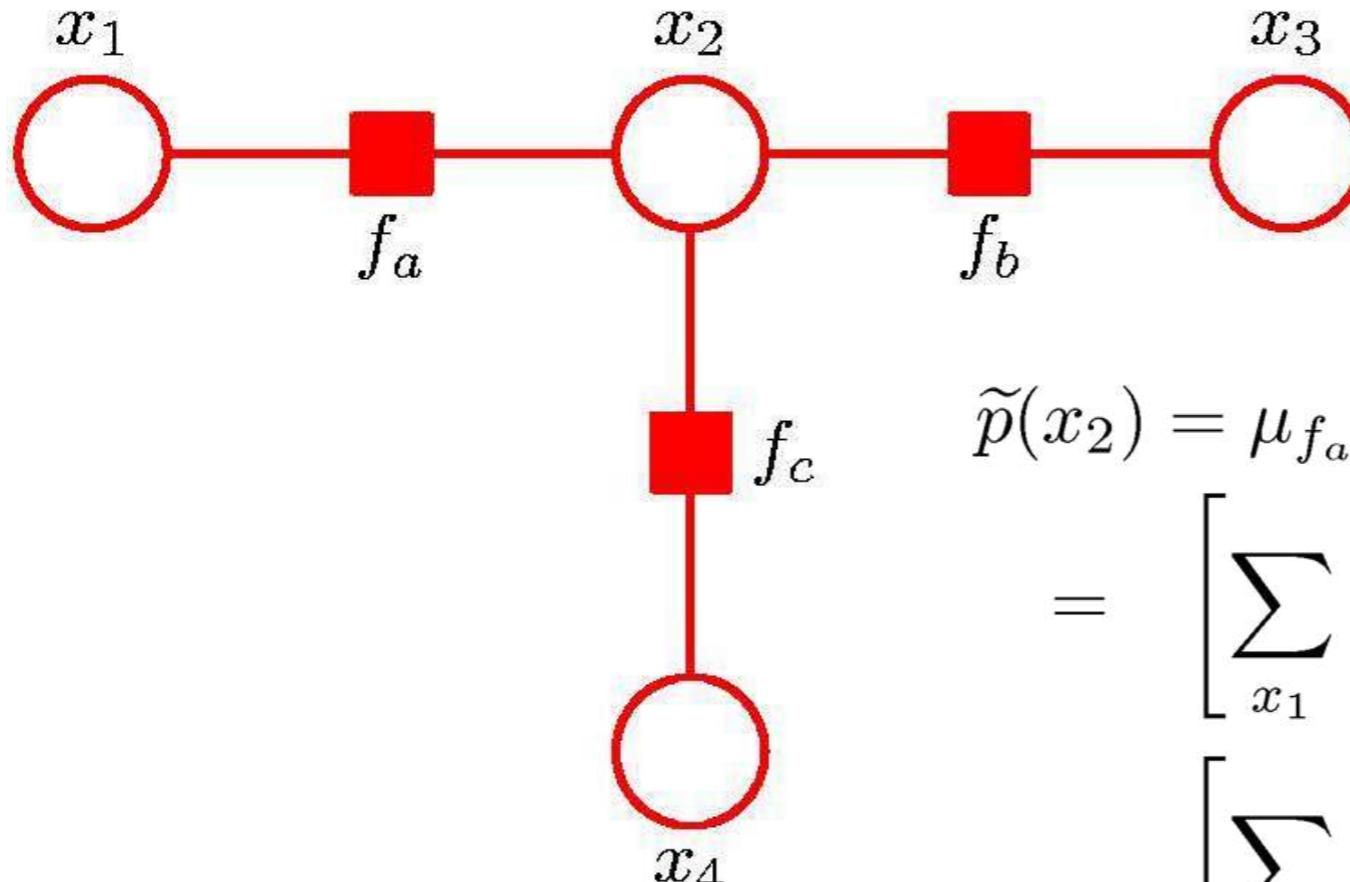
$$\mu_{f_c \rightarrow x_4}(x_4) = \sum_{x_2} f_c(x_2, x_4) \mu_{x_2 \rightarrow f_c}(x_2)$$

# Probabilistic Graphical Models - Factor Graph



$$\begin{aligned}\tilde{p}(\mathbf{x}) &= \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \\ &= \left[ \sum_{x_1} f_a(x_1, x_2) \right] \left[ \sum_{x_3} f_b(x_2, x_3) \right] \\ &\quad \left[ \sum_{x_4} f_c(x_2, x_4) \right] \\ &= \sum_{x_1} \sum_{x_3} \sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\ &= \sum_{x_1} \sum_{x_3} \sum_{x_4} \tilde{p}(\mathbf{x})\end{aligned}$$

# Probabilistic Graphical Models - Factor Graph



$$\begin{aligned}\tilde{p}(x_2) &= \mu_{f_a \rightarrow x_2}(x_2) \mu_{f_b \rightarrow x_2}(x_2) \mu_{f_c \rightarrow x_2}(x_2) \\ &= \left[ \sum_{x_1} f_a(x_1, x_2) \right] \left[ \sum_{x_3} f_b(x_2, x_3) \right] \\ &\quad \left[ \sum_{x_4} f_c(x_2, x_4) \right] \\ &= \sum_{x_1} \sum_{x_3} \sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\ &= \sum_{x_1} \sum_{x_3} \sum_{x_4} \tilde{p}(\mathbf{x})\end{aligned}$$

# Probabilistic Graphical Models - Factor Graph

## Max-Sum Algorithm

Objective: an efficient algorithm for finding

- i. the value  $\mathbf{x}^{\max}$  that maximises  $p(\mathbf{x})$ ; **joint maximum**
- ii. the value of  $p(\mathbf{x}^{\max})$ .

In general, maximum marginals  $\neq$  joint maximum.

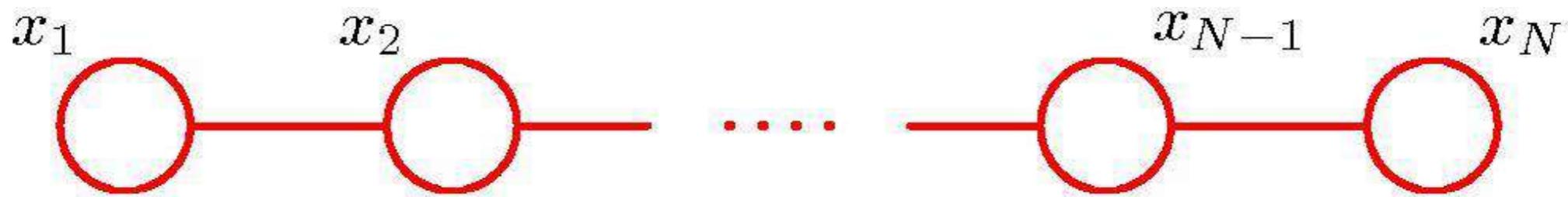
		$x = 0$	$x = 1$
$y = 0$	$x = 0$	0.3	0.4
	$x = 1$	0.3	0.0

$$\arg \max_x p(x, y) = 1 \quad \arg \max_x p(x) = 0$$

# Probabilistic Graphical Models - Factor Graph

## Max-Sum Algorithm

Maximizing over a chain (max-product)



$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_1} \dots \max_{x_M} p(\mathbf{x})$$

**move maximum operations  
to corresponding parts!**

$$= \frac{1}{Z} \max_{x_1} \dots \max_{x_N} [\psi_{1,2}(x_1, x_2) \dots \psi_{N-1,N}(x_{N-1}, x_N)]$$

$$= \frac{1}{Z} \max_{x_1} \left[ \max_{x_2} \left[ \psi_{1,2}(x_1, x_2) \left[ \dots \max_{x_N} \psi_{N-1,N}(x_{N-1}, x_N) \right] \dots \right] \right]$$

# Probabilistic Graphical Models - Factor Graph

## Max-Sum Algorithm

Generalizes to tree-structured factor graph

$$\max_{\mathbf{x}} p(\mathbf{x}) = \max_{x_n} \prod_{f_s \in \text{ne}(x_n)} \max_{X_s} f_s(x_n, X_s)$$

maximizing as close to the leaf nodes as possible

If a message is sent from a factor node  $f$  to a variable node  $x$   
→ Maximization is performed over all other variable nodes  $x_1, \dots, x_n$  that are neighbors of the factor node → backtracing!

Max-Product → Max-Sum

For numerical reasons, use

$$\ln \left( \max_{\mathbf{x}} p(\mathbf{x}) \right) = \max_{\mathbf{x}} \ln p(\mathbf{x})$$

# Probabilistic Graphical Models - Factor Graph

## Max-Sum Algorithm

Initialization (leaf nodes)

$$\mu_{x \rightarrow f}(x) = 0 \quad \mu_{f \rightarrow x}(x) = \ln f(x)$$

Recursion

$$\begin{aligned}\mu_{f \rightarrow x}(x) &= \max_{x_1, \dots, x_M} \left[ \ln f(x, x_1, \dots, x_M) + \sum_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f}(x_m) \right] \\ \phi(x) &= \arg \max_{x_1, \dots, x_M} \left[ \ln f(x, x_1, \dots, x_M) + \sum_{m \in \text{ne}(f_s) \setminus x} \mu_{x_m \rightarrow f}(x_m) \right] \\ \mu_{x \rightarrow f}(x) &= \sum_{l \in \text{ne}(x) \setminus f} \mu_{f_l \rightarrow x}(x)\end{aligned}$$

# Probabilistic Graphical Models - Factor Graph

## Max-Sum Algorithm

Termination (root node)

$$p^{\max} = \max_x \left[ \sum_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x) \right]$$

$$x^{\max} = \arg \max_x \left[ \sum_{s \in \text{ne}(x)} \mu_{f_s \rightarrow x}(x) \right]$$

Back-track, for all nodes  $i$  with  $l$  factor nodes  
to the root ( $l=0$ )

$$\mathbf{x}_l^{\max} = \phi(x_{i,l-1}^{\max})$$

# Probabilistic Graphical Models -

## Factor Graph

- ▶ Factor-to-Variable messages

in large graph, messages may become very small. Similar trick to use log-message then can be applied on normal sum-product

$$\mu_{f \rightarrow x}(x) = \sum_{y \in \mathcal{X}_f \setminus x} \Phi_f(\mathcal{X}_f) \prod_{y \in \{\text{ne}(f) \setminus x\}} \mu_{y \rightarrow f}(y)$$

then becomes

$$\lambda_{f \rightarrow x}(x) = \log \left( \sum_{y \in \mathcal{X}_f \setminus x} \Phi(\mathcal{X}_f) \exp \left[ \sum_{y \in \{\text{ne}(f) \setminus x\}} \lambda_{y \rightarrow f}(y) \right] \right)$$

- ▶ large numbers lead to numerical instability
- ▶ Use the following equality



$$\log \sum_i \exp(v_i) = \alpha + \log \sum_i \exp(v_i - \alpha)$$

- ▶ With  $\alpha = \max \lambda_{y \rightarrow f}(y)$

# Probabilistic Graphical Models -

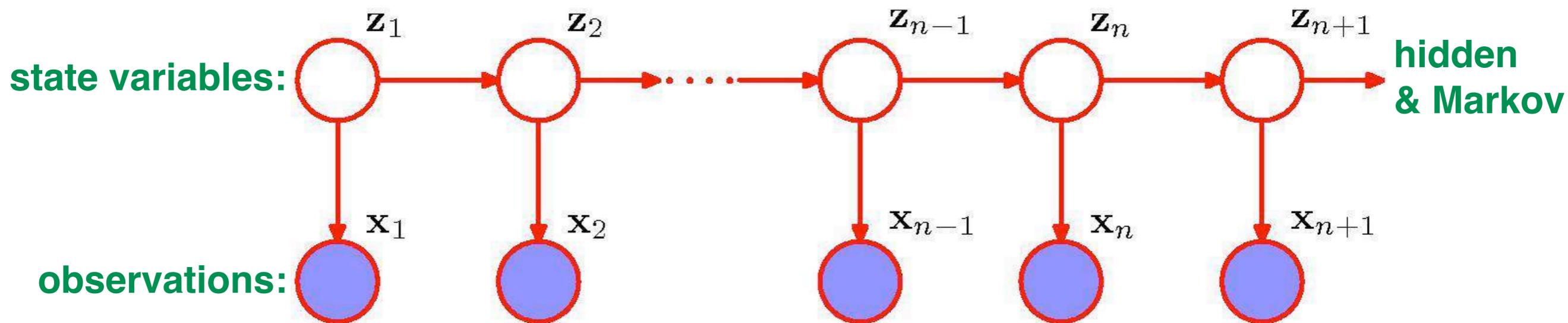
## Factor Graph

- Sum-product and max-sum algorithms
  - are efficient and exact solutions
  - to inference problems in **tree-structured** graphs
- In some cases we need to deal with graphs with loops
  - Message passing framework can be **generalized to arbitrary graph topologies**
  - **Junction tree algorithm (exact)**
  - **Loop belief propagation (approximate)**



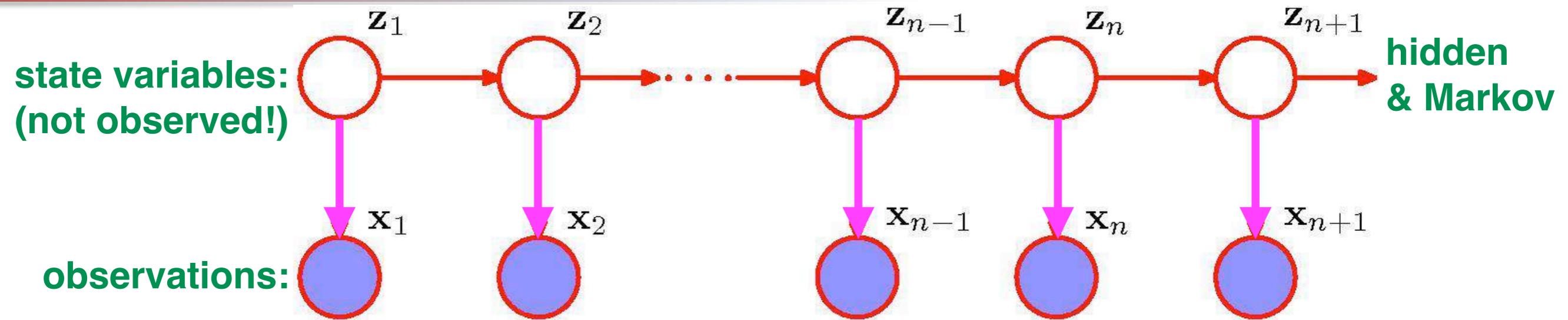
# Hidden Markov Model

- Formally defined on a sequence of observed random variables  $\{x_1, \dots, x_N\}$  through corresponding latent sequence  $\{z_1, \dots, z_N\}$
- HMMs are **generative models** (state:  $z_n$ ; observation:  $x_n$ )



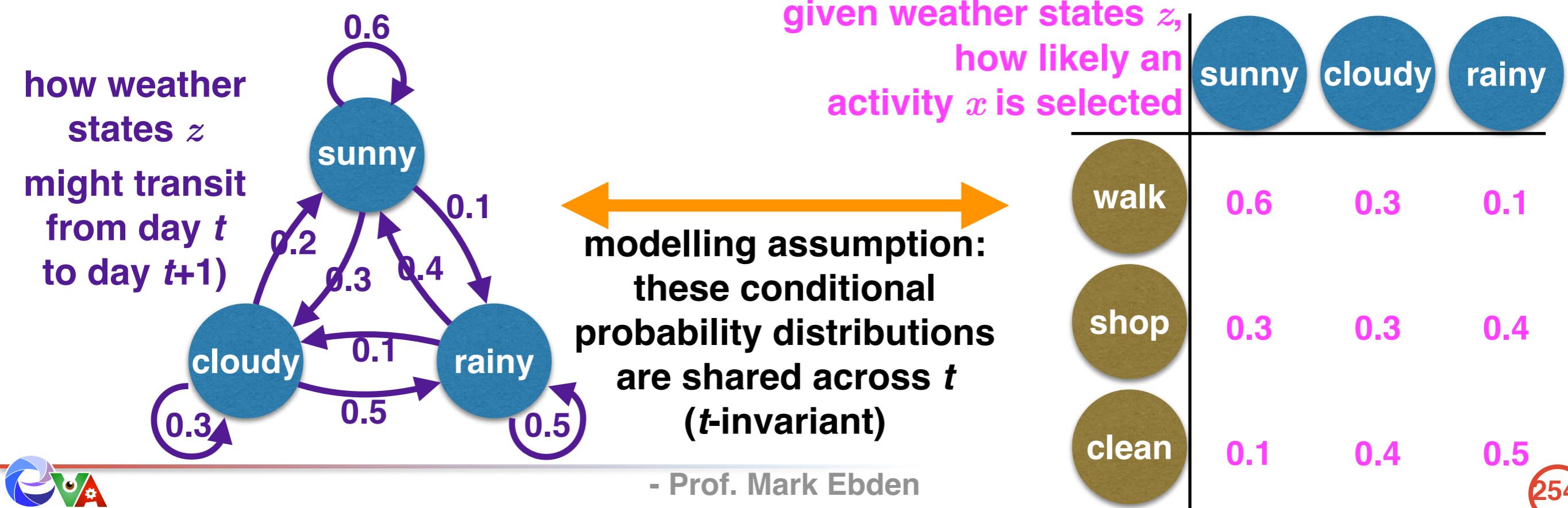
- hidden states are connected through a **degree-2 Markov** chain
$$\forall n < m - 1 : z_n \perp\!\!\!\perp z_m \mid z_{m-1}$$
- the future observations and the past observations are conditionally independent given the current hidden states
$$\{x_1, \dots, x_{m-1}\} \perp\!\!\!\perp \{x_{m+1}, \dots, x_N\} \mid z_m$$
- the present is dependent on the entire past observations
$$x_{m-2} \not\perp\!\!\!\perp x_m \mid x_{m-1}$$

# Hidden Markov Model



→ state transition probabilities  $p(z_t|z_{t-1}) : \{p(z_2|z_1), p(z_3|z_2), \dots, p(z_n|z_{n-1})\}$

↓  
emission probabilities  $p(x_t|z_t) : \{p(x_1|z_1), p(x_2|z_2), \dots, p(x_n|z_n)\}$



# Hidden Markov Model

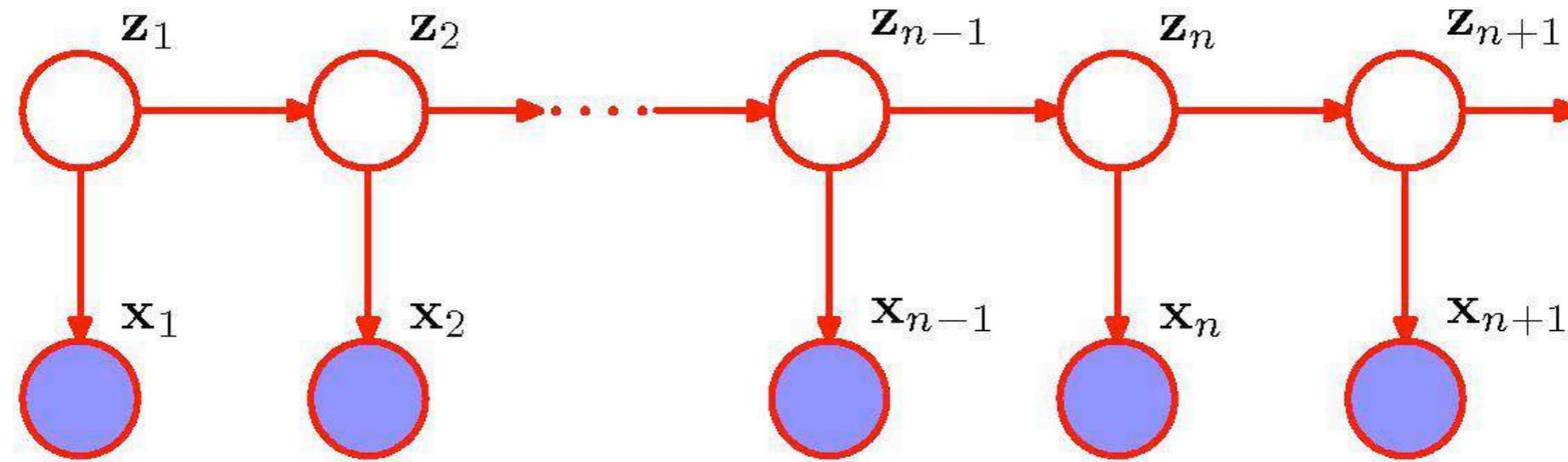
- Applications:
  - ▶ Speech recognition
    - inputs: observed Fourier coefficient  
→ states: utterances
  - ▶ Bioinformatics
    - inputs: raw DNA sequences  
→ states: protein-coding region or not
  - ▶ Communication
    - inputs: raw bit stream  
→ states: correlated bit stream
  - ▶ Tracking:
    - inputs: raw sensory data  
→ states: location, velocity, pose...

# Hidden Markov Model

- HMM learning and inference!
- Three main questions:
  - **probability of an observed data sequence (model is known)**
    - what's the probability of going walking, going shopping, and do cleaning in 3 consecutive days (observation sequence)?
  - **learning the model parameters**
    - there are several activity recordings of consecutive days (the length might vary), what are the most possible transition probabilities, emission probabilities, and state probability of first day?
  - **inferring the most likely state sequence (model is known)**
    - given that the activities of 3 consecutive days are: going shopping, do cleaning, and going walking (an observation sequence), what would be most likely weather states of these 3 days (corresponding hidden state sequence)?

# Hidden Markov Model

- **transition probabilities:**  $p(\mathbf{z}_n = k | \mathbf{z}_{n-1} = j) = A_{jk}$   
 $p(\mathbf{z}_1 = k) = \pi_k$
- **emission probabilities:**  $p(\mathbf{x}_n | \mathbf{z}_n, \phi)$  HMM model parameters  $\theta$   
can be:  
Gaussian if  $x$  is continuous;  
conditional probability table if  $x$  is discrete



$$p(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{z}_1, \dots, \mathbf{z}_N) = p(\mathbf{z}_1) \prod_{n=2}^N p(\mathbf{z}_n | \mathbf{z}_{n-1}) \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n)$$

$$\rightarrow p(\mathbf{X}, \mathbf{Z} | \theta) = p(\mathbf{z}_1 | \pi) \prod_{n=2}^N p(\mathbf{z}_n | \mathbf{z}_{n-1}, A) \prod_{n=1}^N p(\mathbf{x}_n | \mathbf{z}_n, \phi)$$

# Hidden Markov Model

**Three problems and three solutions:**

- 👉 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

$$\begin{aligned} p(\mathbf{X}|\theta) &= \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}|\theta) = \sum_{\mathbf{Z}} p(\mathbf{X}|\mathbf{Z}, \theta)p(\mathbf{Z}|\theta) \\ &= \sum_{\mathbf{z}_1, \dots, \mathbf{z}_n} p(\mathbf{z}_1, \mathbf{x}_1) \prod_{n=2}^N p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n) \end{aligned}$$

**all state sequences**

Now the problem#1 is related to sum over “all possible state sequences”.  
We need to consider hidden states anyway,  
can we get some ideas from the problem#3 of inferring hidden states?  
To start with, let us consider a simpler case of estimating a “single” hidden state:)

# Hidden Markov Model

**Three problems and three solutions:**

- 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

estimating a “single” hidden state given observations

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n | \mathbf{X}) = \frac{p(\mathbf{X} | \mathbf{z}_n)p(\mathbf{z}_n)}{p(\mathbf{X})}$$

using the conditional-independence property  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\} \perp\!\!\!\perp \{\mathbf{x}_{n+1}, \dots, \mathbf{x}_N\} \mid \mathbf{z}_n$

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{p(\mathbf{x}_1, \dots, \mathbf{x}_n | \mathbf{z}_n)p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N | \mathbf{z}_n)p(\mathbf{z}_n)}{p(\mathbf{X})}$$

$$= \frac{p(\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{z}_n)p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N | \mathbf{z}_n)}{p(\mathbf{X})} = \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

joint probability of observing all  
of the data up to time  $n$ , and  $\mathbf{z}_n$

conditional probability of all  
future data from time  $n+1$  to  $N$

# Hidden Markov Model

**Three problems and three solutions:**

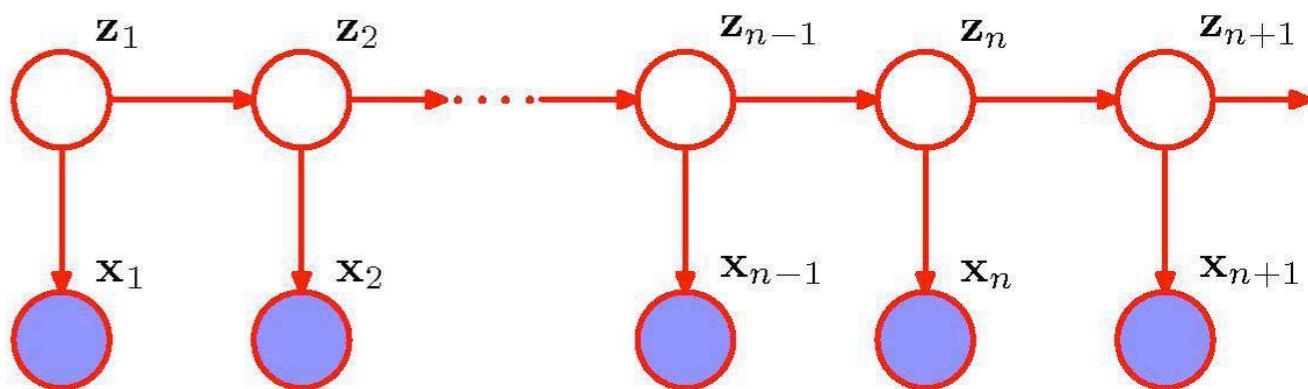
- 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

joint probability of observing all of the data up to time  $n$ , and  $\mathbf{z}_n \rightarrow$  **Forward**

$$\begin{aligned} \underline{\alpha(\mathbf{z}_n)} &= p(\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{z}_n) \\ &\quad \text{conditional independence} \\ &= p(\mathbf{x}_n | \mathbf{z}_n) p(\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{z}_n) \\ &= p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{z}_{n-1}, \mathbf{z}_n) \\ &= p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1}) \\ &= \underline{p(\mathbf{x}_n | \mathbf{z}_n)} \sum_{\mathbf{z}_{n-1}} \underline{\alpha(\mathbf{z}_{n-1})} \underline{p(\mathbf{z}_n | \mathbf{z}_{n-1})} \end{aligned}$$

recursion!



\* note that

$$\begin{aligned} \alpha(\mathbf{z}_N) &= p(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{z}_N) \\ \Rightarrow p(\mathbf{X}) &= \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N) \end{aligned}$$

# Hidden Markov Model

**Three problems and three solutions:**

- 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

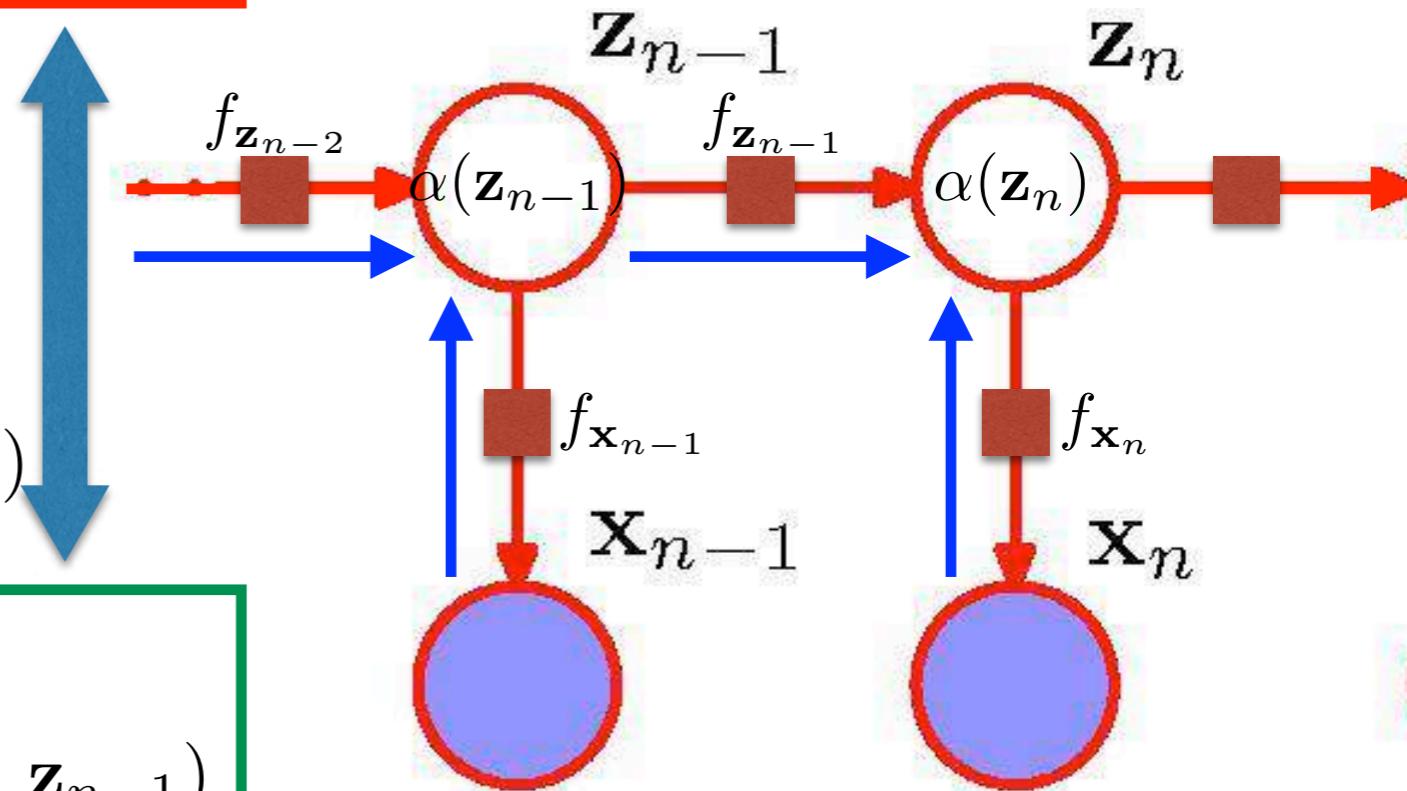
joint probability of observing all of the data up to time  $n$ , and  $\mathbf{z}_n \rightarrow$  **Forward**

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1})$$

$$\mu_{\mathbf{z}_{n-1} \rightarrow f_{\mathbf{z}_{n-1}}} = \alpha(\mathbf{z}_{n-1})$$

$$\begin{aligned} \mu_{f_{\mathbf{z}_{n-1}} \rightarrow \mathbf{z}_n} &= \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) f_{\mathbf{z}_{n-1}} \\ &= \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1}) \end{aligned}$$

$$\begin{aligned} \alpha(\mathbf{z}_n) &= \mu_{f_{\mathbf{x}_n} \rightarrow \mathbf{z}_n} \times \mu_{f_{\mathbf{z}_{n-1}} \rightarrow \mathbf{z}_n} \\ &= p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1}) \end{aligned}$$



remember our factor graph story? 261

# Hidden Markov Model

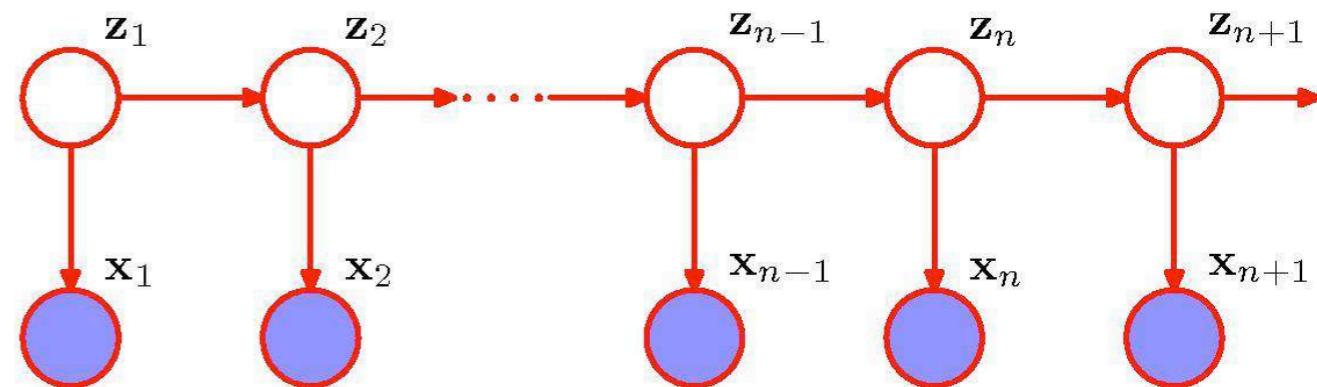
**Three problems and three solutions:**

- 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

conditional probability of all future data from time  $n+1$  to  $N \rightarrow$  Backward

$$\begin{aligned} \underline{\beta(\mathbf{z}_n)} &= p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N | \mathbf{z}_n) \\ &= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N, \mathbf{z}_{n+1} | \mathbf{z}_n) \\ &= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N | \mathbf{z}_{n+1}, \mathbf{z}_n) p(\mathbf{z}_{n+1} | \mathbf{z}_n) \\ &= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n) \\ &= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+2}, \dots, \mathbf{x}_N | \mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n) \\ &= \sum_{\mathbf{z}_{n+1}} \underline{\beta(\mathbf{z}_{n+1})} \underline{p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1})} \underline{p(\mathbf{z}_{n+1} | \mathbf{z}_n)} \end{aligned}$$



# Hidden Markov Model

**Three problems and three solutions:**

- 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

conditional probability of all future data from time  $n+1$  to  $N \rightarrow$  Backward

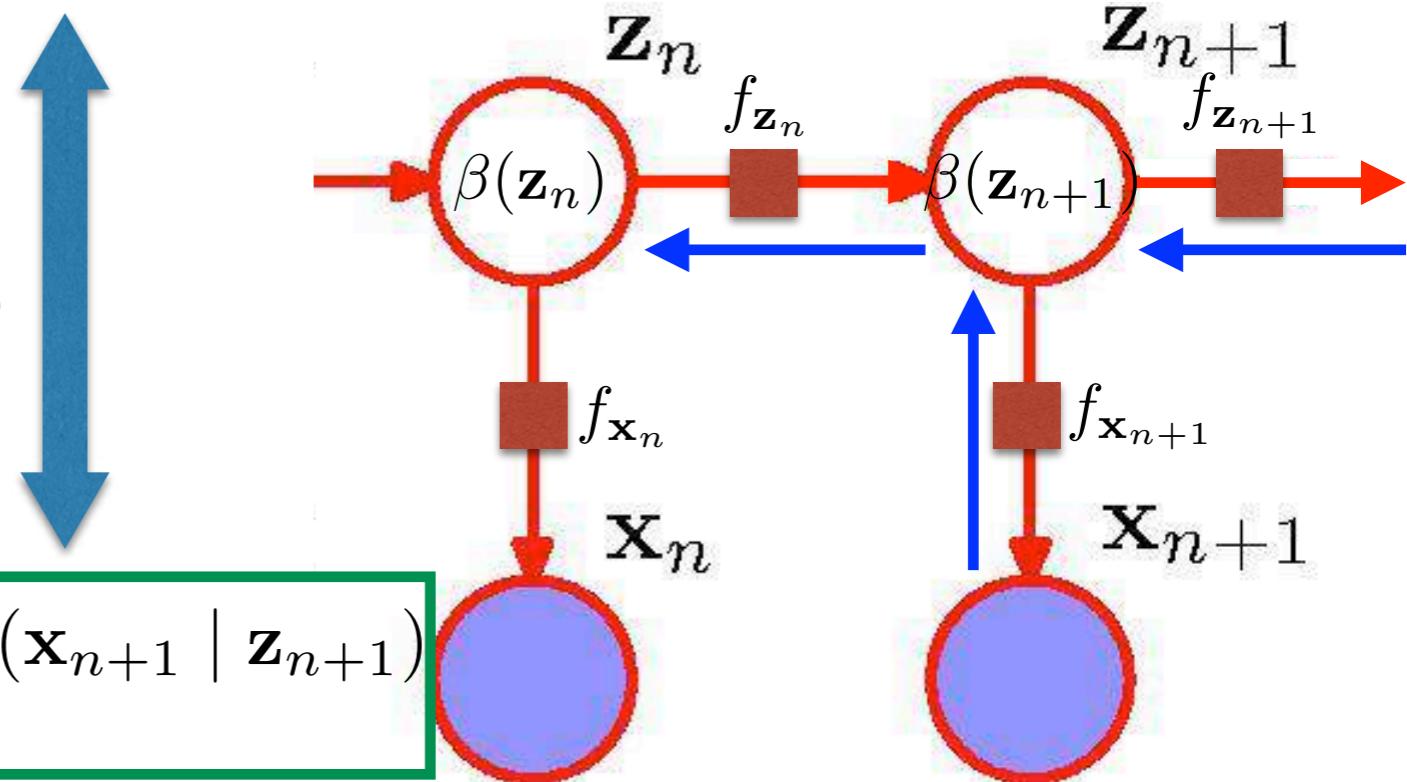
$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n)$$

$$\begin{aligned} \mu_{\mathbf{z}_{n+1} \rightarrow f_{\mathbf{z}_n}} &= \beta(\mathbf{z}_{n+1}) \times \mu_{f_{\mathbf{x}_{n+1}} \rightarrow \mathbf{z}_{n+1}} \\ &= \underline{\beta(\mathbf{z}_{n+1})} p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) \end{aligned}$$

what variable  $\mathbf{z}_{n+1}$  receives from right-hand side

$$\mu_{f_{\mathbf{z}_n} \rightarrow \mathbf{z}_n} = \sum_{\mathbf{z}_{n+1}} f_{\mathbf{z}_n} \mu_{\mathbf{z}_{n+1} \rightarrow f_{\mathbf{z}_n}}$$

$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} p(\mathbf{z}_{n+1} | \mathbf{z}_n) \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1})$$



remember our factor graph story? 263

# Hidden Markov Model

**Three problems and three solutions:**

- 1. Computing probabilities of observed sequences: Forward-backward algorithm
- 2. Learning of parameters: Baum-Welch algorithm
- 3. Inference of hidden state sequences: Viterbi algorithm

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

conditional probability of all future data from time  $n+1$  to  $N \rightarrow$  Backward

$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n)$$

\* note that

$$p(\mathbf{z}_N | \mathbf{X}) = \frac{\alpha(\mathbf{z}_N) \beta(\mathbf{z}_N)}{p(\mathbf{X})}$$

$$= \frac{p(\mathbf{X}, \mathbf{z}_N) \beta(\mathbf{z}_N)}{p(\mathbf{X})}$$

$\Rightarrow$  initialize  $\beta(\mathbf{z}_N) = 1$

to make Bayesian holds  $p(\mathbf{z}_N | \mathbf{X}) = \frac{p(\mathbf{X}, \mathbf{z}_N)}{p(\mathbf{X})}$

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

$$p(\mathbf{z}_n | \mathbf{X}) = \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

**Forward**  $\alpha(\mathbf{z}_n) = p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1})$

**Backward**  $\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} | \mathbf{z}_{n+1}) p(\mathbf{z}_{n+1} | \mathbf{z}_n)$

since  $\sum_{\mathbf{z}_n} \gamma(\mathbf{z}_n) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{X}) = 1 = \sum_{\mathbf{z}_n} \frac{\alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)}{p(\mathbf{X})} \rightarrow p(\mathbf{X} | \theta) = \sum_{\mathbf{z}_n} \alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)$

sum over all possible  $\mathbf{z}_n$   
will result to 1

problem#1 can be resolved  
by forward-backward algo.,  
and we also see that both  
forward-backward steps are  
linked to message passing  
→ forward-backward algo.  
is a special case of  
belief propagation!

# Hidden Markov Model

Three problems and three solutions:

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

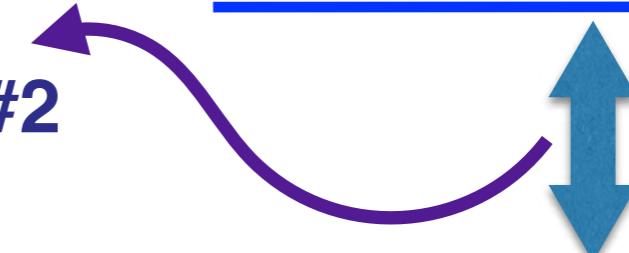
Monitoring difference between these two quantities is a good way to check for convergence during Baum-Welch algorithm for problem#2

remember this note few slides ago when we introduce forward step?

→  $p(\mathbf{X} | \theta)$  can be computed based on final time  $n = N$  (since  $\beta(z_N)=1$ )

→  $p(\mathbf{X} | \theta)$  obtained on arbitrary time  $n$

$$p(\mathbf{X} | \theta) = \sum_{\mathbf{z}_n} \alpha(\mathbf{z}_n) \beta(\mathbf{z}_n)$$



\* note that

$$\alpha(\mathbf{z}_N) = p(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{z}_N)$$

$$\Rightarrow p(\mathbf{X}) = \sum_{\mathbf{z}_N} \alpha(\mathbf{z}_N)$$

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**cross-time statistics for adjacent time steps, which is  
joint posterior distribution of two successive latent variables,  
going to need this later on:**

$$\begin{aligned}\xi(\mathbf{z}_{n-1}, \mathbf{z}_n) &= p(\mathbf{z}_{n-1}, \mathbf{z}_n | \mathbf{X}) \\ &= \frac{p(\mathbf{X} | \mathbf{z}_{n-1}, \mathbf{z}_n) p(\mathbf{z}_{n-1}, \mathbf{z}_n)}{p(\mathbf{X})} \\ &= \frac{p(\mathbf{x}_1, \dots, \mathbf{x}_{n-1} | \mathbf{z}_{n-1}) p(\mathbf{x}_n | \mathbf{z}_n) p(\mathbf{x}_{n+1}, \dots, \mathbf{x}_N | \mathbf{z}_n) p(\mathbf{z}_n | \mathbf{z}_{n-1}) p(\mathbf{z}_{n-1})}{p(\mathbf{X})} \\ &= \frac{\alpha(\mathbf{z}_{n-1}) p(\mathbf{x}_n | \mathbf{z}_n) p(\mathbf{z}_n | \mathbf{z}_{n-1}) \beta(\mathbf{z}_n)}{p(\mathbf{X})}.\end{aligned}$$

obtained by also  
forward-backward

**K-by-K matrix (K: #states) with elements  $(i, j)$  representing  
the expected number of transitions from state  $i$  to state  $j$   
that begin at time  $n-1$ , given all the observations X.**

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**Now take a look of problem#3, find state sequence that best explains observations:**

$$\begin{aligned} \operatorname{argmax}_{\mathbf{Z}} p(\mathbf{Z} \mid \mathbf{X}; \theta) &\Rightarrow \operatorname{argmax}_{\mathbf{Z}} \frac{p(\mathbf{Z}, \mathbf{X}; \theta)}{p(\mathbf{X}; \theta)} \Rightarrow \operatorname{argmax}_{\mathbf{Z}} \frac{p(\mathbf{Z}, \mathbf{X}; \theta)}{\sum_{\mathbf{Z}} p(\mathbf{Z}, \mathbf{X}; \theta)} \\ &\Rightarrow \operatorname{argmax}_{\mathbf{Z}} p(\mathbf{Z}, \mathbf{X}; \theta) \end{aligned}$$

**Viterbi algorithm defines:**

$$\delta_j(n) = \max_{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{n-1}} p(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{n-1}, \mathbf{z}_n = j, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n; \theta)$$

**i.e. the highest probability of a single state-path of length  $n$  which accounts for observations  $\mathbf{x}_1 \dots \mathbf{x}_n$  and ends in state  $\mathbf{z}_n = j$**

# Hidden Markov Model



**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**Viterbi algorithm defines:**

$$\delta_j(n) = \max_{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{n-1}} p(\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{n-1}, \mathbf{z}_n = j, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n; \theta)$$

**which gives induction:**

$$\begin{aligned}\delta_j(1) &= p(\mathbf{z}_1 = j) \underbrace{p(\mathbf{x}_1 \mid \mathbf{z}_1 = j, \phi)}_{\text{emission}} = \pi_j p(\mathbf{x}_1 \mid \mathbf{z}_1 = j, \phi) \\ \delta_j(n+1) &= \max_i \delta_i(n) \underbrace{p(\mathbf{z}_{n+1} = j \mid \mathbf{z}_n = i)}_{\text{transition}} \underbrace{p(\mathbf{x}_{n+1} \mid \mathbf{z}_{n+1} = j, \phi)}_{\text{emission}} \\ &= \max_i \delta_i(n) A_{ij} p(\mathbf{x}_{n+1} \mid \mathbf{z}_{n+1} = j, \phi)\end{aligned}$$

$$\psi_j(n) = \operatorname{argmax}_i \delta_i(n-1) A_{ij} p(\mathbf{x}_n \mid \mathbf{z}_n = j, \phi)$$

**a helper variable that stores  
the  $n-1$  state index  $i$  on the  
highest probability path**

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm



$$\hat{\mathbf{z}}_N = \operatorname{argmax}_i \delta_i(N)$$



$$p(\hat{\mathbf{Z}}) = \max_i \delta_i(N)$$

**recursive computation from  $n=1$  to  $n=N$   
in order to obtain best state  $\hat{\mathbf{z}}_N$  on time  $N$   
according to highest probability state-path**

$$\hat{\mathbf{z}}_n = \psi_{\hat{\mathbf{z}}_{n+1}}(n+1)$$



**after forward recursive computation, walking  
backwards to get the most likely state-path**

**problem#3 can be resolved  
by Viterbi algorithm  
→ actually it is a special case  
of max-sum algorithm!**

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: *Forward-backward algorithm*
2. Learning of parameters: *Baum-Welch algorithm*
3. Inference of hidden state sequences: *Viterbi algorithm*

Now take a look of problem#2, given only observation sequences, need to find the **model** that is most likely to produce the sequences.

→ intuition: if we knew the true **state path** then maximum-likelihood estimation for model parameters would be trivial

→ **EM** (expectation-maximization) algorithm, iterate over **model** and **path**

→ known as **Baum-Welch algorithm** (EM applied to HMMs)

**E-step: compute posterior distribution over state path**

$$\underline{p(\mathbf{Z}|\mathbf{X}, \theta^{old})}$$

**M-step: maximize expected complete data log-likelihood  
(model parameter re-estimation)**

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{Z}} \underline{p(\mathbf{Z}|\mathbf{X}, \theta^{old})} \log p(\mathbf{X}, \mathbf{Z}|\theta)$$

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: *Forward-backward algorithm*
2. Learning of parameters: *Baum-Welch algorithm*
3. Inference of hidden state sequences: *Viterbi algorithm*

**E-step: compute posterior distribution over state path**

$$\underline{p(\mathbf{Z}|\mathbf{X}, \theta^{old})}$$

$$\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X}) = \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

$$\xi(\mathbf{z}_{n-1}, \mathbf{z}_n) = p(\mathbf{z}_{n-1}, \mathbf{z}_n|\mathbf{X}) = \frac{\alpha(\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n|\mathbf{z}_{n-1})\beta(\mathbf{z}_n)}{p(\mathbf{X})}$$

**forward-backward algorithm  
can be easily used here:)**

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**M-step: maximize expected complete data log-likelihood**

$$Q(\theta, \theta^{old}) = \sum_{\mathbf{Z}} \underline{p(\mathbf{Z}|\mathbf{X}, \theta^{old})} \log p(\mathbf{X}, \mathbf{Z}|\theta)$$

since  $\log p(\mathbf{X}, \mathbf{Z}|\theta) = \log \left[ p(\mathbf{z}_1|\boldsymbol{\pi}) \prod_{n=2}^N p(\mathbf{z}_n|\mathbf{z}_{n-1}, A) \prod_{n=1}^N p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\phi}) \right]$

$$= \sum_{k=1}^K z_{1k} \log \pi_k + \sum_{n=2}^N \sum_{k=1}^K \sum_{j=1}^K [z_{n-1,j} z_{nk}] \log A_{jk} + \sum_{n=1}^N \sum_{k=1}^K \boxed{z_{nk}} \log p(\mathbf{x}_n|\mathbf{z}_n)$$

**probability of  $\mathbf{z}_n = k$**

then  $Q(\theta, \theta^{old}) = \sum_{k=1}^K \gamma(z_{1k}) \log \pi_k + \sum_{n=2}^N \sum_{k=1}^K \sum_{j=1}^K \xi(z_{n-1,j} z_{nk}) \log A_{jk}$

$+ \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \log p(\mathbf{x}_n|\mathbf{z}_n)$

**recall that**  $\gamma(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{X})$   
 $\xi(\mathbf{z}_{n-1}, \mathbf{z}_n) = p(\mathbf{z}_{n-1}, \mathbf{z}_n|\mathbf{X})$

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**M-step:** we maximize  $Q(\theta, \theta^{\text{old}})$  w.r.t.  $\theta = \{\pi, A, \phi\}$

in which we treat  $\gamma(\mathbf{z}_n)$  and  $\xi(\mathbf{z}_{n-1}, \mathbf{z}_n)$  as constant

$$Q(\theta, \theta^{\text{old}}) = \sum_{k=1}^K \gamma(z_{1k}) \log \pi_k + \sum_{n=2}^N \sum_{k=1}^K \sum_{j=1}^K \xi(z_{n-1,j}, z_{nk}) \log A_{jk} \\ + \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \log p(\mathbf{x}_n | \mathbf{z}_n)$$

$$\rightarrow \pi_k^{\text{new}} = \frac{\gamma(z_{1k})}{\sum_{j=1}^K \gamma(z_{1j})} \quad A_{jk}^{\text{new}} = \frac{\sum_{n=2}^N \xi(z_{n-1,j}, z_{nk})}{\sum_{l=1}^K \sum_{n=2}^N \xi(z_{n-1,j}, z_{nl})}$$

# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**M-step:** we maximize  $Q(\theta, \theta^{\text{old}})$  w.r.t.  $\theta = \{\pi, A, \phi\}$

in which we treat  $\gamma(\mathbf{z}_n)$  and  $\xi(\mathbf{z}_{n-1}, \mathbf{z}_n)$  as constant

- For the case of **discrete multinomial observed variables**, the observation model takes the form:

$$p(\mathbf{x}_n | \mathbf{z}_n, \phi) = \prod_{i=1}^D \prod_{k=1}^K \mu_{ik}^{x_{ni} z_{nk}}.$$

Same as fitting Bernoulli mixture model.

- And the corresponding M-step update:  $\mu_{ik}^{\text{new}} = \frac{\sum_{n=1}^N \gamma(z_{nk}) x_{ni}}{\sum_{n=1}^N \gamma(z_{nk})}$ .



# Hidden Markov Model

**Three problems and three solutions:**

1. Computing probabilities of observed sequences: Forward-backward algorithm
2. Learning of parameters: Baum-Welch algorithm
3. Inference of hidden state sequences: Viterbi algorithm

**M-step:** we maximize  $Q(\theta, \theta^{\text{old}})$  w.r.t.  $\theta = \{\pi, A, \phi\}$

in which we treat  $\gamma(\mathbf{z}_n)$  and  $\xi(\mathbf{z}_{n-1}, \mathbf{z}_n)$  as constant

- For the case of the **Gaussian emission model**:

$$p(\mathbf{x}_n | \mathbf{z}_n, \phi) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}.$$

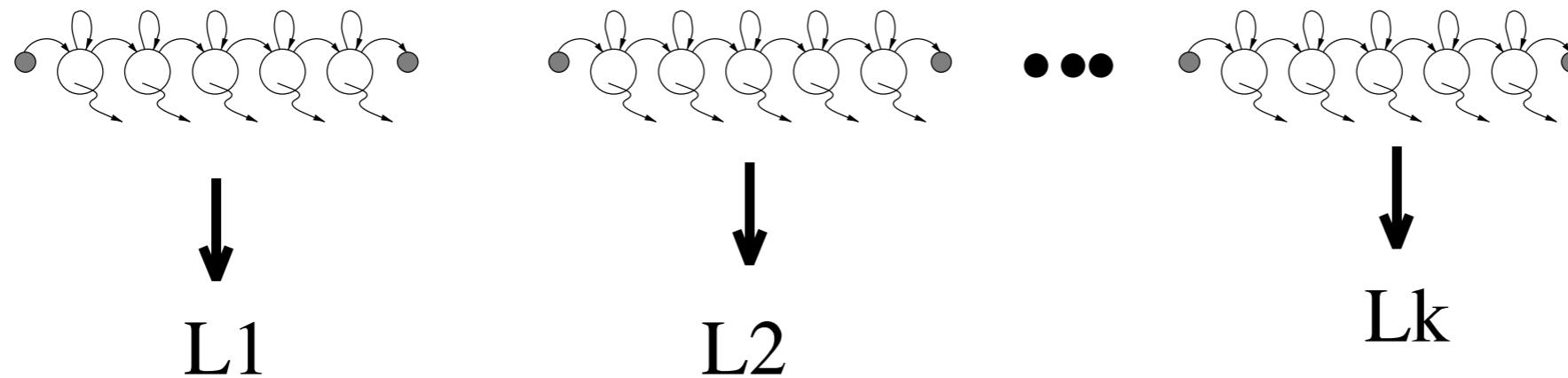
- And the corresponding M-step updates: Same as fitting a Gaussian mixture model.

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_n \gamma(z_{nk}) \mathbf{x}_n, \quad N_k = \sum_n \gamma(z_{nk}),$$

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T,$$

# Hidden Markov Model

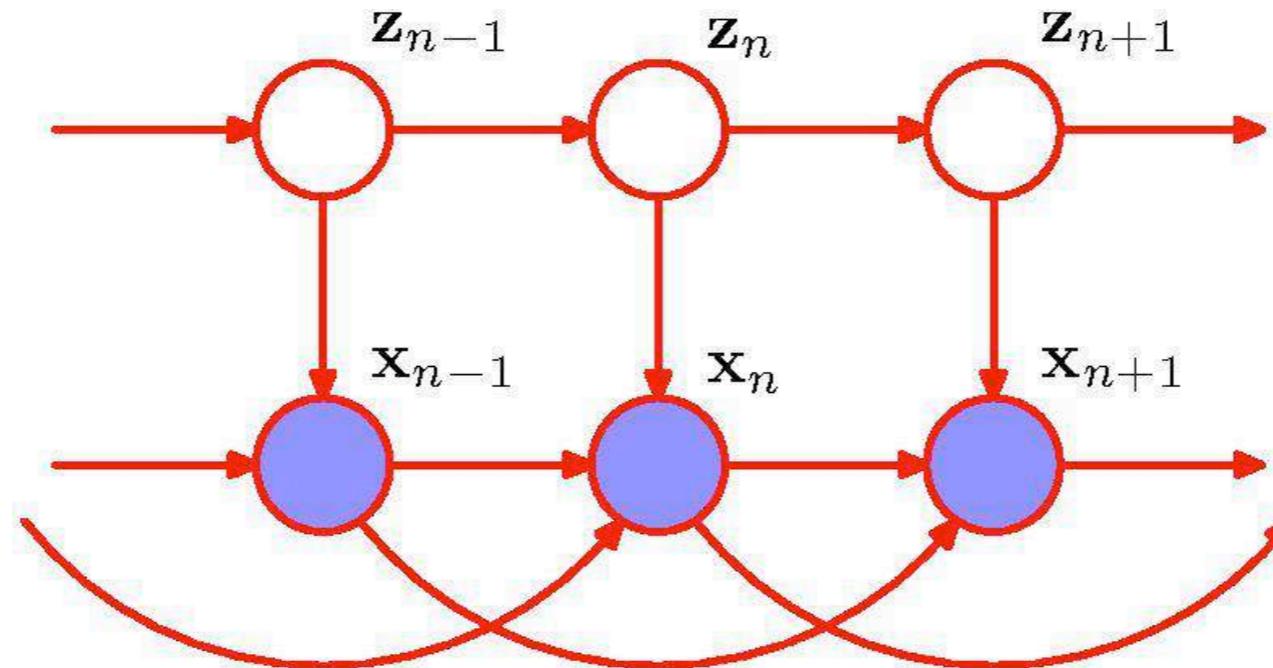
- We can use HMMs for recognition by:
  - Training one HMM for each class (requires labelled training data)
  - Evaluating the probability of an unknown sequence under each HMM
  - Classifying the unknown sequence by choosing an HMM with highest likelihood



- This requires the solution of two problems:
  - Given a model, **evaluate the probability of a sequence.**  
(We can do this exactly and efficiently.)
  - Given some training sequences, **estimate the model parameters.**  
(We can find the local maximum using EM.)

# Hidden Markov Model

- One limitation of the standard HMM is that it is poor at **capturing long-range correlations** between observations, as these have to be mediated via the first order Markov chain of hidden states.



- **Autoregressive HMM:** The distribution over  $x_n$  depends also on a subset of previous observations.
- The number of additional links must be limited to avoid an excessive number of free parameters.
- The **graphical model framework** motivates a number of different models based on HMMs.

# Motivations Again For Probabilistic Graphical Models

Take human body pose estimation as example

- Body is represented as **flexible configuration of body parts**

posterior over body poses

$$p(L|E) \propto p(E|L)p(L)$$

likelihood of observations

prior on body poses

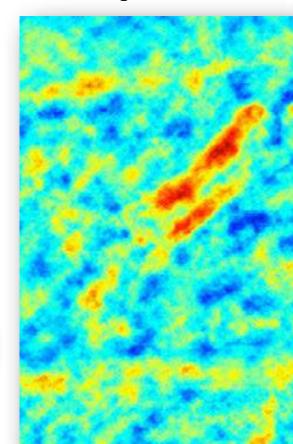
orientation K



Local Features

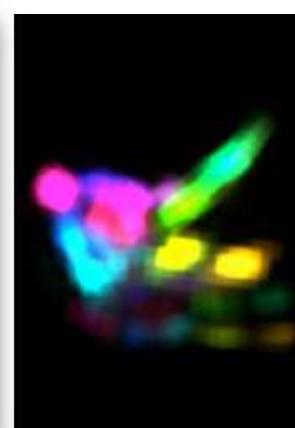
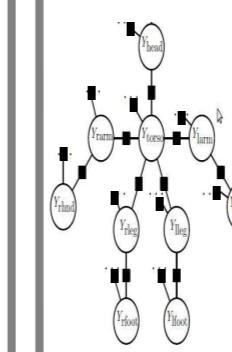
AdaBoost

likelihood of part N



estimated pose

part posteriors



# Motivations Again For Probabilistic Graphical Models

## [prior]

- Notation

- from [Andriluka,Roth,Schiele@ijcv11]
- body configuration:

$$L = \{l_0, l_1, \dots, l_N\}$$

- each body part:  $l_i = (\underline{x_i, y_i, \theta_i, s_i})$   
**position, rotation, and scale**

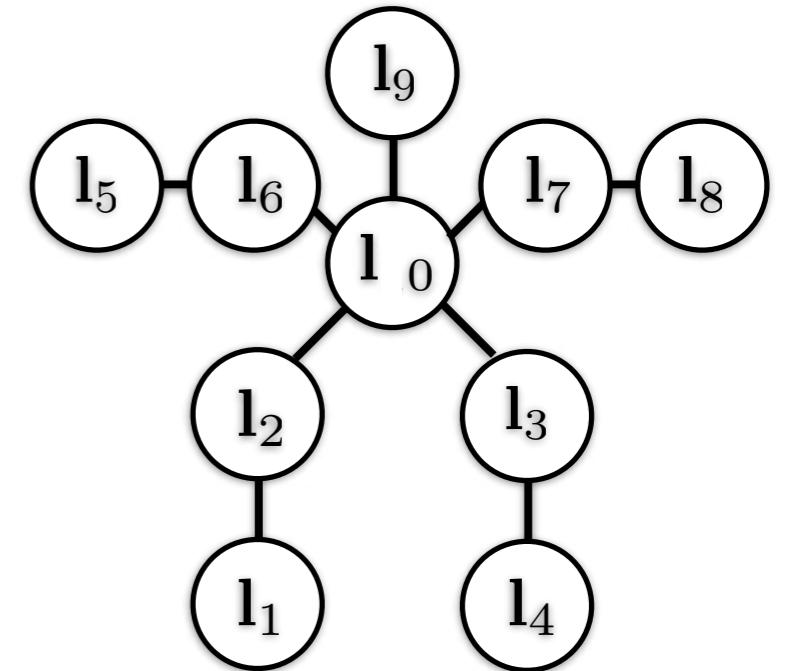
- Prior:

$$p(L) = p(l_0) \prod_{(i,j) \in G} p(l_i | l_j)$$

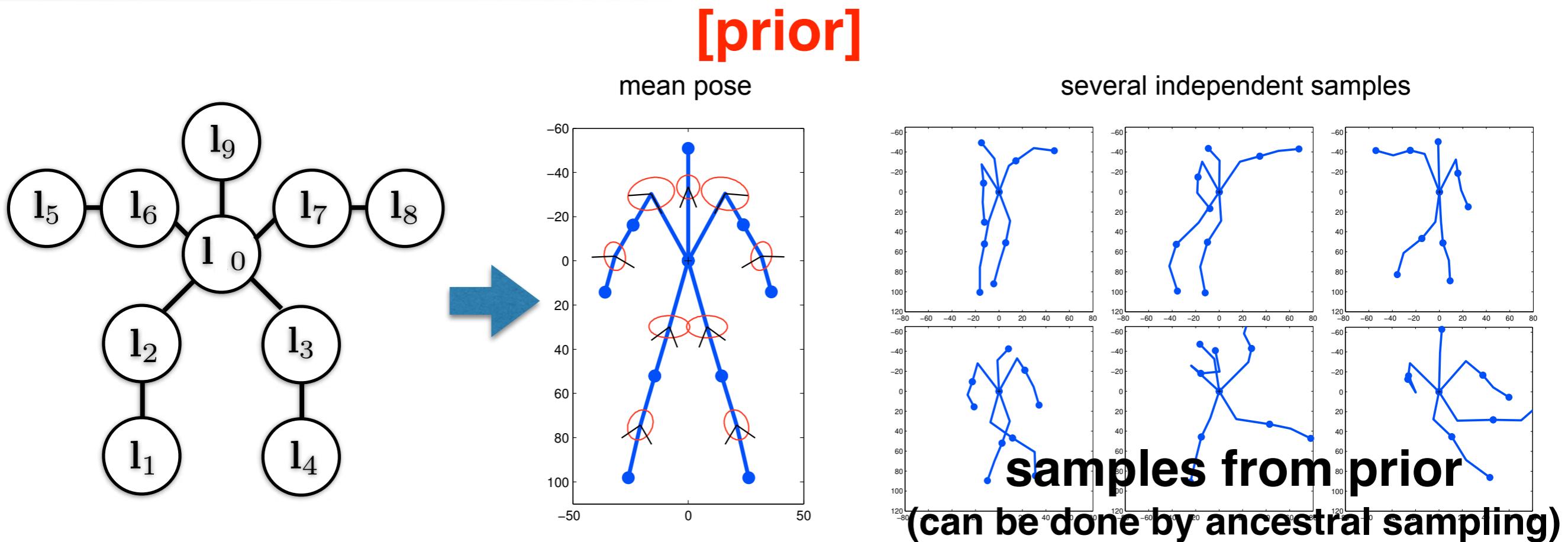
a certain body part can appear anywhere

- with  $p(l_0)$  assumed uniform **with different scales and rotations**
- with  $p(l_i | l_j)$  modeled using a Gaussian **given the state of torso, the possible state of head is conditioned**

**Note: if you have many observations (training data),  
you can estimate the parameters of the prior by using maximum likelihood**

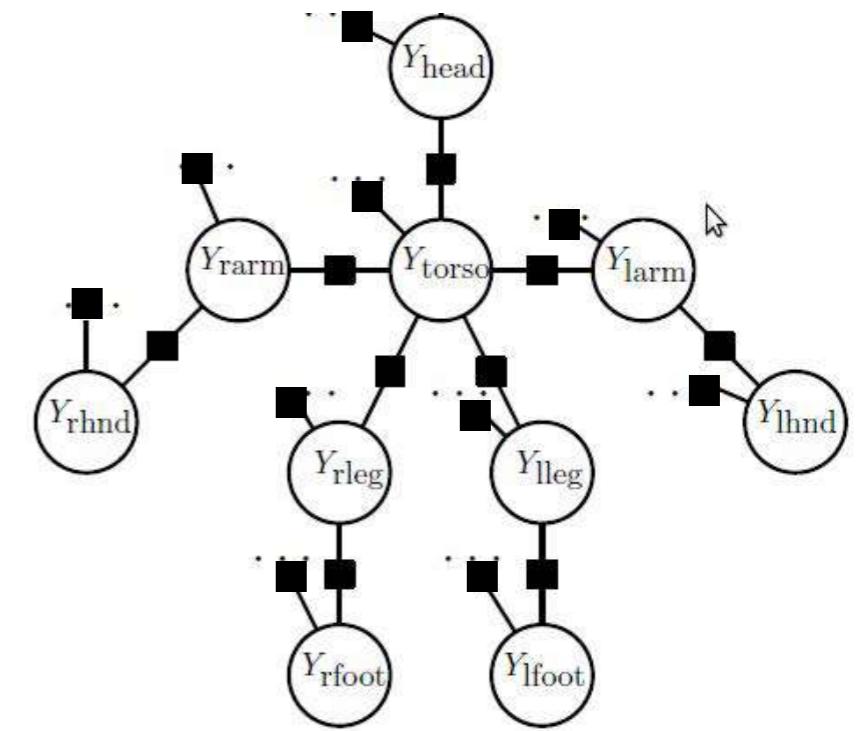


# Motivations Again For Probabilistic Graphical Models



represented as factor graph:)

- potentials (= energies = factors)
  - ▶ unaries for each body part (torso, head, ...)
  - ▶ pairwise between connected body parts



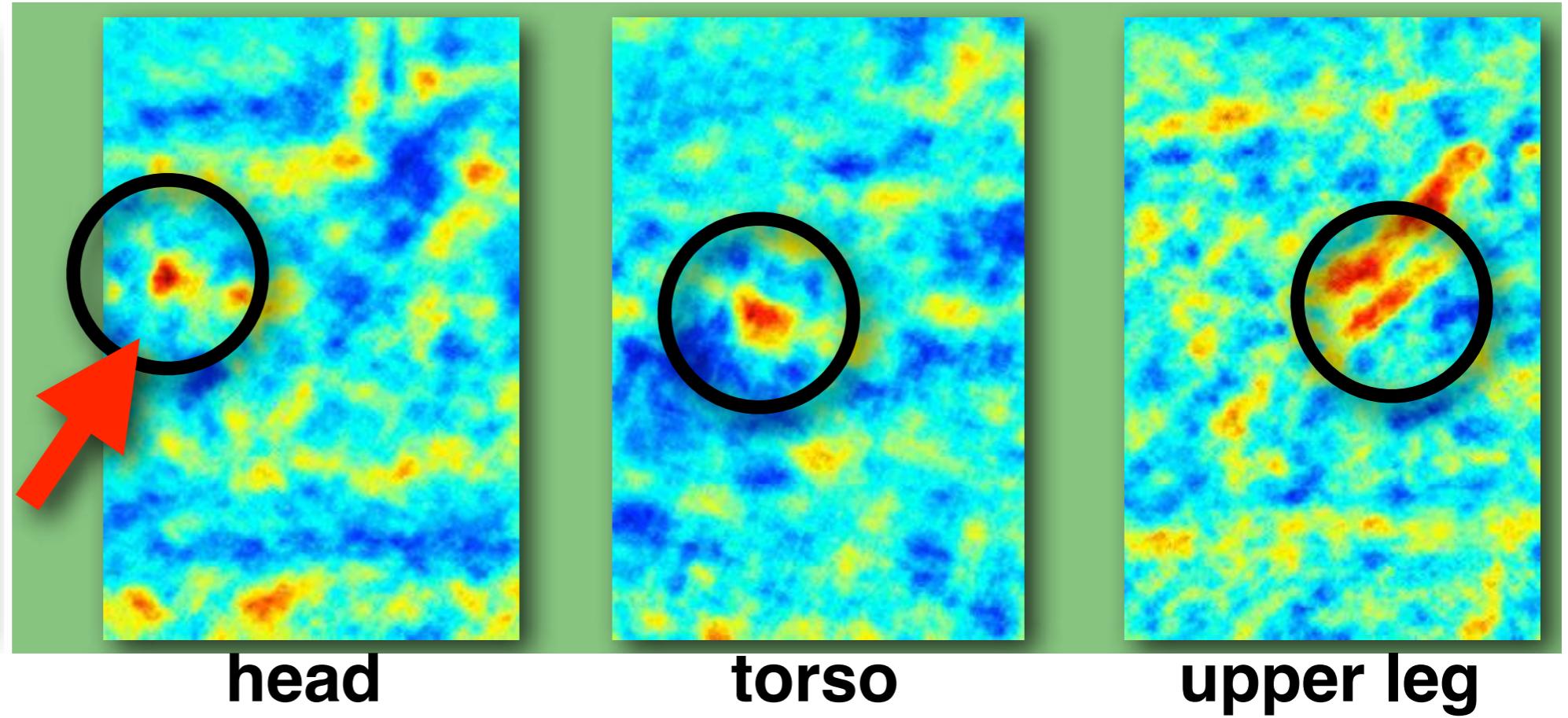
# Motivations Again For Probabilistic Graphical Models

## [likelihood]

- common assumption:
  1. evidence (image features) for each part independent of all other parts
  2. assumption clearly not correct, but allows efficient computation:)

$$p(E|L) = \prod_{i=0}^N p(E|l_i)$$

you can use any the score outputs of any detector:) related to the unary potentials



# Motivations Again For Probabilistic Graphical Models

## [inference]

- body pose estimation

- max-product algorithm (MAP-estimate, best overall configuration) **the pose that best explains the observation**
- sum-product algorithm (marginals of each part)

$$p(L|E) \propto p(L)p(E|L)$$

$$\max_L p(L|E)$$

$$= p(l_0) \prod_{i=1}^N p(l_i|l_0) \prod_{i=0}^N p(e_i|l_i)$$

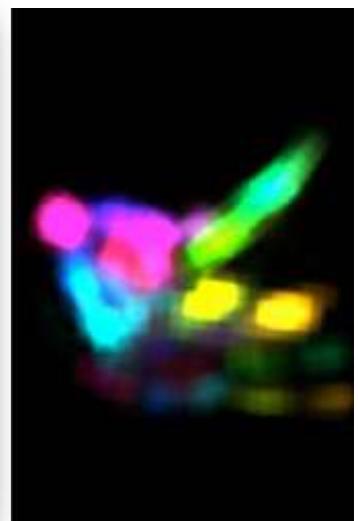
$$= \max_L \prod_{i=0}^N (p(l_i|l_0)p(e_i|l_i))$$

$$= \prod_{i=0}^N p(l_i|l_0) \prod_{i=0}^N p(e_i|l_i)$$



$$= \min_L \sum_{i=0}^N (-\ln p(l_i|l_0) - \ln p(e_i|l_i))$$

**what you have learnt  
max-product → max-sum**



estimated pose part posteriors

# Motivations Again For Probabilistic Graphical Models

- Remember: variable to factor message:

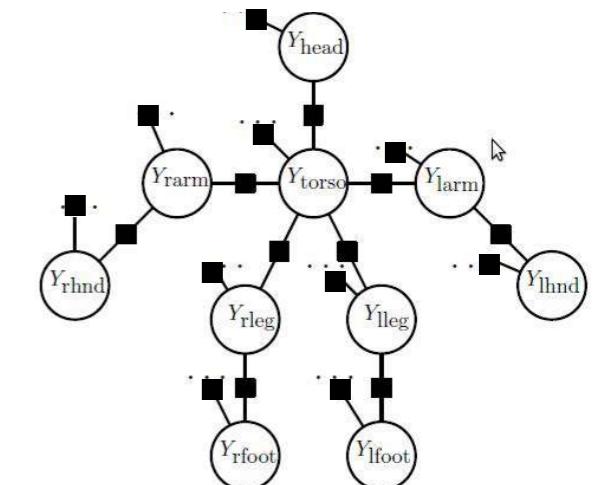
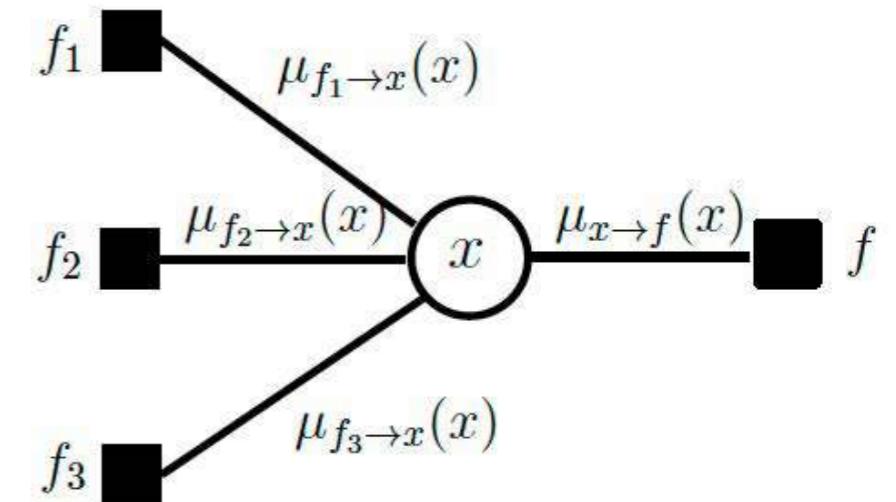
$$\mu_{x \rightarrow f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x)$$

- when using  $-\ln(p(\cdot))$  rather than  $p(\cdot)$  directly:

$$\mu_{x \rightarrow f}(x) = \sum_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \rightarrow x}(x)$$

- in our case:

- $x$  corresponds to  $l_j$ ,
- $f$  corresponds to  $d_{ij}$



$$\mu_{l_j \rightarrow d_{ij}}(l_j) = \sum_{g \in \{\text{ne}(l_j) \setminus d_{ij}\}} \mu_{g \rightarrow l_j}(l_j) = \frac{m_j(l_j)}{\text{unary}} + \sum_{c \in C_j} \frac{B_c(l_j)}{\text{pairwise}}$$

we can use exact inference algo. (max-sum/msg passing) here  
because this is a tree-structured graph!

# Motivations Again For Probabilistic Graphical Models

- Another example in image denoising

$$p(\text{true image}|\text{noisy image}) = p(\mathbf{T}|\mathbf{N})$$

likelihood of noisy given true image  
(observation model)

image prior for all true images

$$p(\mathbf{T}|\mathbf{N}) = \frac{p(\mathbf{N}|\mathbf{T}) \cdot p(\mathbf{T})}{p(\mathbf{N})}$$

posterior

normalization term (constant)



# Motivations Again For Probabilistic Graphical Models

- Another example in image denoising

$$p(\text{true image}|\text{noisy image}) = p(\mathbf{T}|\mathbf{N})$$

likelihood of noisy given true image  
(observation model)

image prior for all true images

$$p(\mathbf{T}|\mathbf{N}) = \frac{p(\mathbf{N}|\mathbf{T}) \cdot p(\mathbf{T})}{p(\mathbf{N})}$$

posterior

normalization term (constant)

[likelihood]

- Simplification: assume that the noise at one pixel is independent of the others.

$$p(\mathbf{N}|\mathbf{T}) = \prod_{i,j} p(N_{i,j}|T_{i,j})$$

- Then we will assume that the noise at each pixel is additive and Gaussian distributed:

$$p(N_{i,j}|T_{i,j}) = \mathcal{N}(N_{i,j} - T_{i,j} | 0, \sigma^2)$$

- The variance  $\sigma^2$  controls the amount of noise.

# Motivations Again For Probabilistic Graphical Models

- Another example in image denoising

$$p(\text{true image}|\text{noisy image}) = p(\mathbf{T}|\mathbf{N})$$

likelihood of noisy given true image  
(observation model)

image prior for all true images

$$p(\mathbf{T}|\mathbf{N}) = \frac{p(\mathbf{N}|\mathbf{T}) \cdot p(\mathbf{T})}{p(\mathbf{N})}$$

posterior

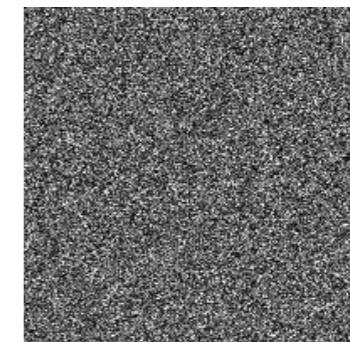
normalization term (constant)

[prior]

- We want the prior to describe how probable it is (a-priori) to have a particular true image among the set of all possible images.



probable



improbable

# Motivations Again For Probabilistic Graphical Models

- Another example in image denoising

$$p(\text{true image}|\text{noisy image}) = p(\mathbf{T}|\mathbf{N})$$

likelihood of noisy given true image  
(observation model)

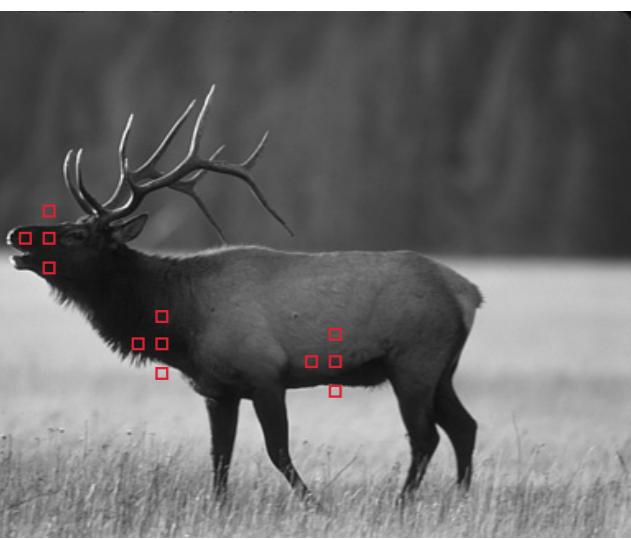
image prior for all true images

$$p(\mathbf{T}|\mathbf{N}) = \frac{p(\mathbf{N}|\mathbf{T}) \cdot p(\mathbf{T})}{p(\mathbf{N})}$$

posterior

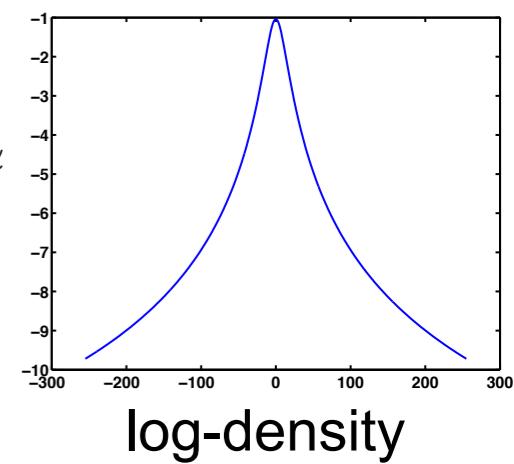
normalization term (constant)

[prior]



- Nearby pixels often have similar intensity:
- But sometimes there are large intensity changes. **edge/boundary**
  - We need discontinuity-preserving potentials:
    - One possibility: Student-t distribution.

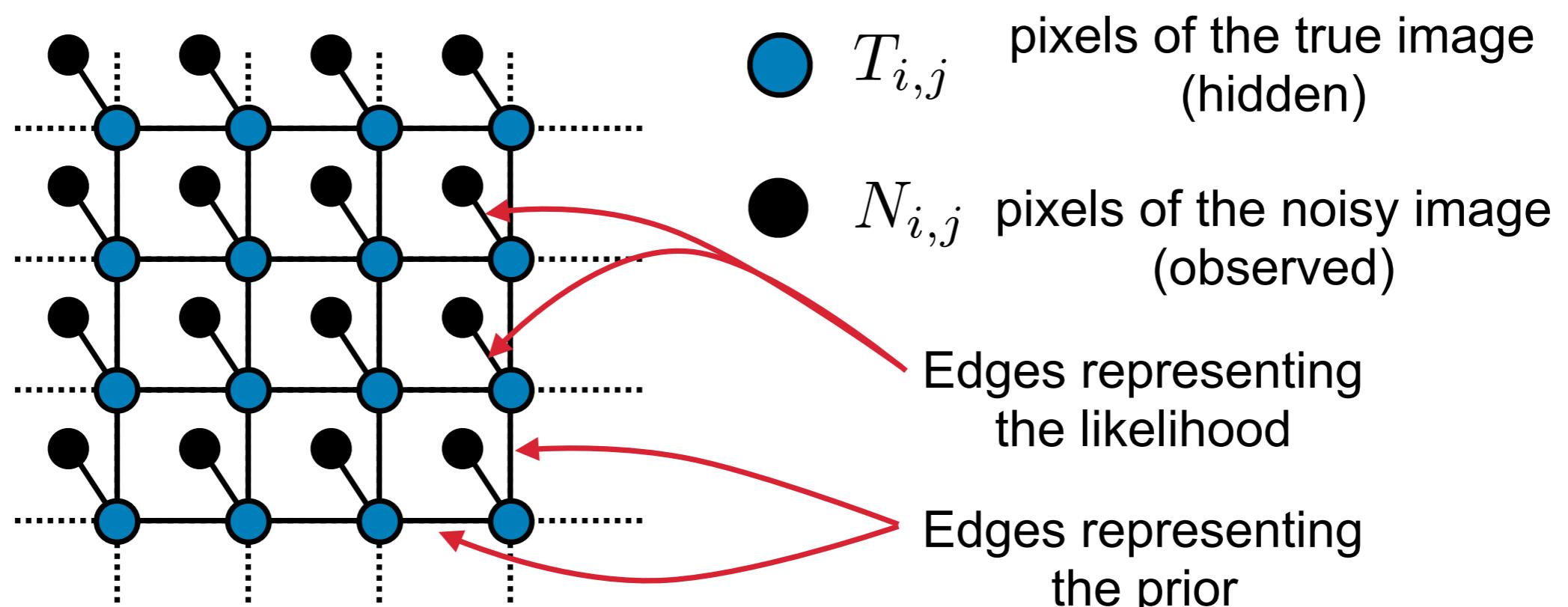
$$f_H(T_{i,j}, T_{i+1,j}) = \left( 1 + \frac{1}{2\sigma^2} (T_{i,j} - T_{i+1,j})^2 \right)^{-\alpha}$$



# Motivations Again For Probabilistic Graphical Models

## [inference]

- We can now put the likelihood and the prior together in a single MRF model:



$$p(\mathbf{T}|\mathbf{N}) \propto p(\mathbf{N}|\mathbf{T})p(\mathbf{T})$$

actually in this problem you can also use continuous optimization by gradient decent

$$= \left( \prod_{i,j} p(N_{i,j}|T_{i,j}) \right) \left( \prod_{i,j} f_H(T_{i,j}, T_{i+1,j}) \cdot f_V(T_{i,j}, T_{i,j+1}) \right)$$

now the graph is loopy, exact inference is almost impossible  
we need to use approximate inference

# Sampling

- Inference in General Graphs – Approximate Inference
  - For trees, inference usually exact
  - Approximate Inference comes into play whenever exact inference is not tractable, e.g. the model is not tree-structured
- What would we like to approximate?
  - e.g. posterior distribution  $p(z|x)$
  - **Expectations**
    - continuous: integrals may be intractable
    - discrete: sum over exponentially many states → infeasible
- Conceptually there are two approaches
  - Deterministic approximation
  - Numerical Sampling (e.g. **Markov Chain Monte Carlo**)

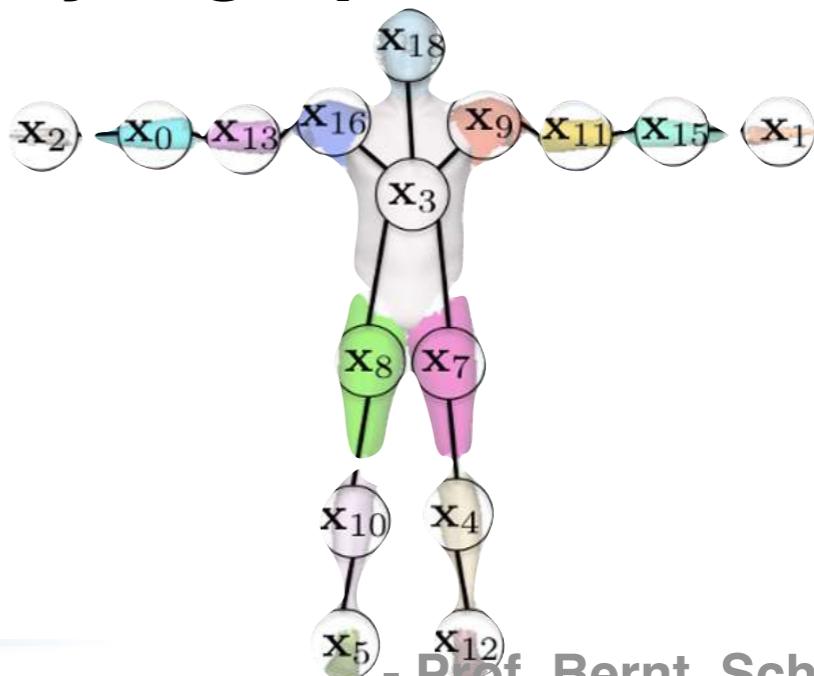
# Sampling

---

- Deterministic approximation  
(e.g. **Variational Inference, Loopy Belief Propagation, Mean Field, Expectation Propagation**)
  - Approximate the quantity of interest
  - Solve the approximation analytically
  - Results depends on the quality of the approximation
- Numerical Sampling  
(e.g. **Markov Chain Monte Carlo, Gibbs Sampling...**)
  - Take the quantity of interest
  - **Use the random samples to approximate it**
  - Results depend on the quality and amount of random samples

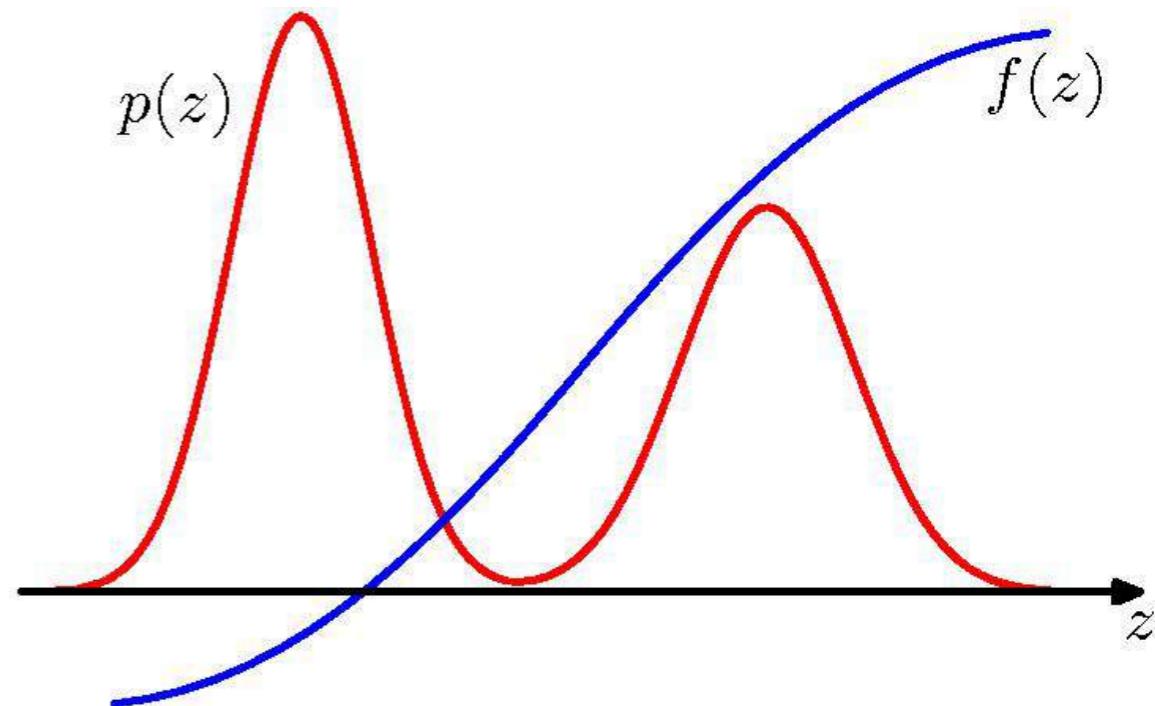
# Sampling - how to draw random samples?

- draw random samples from some distributions  $p(x)$ 
  - discrete or continuous
  - univariate or multi-variate
- For instance, Gaussian, Poisson, Uniform, Dirichlet
  - available in most of programming languages
- More general: what about **sampling from some distributions, e.g. defined by a graphical model?**
  - e.g. a distribution over body parts,  
we want to find likely body poses
  - e.g. a distribution over images,  
we want to look at likely images



# Sampling

- Example: Expectation
- We want to evaluate



$$\mathbb{E}[f] = \int f(x)p(x)dx \quad \text{or} \quad \mathbb{E}[f] = \sum_{x \in \mathcal{X}} f(x)p(x)$$

- Sample idea (idea in **Monte Carlo** estimation):
  - draw L independent samples  $x^1, x^2, \dots, x^L$  from  $p(\cdot)$ :  $x^l \sim p(\cdot)$
  - replace the integral/sum with the finite set of samples

$$\hat{f} = \frac{1}{L} \sum_{l=1}^L f(x^l)$$

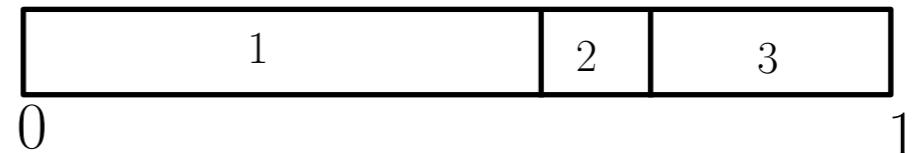
- as long as  $x^l \sim p(\cdot)$  then

$$\mathbb{E}[\hat{f}] = \mathbb{E}[f]$$

# Sampling

- How to sample? A simple case for you to get an idea
- Assume we can draw a value uniformly at random from the unit interval  $[0, 1]$  → How? Pseudo-Random number generator
  - discrete example:
    - consider we want to sample from a univariate discrete distribution  $p$
    - one dimensional and  $K$  states
    - so we have  $p(x = k) = p_k$
    - calculate the **cumulant**  $c_i = \sum_{j \leq i} p_j$
    - Draw  $u \sim [0,1]$
    - Find that  $i$  for which  $c_{i-1} < u < c_i$
    - Return state  $i$  as sample from  $p$

$$p(x) = \begin{cases} 0.6 & x = 1 \\ 0.1 & x = 2 \\ 0.3 & x = 3 \end{cases}$$



# Sampling

- How to sample? A simple case for you to get an idea
- Assume we can draw a value uniformly at random from the unit interval  $[0, 1]$  → How? Pseudo-Random number generator
  - continuous example:
    - extension to continuous variable is clear
    - calculate the **cumulant**  $C(y) = \int_{-\infty}^y p(x)dx$
    - then sample  $u \sim [0,1]$
    - comput  $x = C^{-1}(u)$
    - so sampling is possible if we can compute the integral
      - e.g. Gaussian distribution

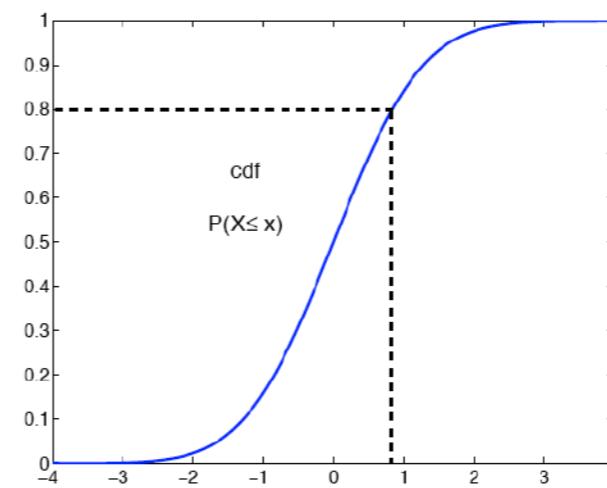
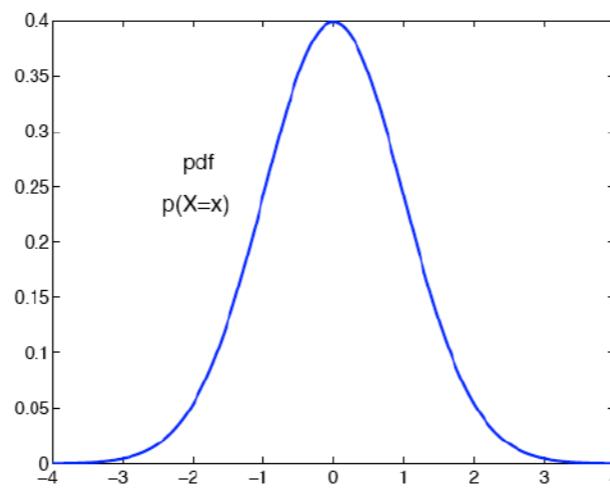
# Sampling

- How to sample? A simple case for you to get an idea
- 1-dimensional Gaussian pdf (probability density function)  $p(x|\mu, \sigma^2)$  and the corresponding cumulative distribution:

$$F_{\mu, \sigma^2}(x) = \int_{-\infty}^x p(z|\mu, \sigma^2) dz$$

- to draw a sample from a Gaussian, we invert the cumulative distribution function:

$$u \sim \text{uniform}(0, 1) \Rightarrow x = F_{\mu, \sigma^2}^{-1}(u) \sim p(x|\mu, \sigma^2)$$



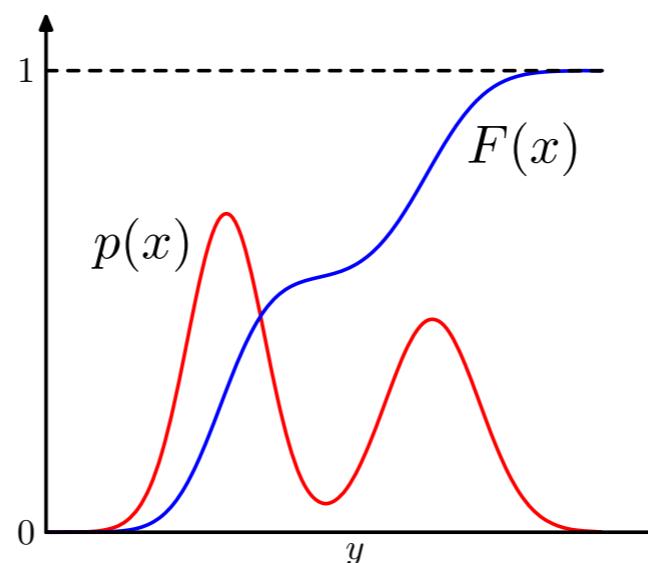
# Sampling

- How to sample? A simple case for you to get an idea
- assume pdf (probability density function)  $p(x)$  and the corresponding cumulative distribution:

$$F(x) = \int_{-\infty}^x p(z)dz$$

- to draw a sample from this pdf, we invert the cumulative distribution function:

$$u \sim \text{uniform}(0, 1) \Rightarrow x = F^{-1}(u) \sim p(x)$$



# Sampling

---

- **Rejection Sampling**
- Ancestral Sampling
- Importance Sampling
- Markov Chain Monte Carlo
- Gibbs Sampling

# Rejection Sampling

- Suppose we want to sample from  $p(x)$  (but that is difficult)
- Furthermore assume we can evaluate  $p(x)$  up to a constant (think of Markov Networks)

$$p(x) = \frac{1}{Z} \tilde{p}(x) = \frac{1}{Z} \prod_c \phi_c(\mathcal{X}_c)$$

- Instead sample from a **proposal distribution**  $q(x)$
- Choose  $q$  such that we can easily sample and a  $k$  exists such that

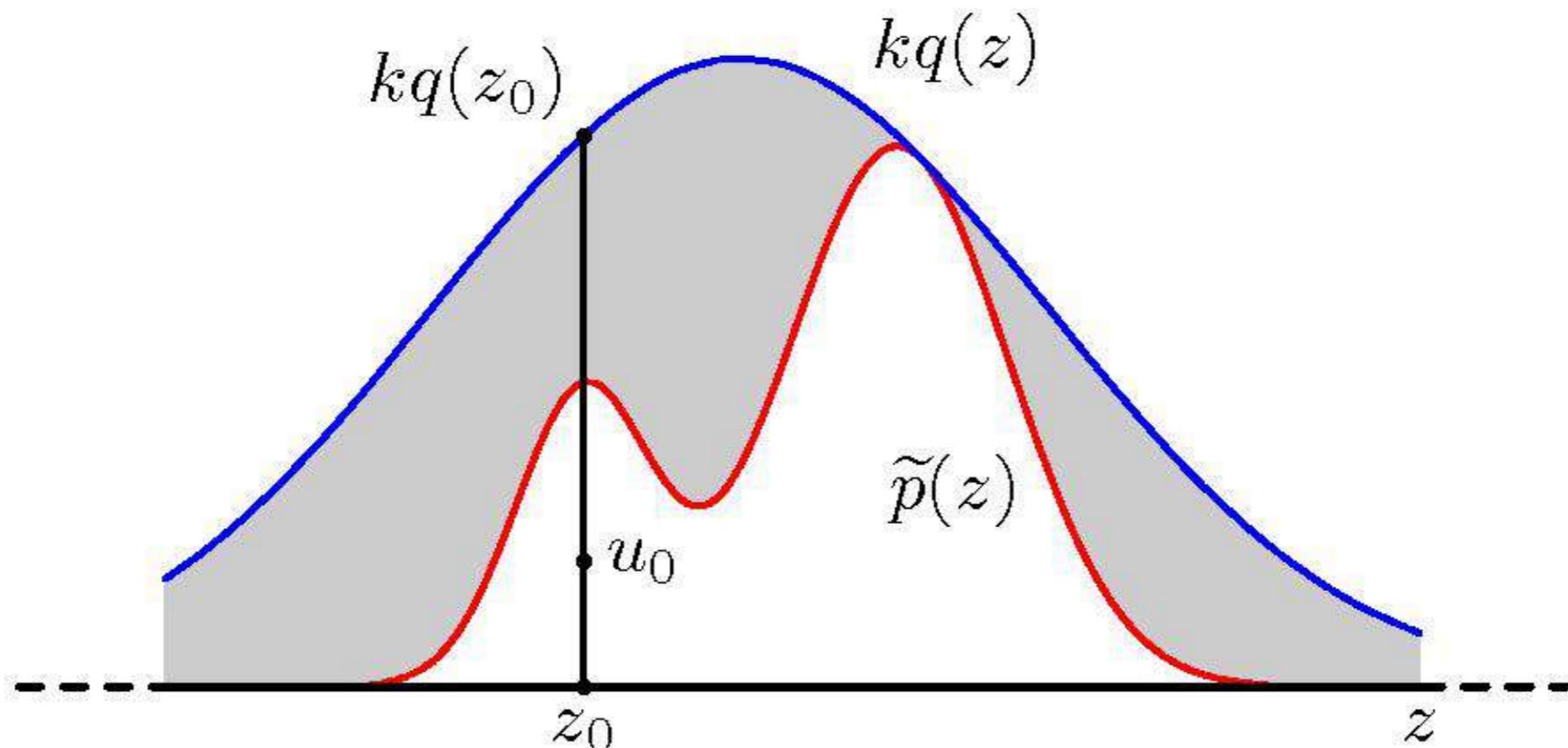
$$\underline{kq(x)} \geq \tilde{p}(x) \quad \forall x$$

**called as comparison function  
(something like an envelope)**

# Rejection Sampling

- Sample two random variables
  - $z_0 \sim q(x)$
  - $u_0 \sim [0, kq(z_0)]$  uniform
- Reject sample  $z_0$  if  $u_0 > \tilde{p}(z_0)$

$$p(x) = \frac{1}{Z} \tilde{p}(x) = \frac{1}{Z} \prod_c \phi_c(\mathcal{X}_c)$$
$$kq(x) \geq \tilde{p}(x) \quad \forall x$$



# Rejection Sampling

$$\begin{aligned} p(\text{accept}) &= \int \left\{ \frac{\tilde{p}(z)}{kq(z)} \right\} q(z) dz \\ &= \frac{1}{k} \int \tilde{p}(z) dz \end{aligned}$$

The fraction of points that are rejected by this method depends on the ratio of the area under the unnormalized distribution  $\tilde{p}(z)$  to the area under the curve  $kq(z)$  ( $k$  should be as small as possible)

- if  $q(x) = p(x)$  and  $k = 1$  then  $p(\text{accept}) = 1$
- but  $k > 1$  is typical
- for the easiest case of factoring distribution  $p(x) = \prod_{i=1}^D p(x_i)$  we have

$$p(\text{accept} \mid x) = \prod_{i=1}^D p(\text{accept} \mid x_i) = \mathcal{O}(\gamma^D)$$

where  $0 \leq \gamma < 1$  typical value for  $p(\text{accept} \mid x_i)$

- Thus rejection sampling is usually impractical in high dimensions

# Rejection Sampling

- Efficiency of rejection sampling: example

- ▶ Example:

- ▶ assume  $p(x)$  is Gaussian with covariance matrix:  $\sigma_p^2 I$

- ▶ assume  $q(x)$  is Gaussian with covariance matrix:  $\sigma_q^2 I$

- ▶ clearly:  $\sigma_q^2 \geq \sigma_p^2$

- ▶ in  $D$  dimensions:  $k = \left(\frac{\sigma_q}{\sigma_p}\right)^D$

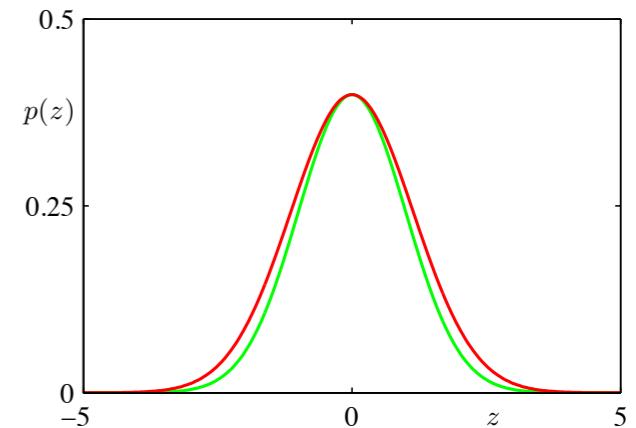
- ▶ assume:

- ▶  $\sigma_q$  is 1% larger than  $\sigma_p$ ,  $D = 1000$

- ▶ then  $k = 1.01^{1000} \geq 20000$

- ▶ and  $p(\text{accept}) \leq \frac{1}{20000}$

- ▶ therefore: often impractical to find good proposal distribution  $q(x)$  for high dimensions



# Rejection Sampling

- ▶ Multivariate: more than one dimension
- ▶ Idea: translate multivariate case into a univariate case:
- ▶ Enumerate all joint states  $(x_1, x_2, \dots, x_n)$  (assume discrete), i.e. give them each a unique  $i$  from 1 to the total (exponential) number of states
- ▶ Now we have to sample from univariate distributions again
- ▶ Problem: Exponential growth of states (with  $n$ )
- ▶ Another idea, use Bayes rule

$$p(x_1, x_2) = p(x_2 | x_1)p(x_1)$$

- ▶ Now first sample  $x_1$ , then  $x_2$  both of which are univariate
- ▶ Now we have a one dimensional distribution again
- ▶ Problem: Need to know the conditional distributions

# Sampling

---

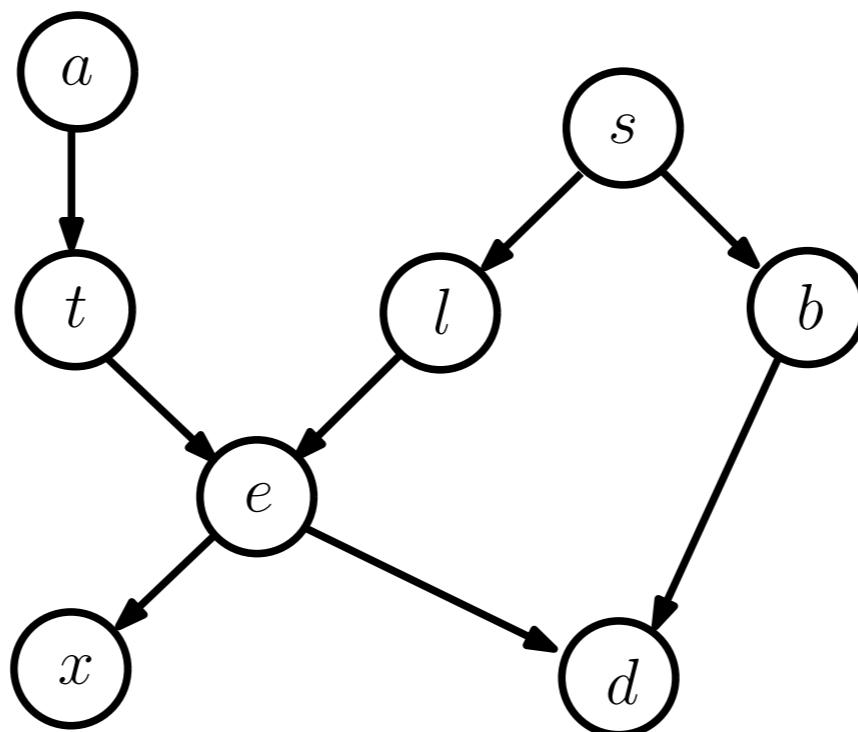
- Rejection Sampling
- **Ancestral Sampling**
- Importance Sampling
- Markov Chain Monte Carlo
- Gibbs Sampling

# Ancestral Sampling

- ▶ For Belief Networks (remember)  $p(x) = \prod_i p(x_i | \text{pa}(x_i))$
- ▶ So the sampling algorithm should be clear

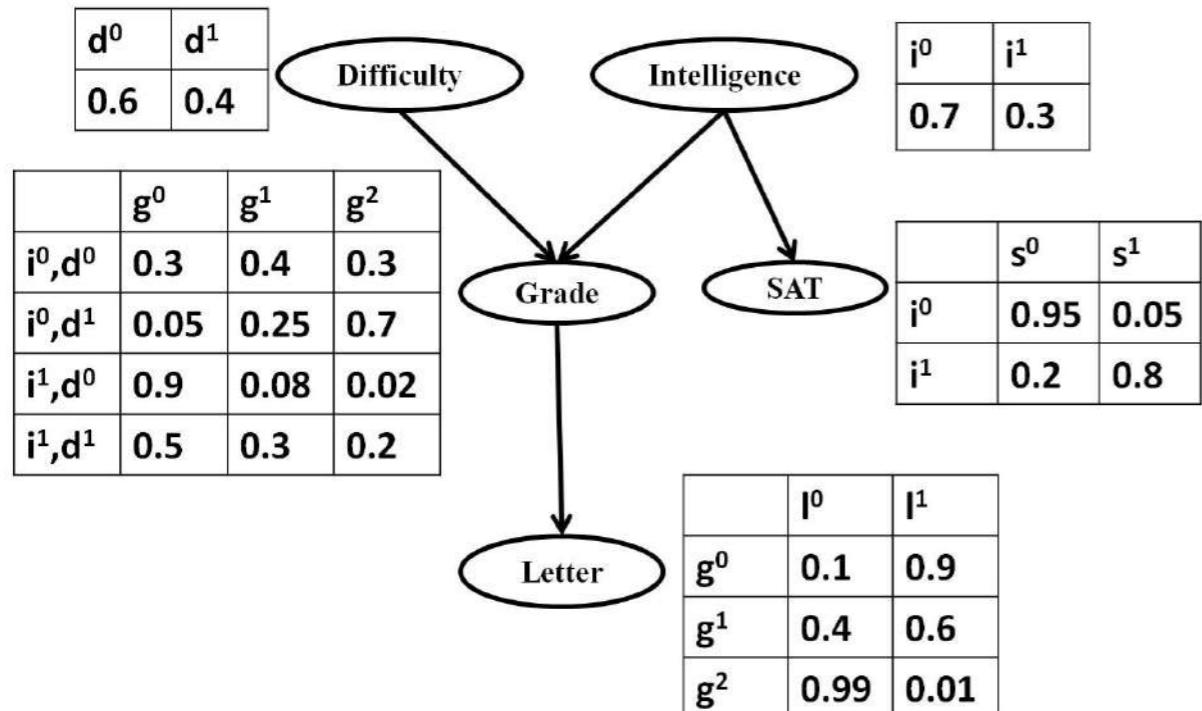
$$p(a, t, e, x, l, s, b, d) = p(a)p(s)p(t|a)p(l|s)p(b|s)p(e|t, l)p(x|e)p(d|e, b)$$

- ▶ **Forward sampling:** from parents to children
- ▶ sampling from each distribution ( $p(a), p(t | a), \dots$ )  
may be (in itself / as a subproblem) difficult



# Ancestral Sampling

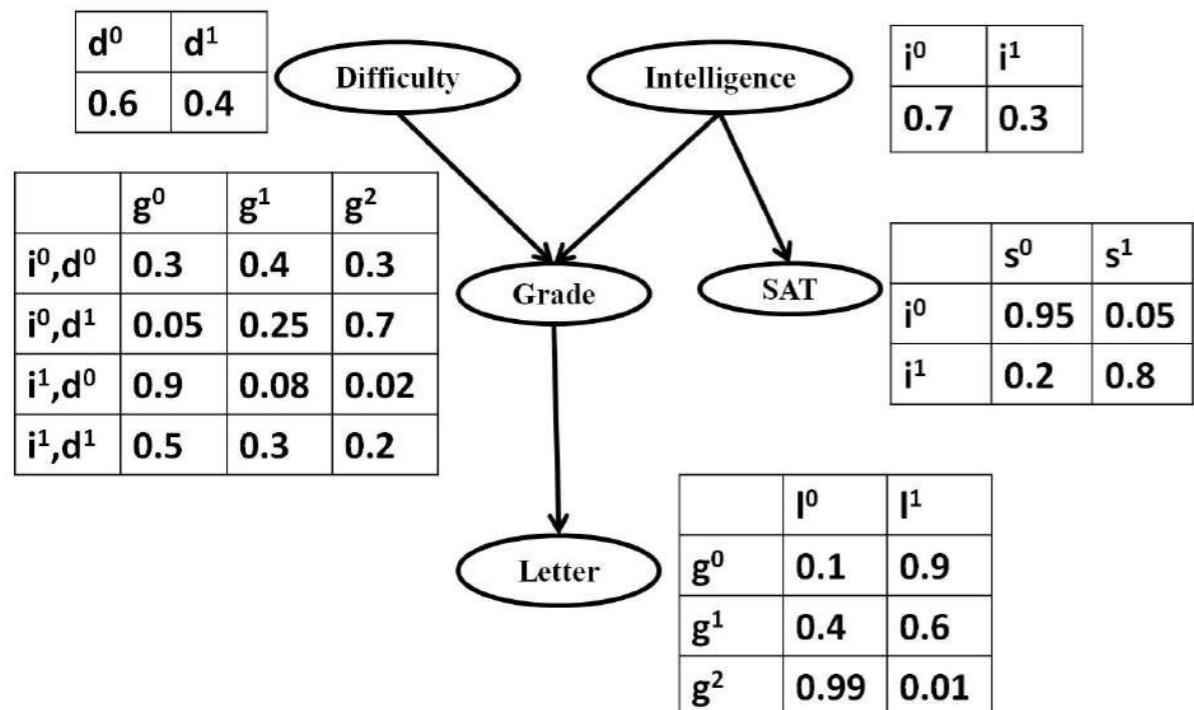
- ▶ Sample variables one by one in a topological order (parents of a node before the node)



- ▶ Sample **Difficulty** from  $P(D)$ .  
 $r = 0.723$ .  $D = ?$
- ▶ Sample **Intelligence** from  $P(I)$ .  
 $r = 0.34$ .  $I = ?$

# Ancestral Sampling

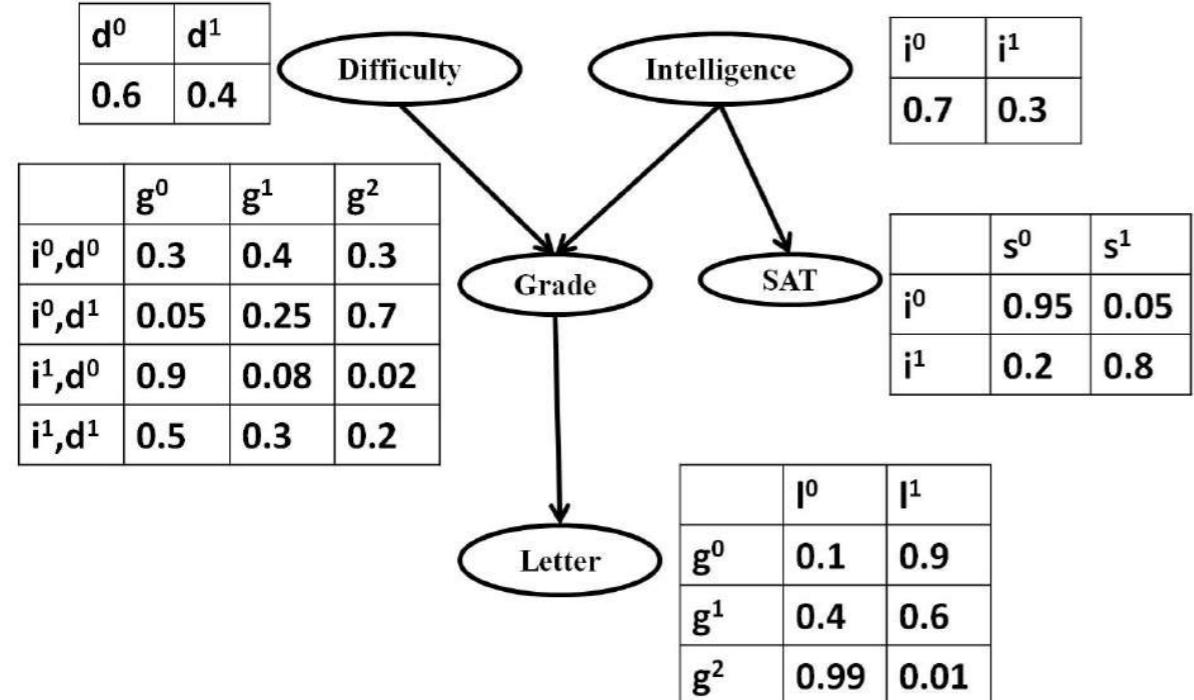
- ▶ Sample variables one by one in a topological order (parents of a node before the node)



- ▶ Sample **Difficulty** from  $P(D)$ .  
 $r = 0.723$ .  $D = d^1$
- ▶ Sample **Intelligence** from  $P(I)$ .  
 $r = 0.349$ .  $I = i^0$ .
- ▶ Sample **Grade** from  $P(G|i^0, d^1)$ .  
 $r = 0.281$ ,  $G = ?$ .
- ▶ Sample **SAT** from  $P(S|i^0)$ .  
 $r = 0.992$ ,  $S = ?$ .

# Ancestral Sampling

- ▶ Sample variables one by one in a topological order (parents of a node before the node)



Sample =  
 $(d^1, i^0, g^1, s^1, l^0)$

- ▶ Sample **Difficulty** from  $P(D)$ .  
 $r = 0.723, D = d^1$
- ▶ Sample **Intelligence** from  $P(I)$ .  
 $r = 0.349, I = i^0$ .
- ▶ Sample **Grade** from  $P(G|i^0, d^1)$ .  
 $r = 0.281, G = g^1$ .
- ▶ Sample **SAT** from  $P(S|i^0)$ .  
 $r = 0.992, S = s^1$ .
- ▶ Sample **Letter** from  $P(L|g^1)$ .  
 $r = 0.034, L = l^0$ .

# Ancestral Sampling

- ▶ Each instance drawn using forward sampling is independent!
- ▶ This is called **perfect sampling**
- ▶ In contrast to MCMC methods, where samples are dependent
- ▶ Problem: Evidence!
  - ▶ when a subset of the variables is observed
- ▶ Example, we have the following distribution
$$p(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3 \mid x_1, x_2)$$
- ▶ and have observed  $x_3$ .
- ▶ We want to sample from

$$p(x_1, x_2, \mid x_3) = \frac{p(x_1)p(x_2)p(x_3 \mid x_1, x_2)}{\sum_{x_1, x_2} p(x_1)p(x_2)p(x_3 \mid x_1, x_2)}$$

**probability of accepting a sample decreases rapidly as the number of observed variables increases**

- ▶ Observing  $x_3$  makes  $x_1, x_2$  dependent
- ▶ Sample and discard inconsistent ones (in-efficient)

# Sampling

---

- Rejection Sampling
- Ancestral Sampling
- **Importance Sampling**
- Markov Chain Monte Carlo
- Gibbs Sampling

# Importance Sampling

- Provide a framework to approximate **expectation** directly (but does not enable to draw samples from  $p(z)$  directly)

- Suppose it is impractical to sample directly from  $p(z)$  but we can evaluate  $p(z)$  easily for any given value of  $z$  (up to a normalization constant), our goal is:

$$\mathbb{E}[f] = \int f(z)p(z)dz$$

- Naive method: grid-sampling
  - discretize  $z$ -space into a uniform grid
  - evaluate the integrand as a sum of the form

$$\mathbb{E}[f] \simeq \sum_{l=1}^L f(z^l)p(z^l)$$

# Expectation

## Importance Sampling

The probability distributions of interest often have much of their mass confined to relatively small regions of  $z$  space.

- Uniform sampling will be very inefficient.
- We would really like to choose the sample points to fall in regions where  $p(z)$  is large, or ideally where the product  $p(z)f(z)$  is large.

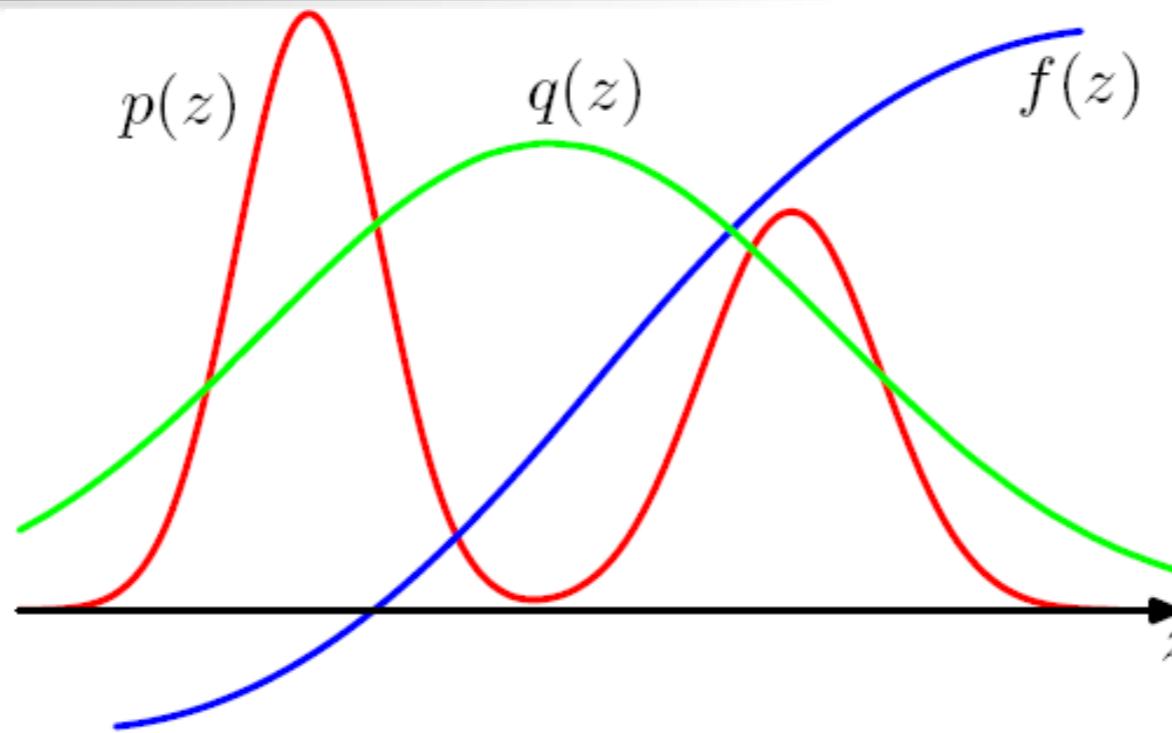
**Choose a proposal distribution  $q(z)$  from which it is easy to draw samples**

- Naive method: grid-sampling
  - discretize  $z$ -space into a uniform grid
  - evaluate the integrand as a sum of the form

$$\mathbb{E}[f] \simeq \sum_{l=1}^L f(z^l)p(z^l)$$

# Expectation

## Importance Sampling



**Choose a proposal distribution  $q(z)$  from which it is easy to draw samples**

$$\mathbb{E}[f] = \int f(z)p(z)dz = \int f(z)\frac{p(z)}{q(z)}q(z)dz$$

$$\approx \frac{1}{L} \sum_{l=1}^L \frac{p(z^l)}{q(z^l)} f(z^l)$$

$$r^l = \frac{p(z^l)}{q(z^l)} \text{ importance weights}$$

**Remark: unlike rejection sampling, all the samples are retained**

# Expectation

## Importance Sampling

Typical setting:

- ▶  $p(z)$  can be only evaluated up to a normalization constant (unkown):

$$p(z) = \tilde{p}(z)/Z_p$$

- ▶  $q(z)$  can be also treated in a similar way:

$$q(z) = \tilde{q}(z)/Z_q$$

- ▶ then:

$$\begin{aligned}\mathbb{E}[f] &= \int f(z)p(z)dz = \frac{Z_q}{Z_p} \int f(z) \frac{\tilde{p}(z)}{\tilde{q}(z)} q(z) dz \\ &\simeq \boxed{\frac{Z_q}{Z_p}} \frac{1}{L} \sum_{l=1}^L \tilde{r}^l f(z^l)\end{aligned}$$

**$q$  over  $p$**

- ▶ with:  $\tilde{r}^l = \frac{\tilde{p}(z^l)}{\tilde{q}(z^l)}$

# Expectation

# Importance Sampling

Ratio of normalization constants can be evaluated :

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q} \int \tilde{p}(z) dz = \int \frac{\tilde{p}(z)}{\tilde{q}(z)} q(z) dz \simeq \frac{1}{L} \sum_{l=1}^L \tilde{r}^l$$

*p over q*

still use samples  
to approximate

- and therefore:

$$\mathbb{E}[f] \simeq \frac{Z_q}{Z_p} \frac{1}{L} \sum_{l=1}^L \tilde{r}^l f(z^l) = \sum_{l=1}^L w^l f(z^l)$$

- ## ► with:

$$w^l = \frac{\tilde{r}^l}{\sum_m \tilde{r}^m} = \frac{\frac{\tilde{p}(z^l)}{\tilde{q}(z^l)}}{\sum_m \frac{\tilde{p}(z^m)}{\tilde{q}(z^m)}}$$

## Importance Sampling

Observations:

- ▶ success of importance sampling depends crucially on how well the sampling distribution  $q(z)$  matches the desired distribution  $p(z)$
- ▶ often,  $p(z)f(z)$  is strongly varying and has significant proportion of its mass concentrated over small regions of  $z$ -space
- ▶ as a result weights  $\tilde{r}^l$  may be dominated by a few weights having large values
- ▶ practical issues: if none of the samples falls in the regions where  $p(z)f(z)$  are large ...
  - ▶ the results may be arbitrarily wrong
  - ▶ and no diagnostic indication !  
(because there is no large variance in  $\tilde{r}^l$  then)

**design proposal distribution  $q(z)$  usually is not science but ART!**

# Sampling

---

- Rejection Sampling
- Ancestral Sampling
- Importance Sampling
- **Markov Chain Monte Carlo**
- Gibbs Sampling

# Markov Chain Monte Carlo

---

- In rejection sampling and importance sampling, we saw they suffer from severe limitations particularly in spaces of high-D
- We turn to introduce a very general & powerful framework: **MCMC**
  - allow sampling from a large class of distributions
  - scale well with the dimensionality of the sample space
  - again, we sample from a **proposal distribution** (but special of course:D)

# Markov Chain Monte Carlo

we assume  $p(z)$  can be evaluated up to a constant (unknown):  $p(z) = \tilde{p}(z)/Z_p$

**again, instead of sampling from  $p(z)$ , we sample from a proposal distribution**

we maintain a record of the current state  $z^{(\tau)}$   
and the proposal distribution  $q(z | z^{(\tau)})$  depends on this current state  
and so the sequence of samples  $z^{(1)}, z^{(2)}, \dots$  forms a **Markov chain**  
**(samples are highly correlated)**

The proposal distribution is chosen to be sufficiently simple  
that it is straightforward to draw sample from it directly

At each iteration, we generate a candidate sample  $z^*$   
from proposal distribution and then accept the sample  
according to an appropriate criterion.

# Markov Chain Monte Carlo

## Metropolis sampling

special case of MCMC with following proposal distribution:

- symmetric:  $q(z_A|z_B) = q(z_B|z_A)$  for all values of  $z_A$  and  $z_B$

sample candidate  $z^*$  and accept with probability

$$A(z^*, z^{(\tau)}) = \min\left(1, \frac{\tilde{p}(z^*)}{\tilde{p}(z^{(\tau)})}\right)$$

▷ if new sample  $z^*$  is more probable, always accept it

▷ if new sample is less probable, accept with  $\frac{\tilde{p}(z^*)}{\tilde{p}(z^{(\tau)})}$

**explores state space according to the probability defined by  $p(z)$ ,  
hence generate a sequence that eventually represents draws from  $p(z)$**

**(as long as  $q(z_A|z_B) > 0$  for any  $z_A$  and  $z_B$ , the distribution of  $z^{(\tau)}$  tends to  $p(z)$  as  $\tau \rightarrow \infty$ )**

# Markov Chain Monte Carlo

## Metropolis-Hastings sampling

slightly more general MCMC method when  
the proposal distribution is not symmetric

sample candidate  $z^*$  and accept with probability

$$A(z^*, z^{(\tau)}) = \min\left(1, \frac{\tilde{p}(z^*)q(z^{(\tau)} \mid z^*)}{\tilde{p}(z^{(\tau)})q(z^* \mid z^{(\tau)})}\right)$$

**Note: when the proposal distribution is symmetric,  
Metropolis-Hastings reduces to standard Metropolis sampling**

# Markov Chain Monte Carlo

- Remark:
  - For continuous state spaces, a common choice is a Gaussian centered on the current state.
$$\tilde{q}(z' | z) = \mathcal{N}(z' | z, \sigma^2 I)$$
which is symmetric  $\tilde{q}(z' | z) = \tilde{q}(z | z')$
  - If the variance is small, the proportion of accepted transitions will be high, but progress through the state space takes the form of a slow random walk leading to long correlation times.
  - If the variance parameter is large, the rejection rate is high because many of the proposed steps will be to states for which the probability  $p(z)$  is low

# Markov Chain Monte Carlo - Proof of Convergence

- Our goal is to use Markov chains to sample from a given distribution
- Let's take a look into the properties of Markov chain first

$$p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}) = p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)}) \quad \text{first-order Markov chain}$$

$$T_m(\mathbf{z}^{(m)}, \mathbf{z}^{(m+1)}) \equiv p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)})$$

**transition probability**

(A Markov chain is called *homogenous* if the  $T_m$  is the same for all  $m$ )

$$p(\mathbf{z}^{(m+1)}) = \sum_{\mathbf{z}^{(m)}} p(\mathbf{z}^{(m+1)} | \mathbf{z}^{(m)}) p(\mathbf{z}^{(m)}) \quad \text{marginal probability}$$

A distribution is said to be *invariant*, or *stationary*, w.r.t. a Markov chain if each step in the chain leaves that distribution *invariant*.

→ For a homogeneous Markov chain with transition probabilities  $T(z', z)$ , the distribution  $p^*(z)$  is invariant if  $p^*(z) = \sum_{z'} T(z', z) p^*(z')$

# Markov Chain Monte Carlo - Proof of Convergence

- Our goal is to use Markov chains to sample from a given distribution

**A sufficient condition for ensuring that “*the distribution  $p(z)$  is invariant*” is to choose the transition probabilities to satisfy “*detailed balance*” defined by:**

$$p^*(\mathbf{z})T(\mathbf{z}, \mathbf{z}') = p^*(\mathbf{z}')T(\mathbf{z}', \mathbf{z})$$

for the particular distribution  $p^*(z)$ .

**Proof:**  $\sum_{\mathbf{z}'} \underline{\underline{p^*(\mathbf{z}') T(\mathbf{z}', \mathbf{z})}} = \sum_{\mathbf{z}'} \underline{\underline{p^*(\mathbf{z}) T(\mathbf{z}, \mathbf{z}')}} = p^*(\mathbf{z}) \sum_{\mathbf{z}'} \underline{\underline{p(\mathbf{z}'|\mathbf{z})}} = p^*(\mathbf{z})$

**(Remark: A Markov chain that satisfies detailed balance is said to be *reversible*)**

**A distribution is said to be *invariant*, or *stationary*, w.r.t. a Markov chain if each step in the chain leaves that distribution *invariant*.**

→ For a homogeneous Markov chain with transition probabilities  $T(z', z)$ ,  
**the distribution  $p^*(z)$  is invariant if**  $p^*(z) = \sum_{z'} T(z', z)p^*(z')$

# Markov Chain Monte Carlo - Proof of Convergence

- Our goal is to use Markov chains to sample from a given distribution
  - We can achieve this if we set up a Markov chain such that the desired distribution is invariant
  - We must also require that for  $m \rightarrow \infty$ , the distribution  $p(z^{(m)})$  converges to the required invariant distribution  $p^*(z)$ , irrespective of the choice of initial distribution  $p(z^{(0)})$ . (The *ergodicity* property)
  - It can be shown that a homogeneous Markov chain will be ergodic, subject only to weak restrictions on the invariant distribution and the transition probabilities
  -

# Markov Chain Monte Carlo - Proof of Convergence

- Our goal is to use Markov chains to sample from a given distribution
  - We can achieve this if we set up a Markov chain such that the desired distribution is invariant
  - We must also require that for  $m \rightarrow \infty$ , the distribution  $p(z^{(m)})$  converges to the required invariant distribution  $p^*(z)$ , irrespective of the choice of initial distribution  $p(z^{(0)})$ . (The *ergodicity* property)
  - We can show that  $p(z)$  is an invariant distribution of the Markov chain defined by Metropolis-Hastings algorithm by showing detailed balance

$$\begin{aligned} p(z)q(z|z')A(z', z) &= \min(p(z)q(z|z'), p(z')q(z'|z)) \\ &= \min(p(z')q(z'|z), p(z)q(z|z')) \\ &= p(z')q(z'|z)A(z, z') \end{aligned}$$

# Sampling

---

- Rejection Sampling
- Ancestral Sampling
- Importance Sampling
- Markov Chain Monte Carlo
- **Gibbs Sampling**

# Gibbs Sampling

- A simple and widely applicable MCMC algorithm
  - Consider  $p(\mathbf{z}) = p(z_1, \dots, z_M)$  from which we wish to sample, and suppose that we have chosen some initial state for Markov chain
    1. Initialize  $\{z_i : i = 1, \dots, M\}$
    2. For  $\tau = 1, \dots, T$ :
      - Sample  $z_1^{(\tau+1)} \sim p(z_1 | z_2^{(\tau)}, z_3^{(\tau)}, \dots, z_M^{(\tau)})$ .
      - Sample  $z_2^{(\tau+1)} \sim p(z_2 | z_1^{(\tau+1)}, z_3^{(\tau)}, \dots, z_M^{(\tau)})$ .
      - ⋮
      - Sample  $z_j^{(\tau+1)} \sim p(z_j | z_1^{(\tau+1)}, \dots, z_{j-1}^{(\tau+1)}, z_{j+1}^{(\tau)}, \dots, z_M^{(\tau)})$ .
      - ⋮
      - Sample  $z_M^{(\tau+1)} \sim p(z_M | z_1^{(\tau+1)}, z_2^{(\tau+1)}, \dots, z_{M-1}^{(\tau+1)})$ .

# Gibbs Sampling

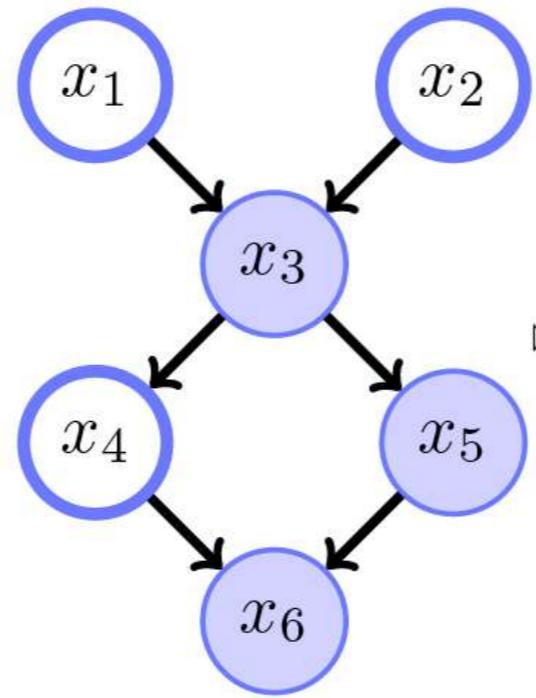
- A simple and widely applicable MCMC algorithm
  - Consider  $p(\mathbf{z}) = p(z_1, \dots, z_M)$  from which we wish to sample, and suppose that we have chosen some initial state for Markov chain
  - The Gibbs sampling procedure is a particular instance of Metropolis-Hastings algorithm

$$q(\mathbf{z}^* \mid \mathbf{z}) = p(z_k^* \mid \mathbf{z}_{\setminus k}) \quad \text{sample only } k\text{-th variable, and keep other variables fixed}$$

$$\mathbf{z}_{\setminus k}^* = \mathbf{z}_{\setminus k} \quad \text{these variables are unchanged (only } k\text{-th variable might change)}$$

$$A(\mathbf{z}^*, \mathbf{z}) = \frac{p(\mathbf{z}^*)q(\mathbf{z} \mid \mathbf{z}^*)}{p(\mathbf{z})q(\mathbf{z}^* \mid \mathbf{z})} = \frac{p(z_k^* \mid \mathbf{z}_{\setminus k}^*)p(\mathbf{z}_{\setminus k}^*)p(z_k \mid \mathbf{z}_{\setminus k}^*)}{p(z_k \mid \mathbf{z}_{\setminus k})p(\mathbf{z}_{\setminus k})p(z_k^* \mid \mathbf{z}_{\setminus k})} = 1 \quad \text{always accept!}$$

# Gibbs Sampling



- ▶ Sample from this distribution  $p(x)$
- ▶ Idea: Sample sequence  $x^0, x^1, x^2, \dots$  by updating one variable at a time
- ▶ Eg. update  $x_4$  by conditioning on the set of shaded variables **Markov blanket**

$$p(x_4 \mid x_1, x_2, x_3, x_5, x_6) = p(x_4 \mid x_3, x_5, x_6)$$

# Gibbs Sampling

- ▶ Update  $x_i$

$$p(x_i \mid x_{\setminus i}) = \frac{1}{Z} p(x_i \mid \text{pa}(x_i)) \prod_{j \in \text{ch}(i)} p(x_j \mid \text{pa}(x_j))$$

- ▶ and the normalisation constant is

$$Z = \sum_{x_i} p(x_i \mid \text{pa}(x_i)) \prod_{j \in \text{ch}(i)} p(x_j \mid \text{pa}(x_j))$$

- ▶ Think of Gibbs sampling as

$$x^{l+1} \sim q(\cdot \mid x^l)$$

- ▶ Problem: States are highly dependent ( $x^1, x^2, \dots$ )
- ▶ Need a long time to run Gibbs sampling to *forget* the initial state, this is called **burn in** phase
- ▶ Dealing with evidence is easy: simply clamp the variables to the values.

# Gibbs Sampling

---

- ▶ Much much more to learn about sampling
- ▶ Widely used: Gibbs Sampling, Metropolis Hastings
- ▶ Usually requires experience and careful adaption to your specific problem