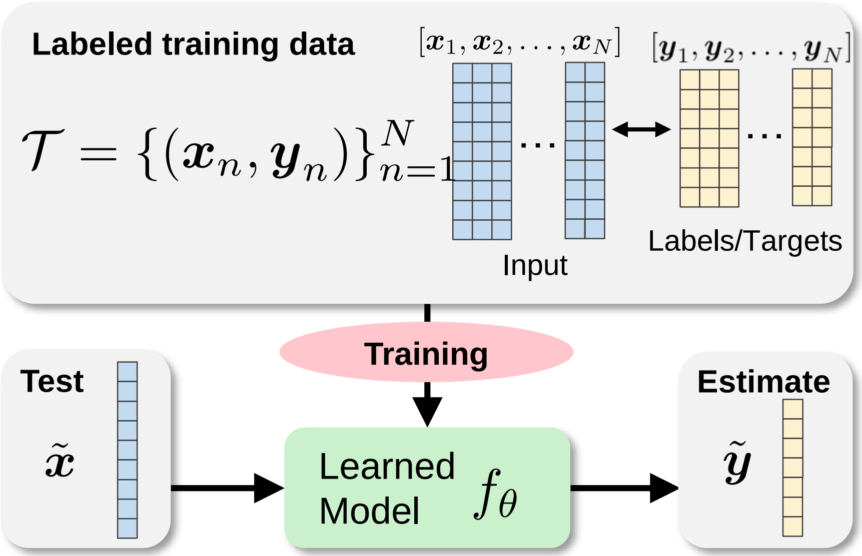
OUTLINE

I. ~~Introduction~~
II. ~~Background~~
III. ~~Fitting a Model~~
IV. ~~Supervised Learning~~
V. Unsupervised Learning

VI. Convolutional Neural Networks

OUTLINE

Supervised Learning



1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

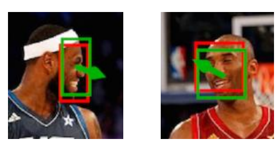
$\boldsymbol{y} =$ 1

13: German Shepherd

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$

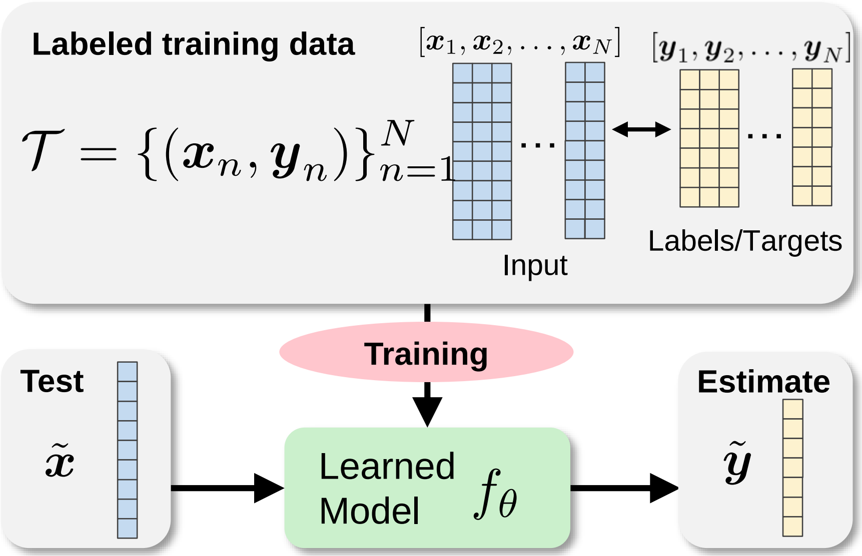


3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1, 0.6, 0.3

Supervised Learning



1. Discrete case: («one-hot»)
- Classification

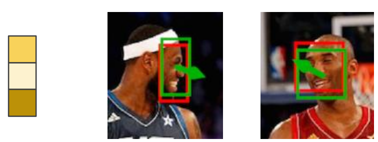
Ex. application: *dog breed*

$\boldsymbol{y} =$

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$

Unsupervised Learning

Supervised Learning

Labeled training data

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$$

$[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ Input ⟷ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$ Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)

- Classification

Ex. application: *dog breed*

$\boldsymbol{y} =$ 1

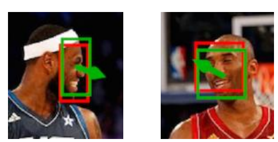
13: German Shepherd

2. Continuous case:

- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:

- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1, 0.6, 0.3

Unsupervised Learning

Unlabeled training data

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$$

$[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ Input

Supervised Learning

Labeled training data

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^N$$

$[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ Input ↔ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$ Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

$\boldsymbol{y} =$ 1

13: German Shepherd

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$

3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1, 0.6, 0.3

Unsupervised Learning

Unlabeled training data

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^N$$

$[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ Input

Training

Learned Model f_θ

Supervised Learning

Labeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$$

Input ⟷ Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

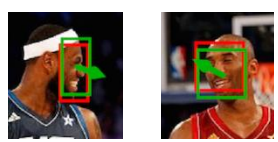
$\boldsymbol{y} =$ 1

13: German Shepherd

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1, 0.6, 0.3

Unsupervised Learning

Unlabeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$$

Input

Training

Learned Model f_θ → **Representation** $[\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_N]$

Supervised Learning

Labeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N]$

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$$

Input

Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

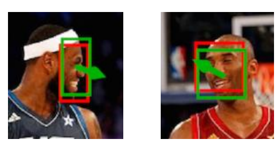
$\boldsymbol{y} =$ 1

13: German Shepherd

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1 0.6 0.3

Unsupervised Learning

Unlabeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$$

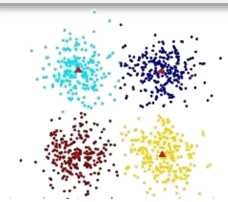
Input

Training

Learned Model f_θ → **Representation** $[\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N]$

1. Discrete case:
- Clustering

$\boldsymbol{z} =$



Supervised Learning

Labeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$$

Input

Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- ▸ Classification

Ex. application: *dog breed*

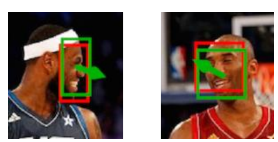
$\boldsymbol{y} =$ 1

13: German Shepherd

2. Continuous case:
- ▸ Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- ▸ Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1 0.6 0.3

Unsupervised Learning

Unlabeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$$

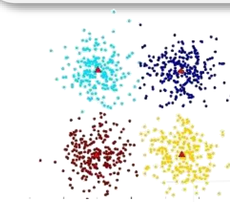
Input

Training

Learned Model f_θ → **Representation** $[\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_N]$

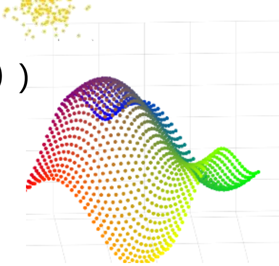
1. Discrete case:
- ▸ Clustering

$\boldsymbol{z} =$



2. Continuous case: ($\dim(\boldsymbol{z}) \ll \dim(\boldsymbol{x})$)
- ▸ Dimensionality Reduction

$\boldsymbol{z} =$



Supervised Learning

Labeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$$

Input — Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

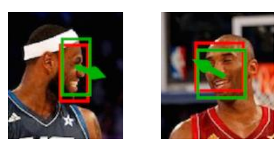
$\boldsymbol{y} =$ 1

13: German Shepherd

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1 0.6 0.3

Unsupervised Learning

Unlabeled training data $[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$$

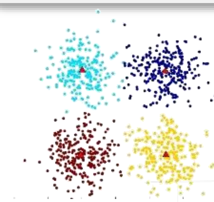
Input

Training

Learned Model f_θ → **Representation** $[\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_N]$

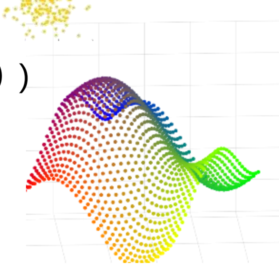
1. Discrete case:
- Clustering

$\boldsymbol{z} =$



2. Continuous case: ($\dim(\boldsymbol{z}) \ll \dim(\boldsymbol{x})$)
- Dimensionality Reduction

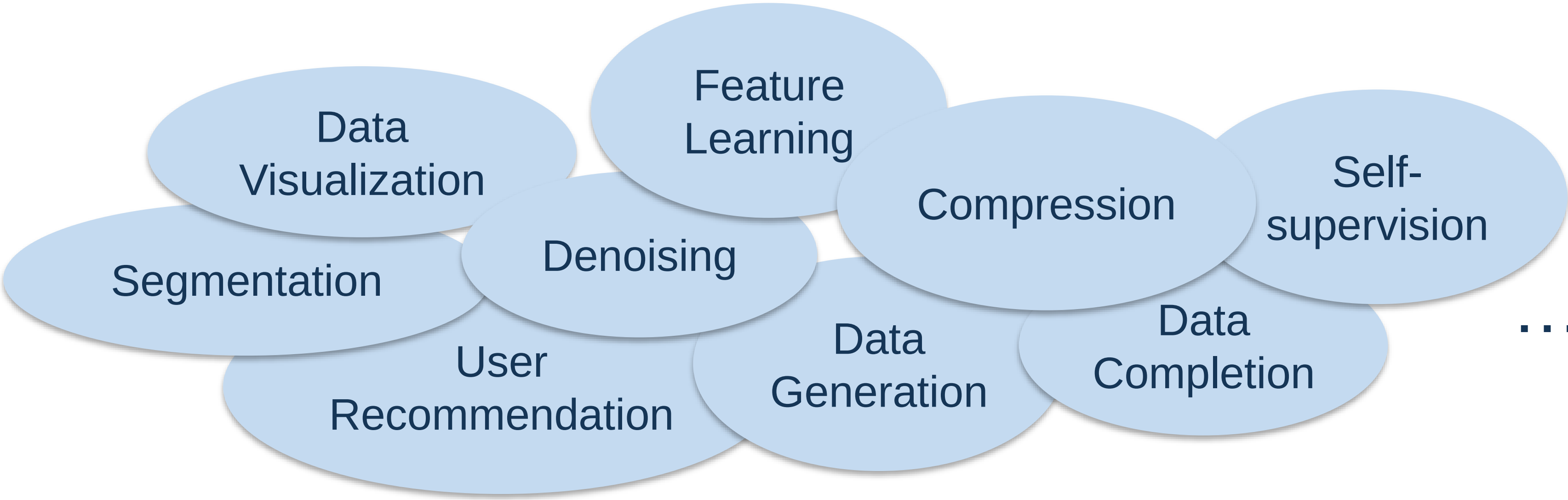
$\boldsymbol{z} =$



3. Sparse case:
- Dictionary Learning

$\boldsymbol{z} =$

Applications of Unsupervised Learning



Applications of Unsupervised Learning

Feature Learning

Data Visualization

Compression

Self-supervision

Denoising

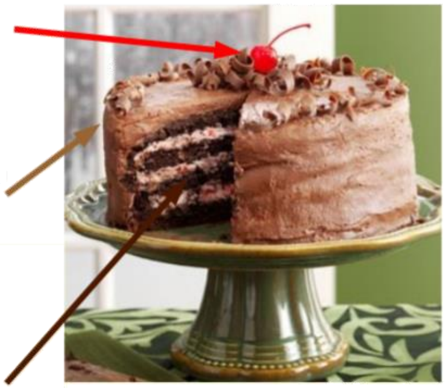
Segmentation

User Recommendation

Data Generation

Data Completion

...



« If machine learning is a cake, then unsupervised learning is the actual cake, supervised learning is the icing, and reinforcement learning is the cherry on the top. »

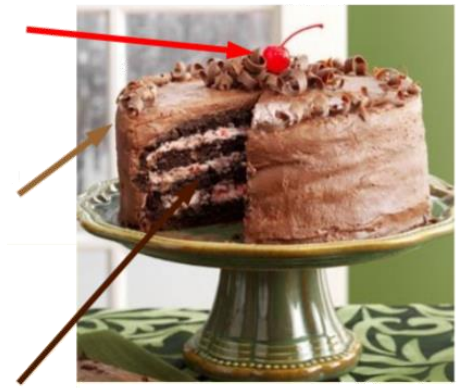
-Yann Lecun (Facebook AI) at NIPS 2016

Applications of Unsupervised Learning

- Data Visualization
- Feature Learning
- Compression
- Self-supervision
- Segmentation
- Denoising
- User Recommendation
- Data Generation
- Data Completion
- ...

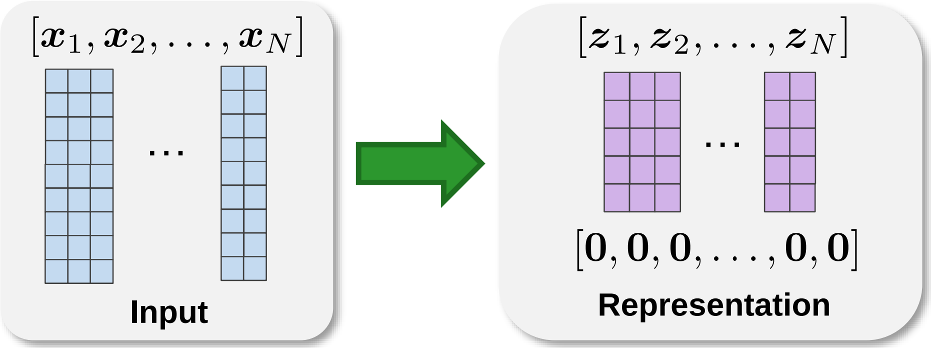
« If machine learning is a cake, then unsupervised learning is the actual cake, supervised learning is the icing, and reinforcement learning is the cherry on the top. »

-Yann Lecun (Facebook AI) at NIPS 2016

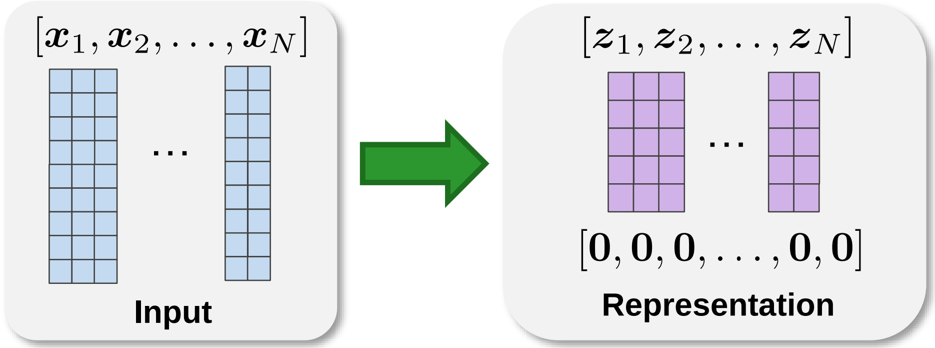


⇨ Potential to learn from massive amount of unlabeled data to generate even more

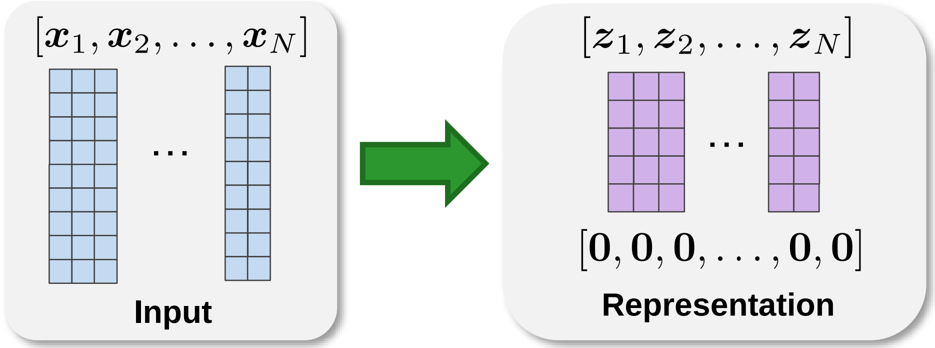
How does it work?



How does it work?

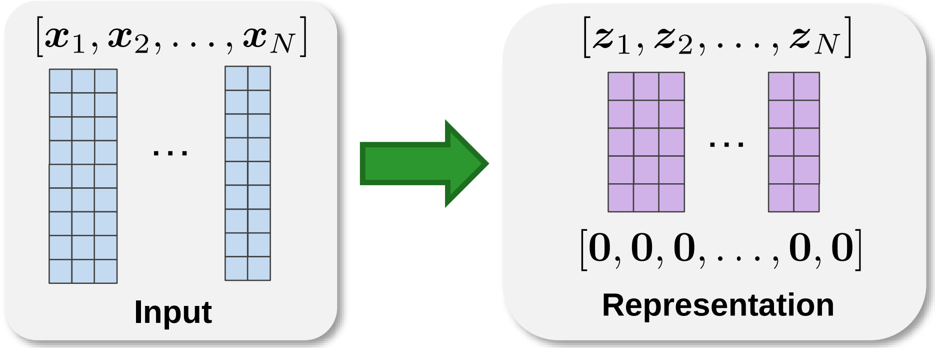


How does it work?



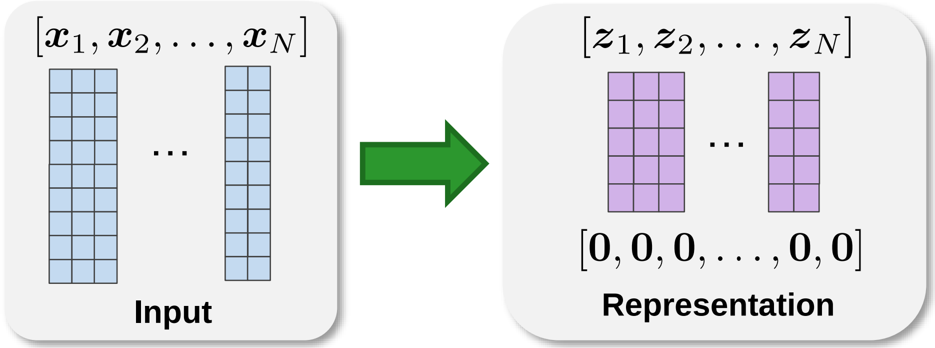
- **Desirable properties**
  - Preserve information from original data
  - Can be used to reconstruct it / generalize it / efficiently learn from it

How does it work?



- **Desirable properties**
  - Preserve information from original data
  - Can be used to reconstruct it / generalize it / efficiently learn from it
- **A unifying view :** Generative models

How does it work?

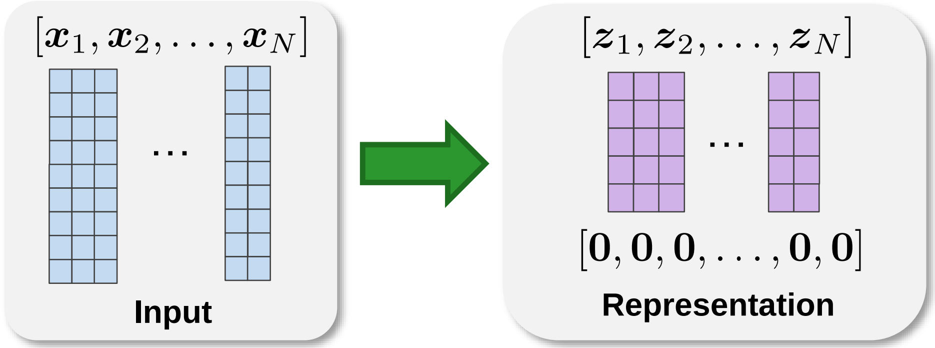


- **Desirable properties**
  - Preserve information from original data
  - Can be used to reconstruct it / generalize it / efficiently learn from it
- **A unifying view :** Generative models

Find $\begin{cases} p_\theta(\boldsymbol{z}) & \text{(as simple as possible)} \\ p_\theta(\boldsymbol{x}|\boldsymbol{z}) & \text{(typically, transformation of } \boldsymbol{z}\text{)} \end{cases}$ so that $p_\theta(\boldsymbol{x}) = \int_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z}) d\boldsymbol{z}$

fits the data (e.g., maximum likelihood).

How does it work?



- **Desirable properties**
  - Preserve information from original data
  - Can be used to reconstruct it / generalize it / efficiently learn from it
- **A unifying view :** Generative models

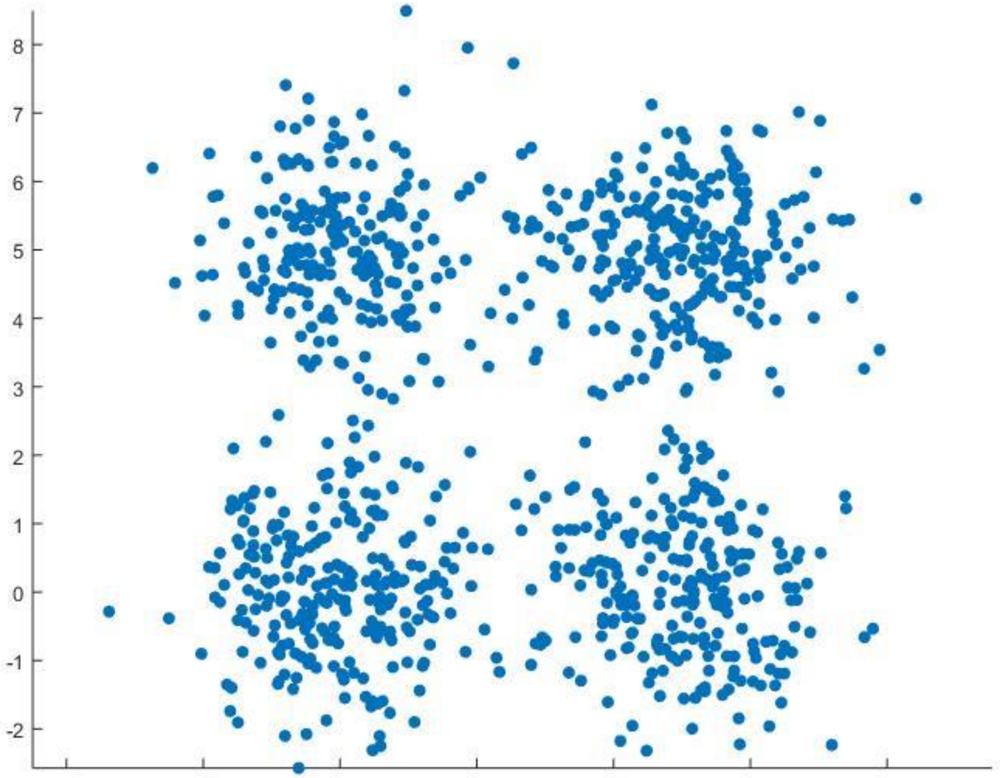
Find $\begin{cases} p_\theta(\boldsymbol{z}) & \text{(as simple as possible)} \\ p_\theta(\boldsymbol{x}|\boldsymbol{z}) & \text{(typically, transformation of } \boldsymbol{z}\text{)} \end{cases}$ so that $p_\theta(\boldsymbol{x}) = \int_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z}) d\boldsymbol{z}$

fits the data (e.g., maximum likelihood).

Generation	$p_\theta(\boldsymbol{z}) \rightarrow \boldsymbol{z} \rightarrow \boldsymbol{x}$
Compression	$\boldsymbol{x} \rightarrow p_\theta(\boldsymbol{z}|\boldsymbol{x}) \rightarrow \boldsymbol{z}$
Reconstruction / Completion / Denoising	$\tilde{\boldsymbol{x}} \rightarrow p_\theta(\boldsymbol{z}|\tilde{\boldsymbol{x}}) \rightarrow \boldsymbol{z} \rightarrow p_\theta(\boldsymbol{x}|\boldsymbol{z}) \rightarrow \hat{\boldsymbol{x}}$

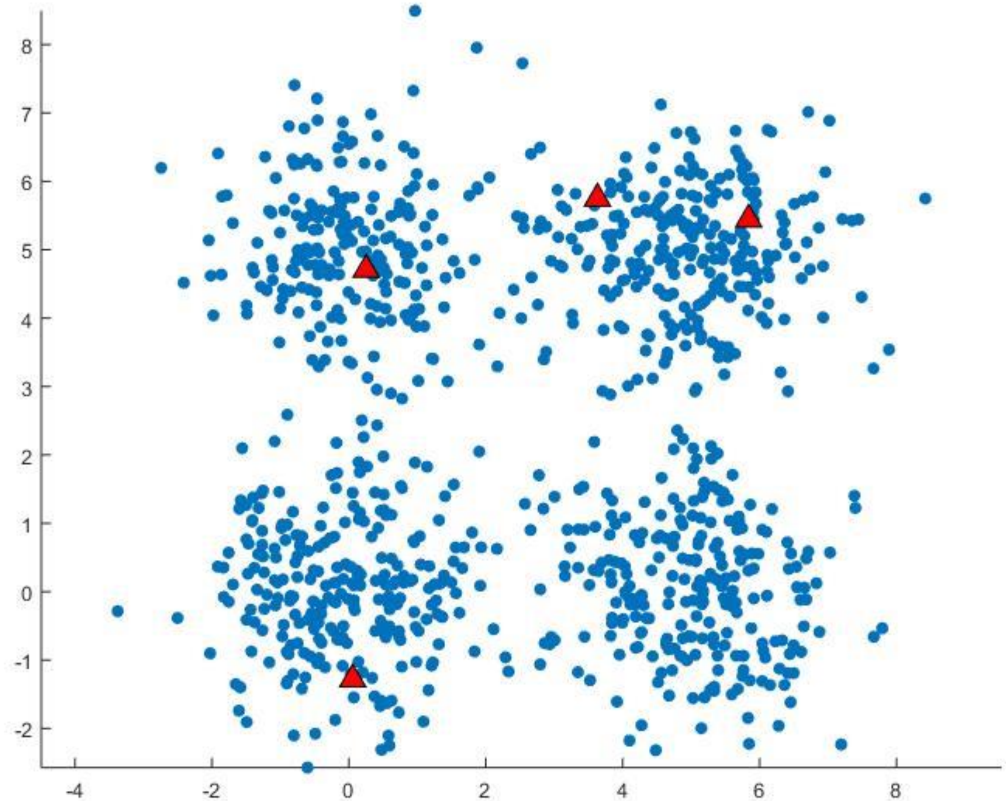
How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^{N} \subset \mathbb{R}^2$



How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$

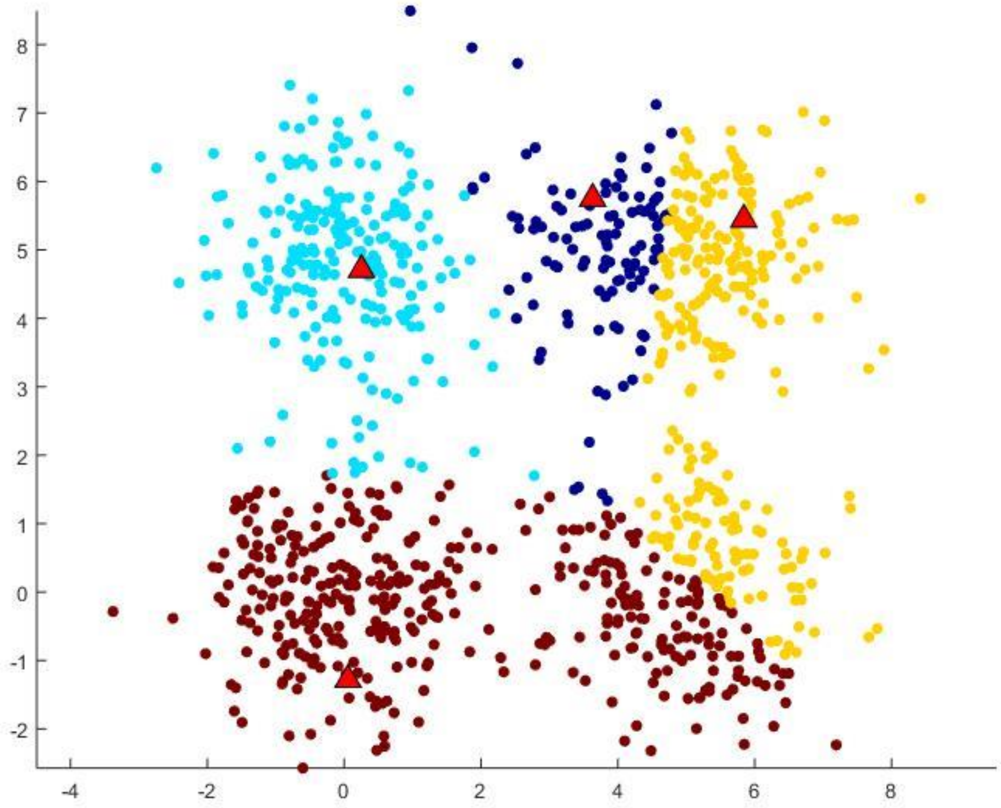


1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



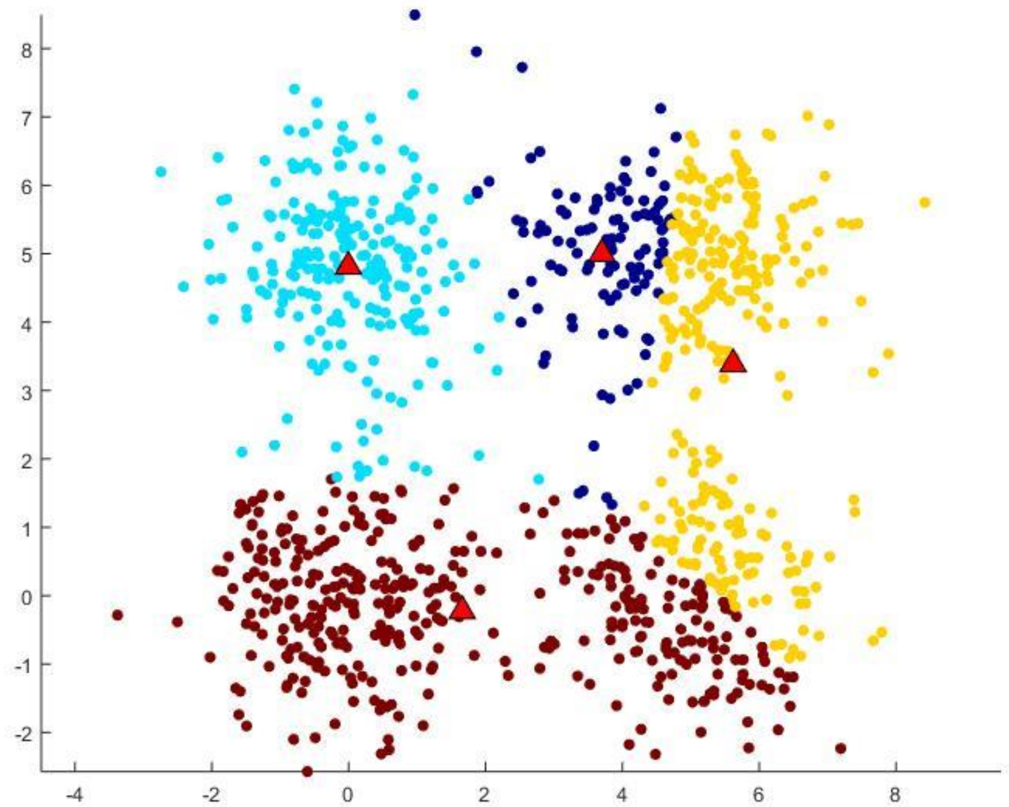
1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^{N} \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^{K} \subset \mathbb{R}^2$

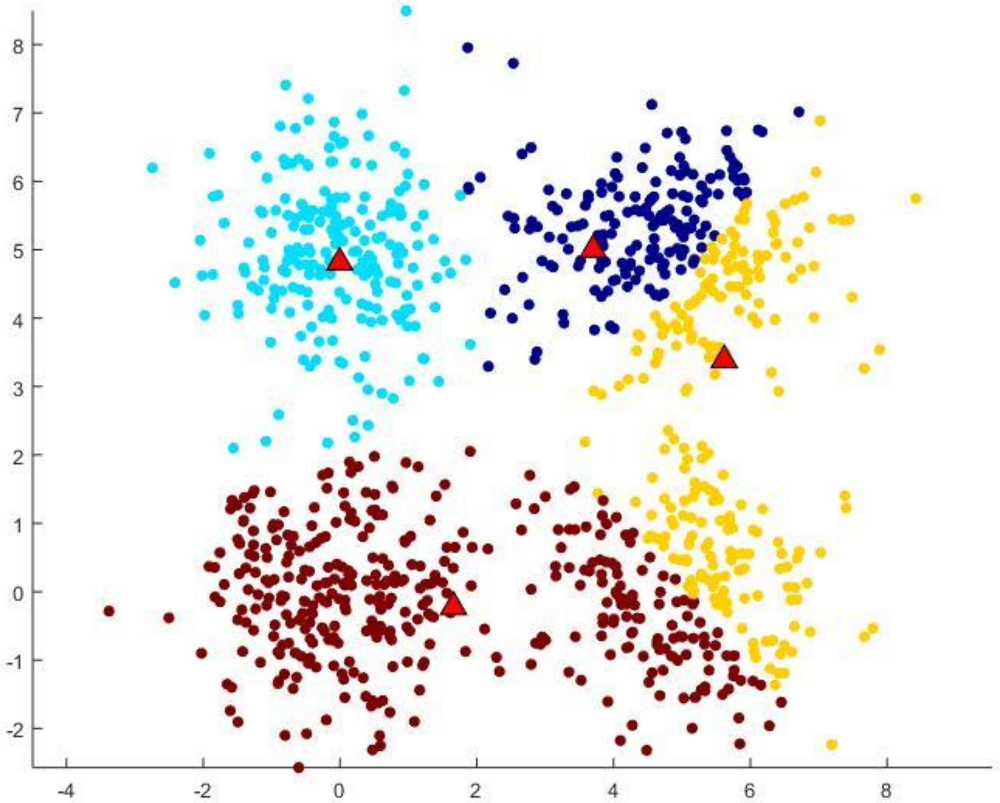
2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

---

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

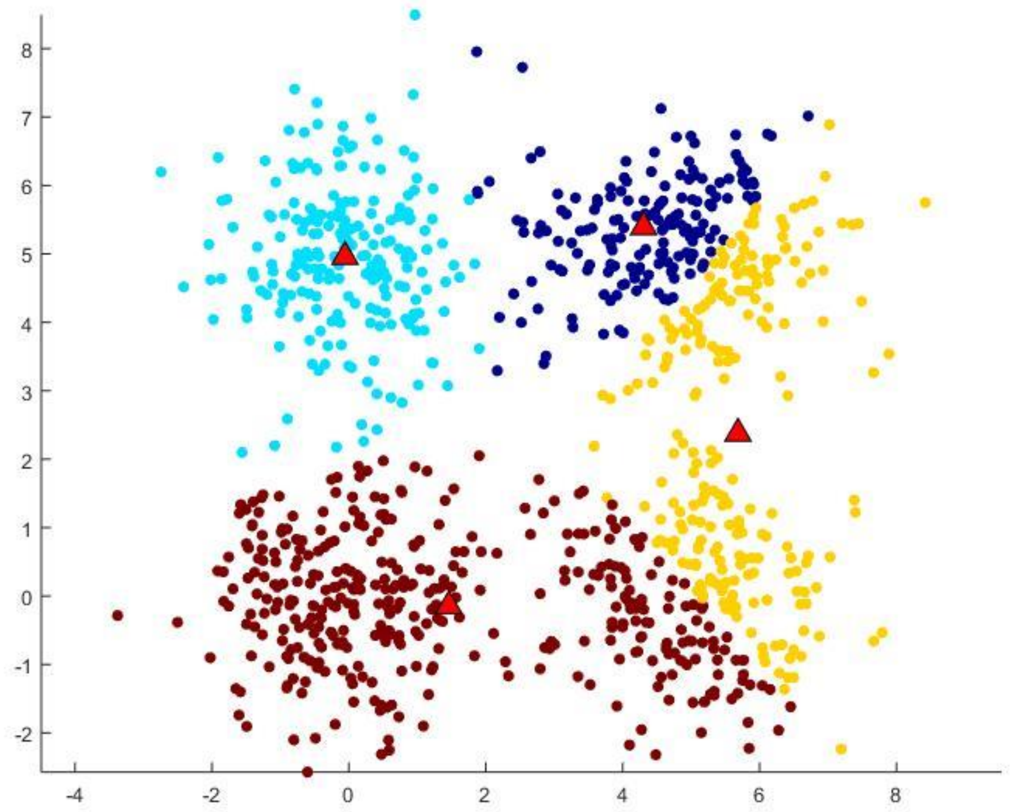
- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

---

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

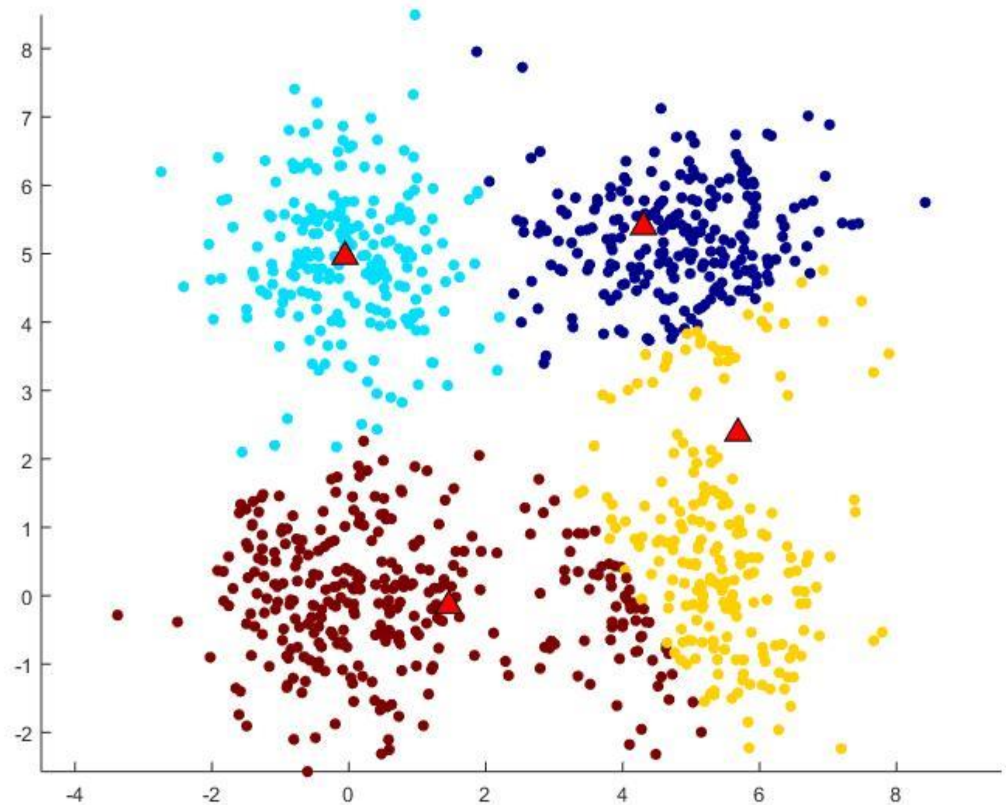
3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

---

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

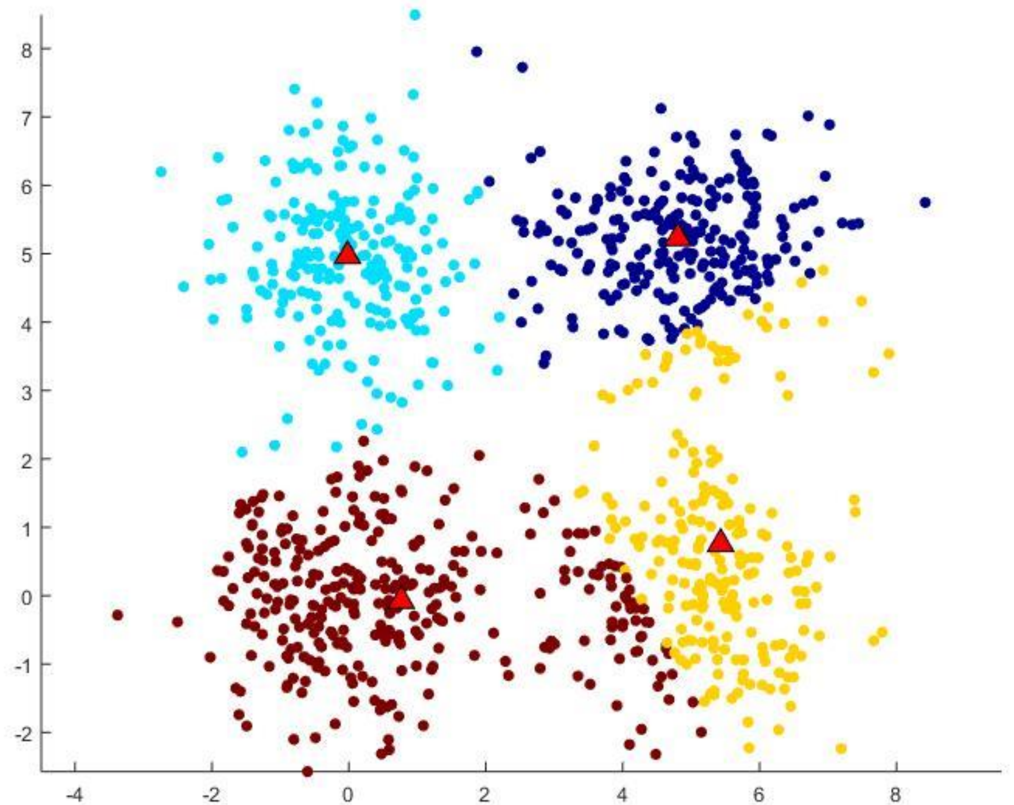
- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

---

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

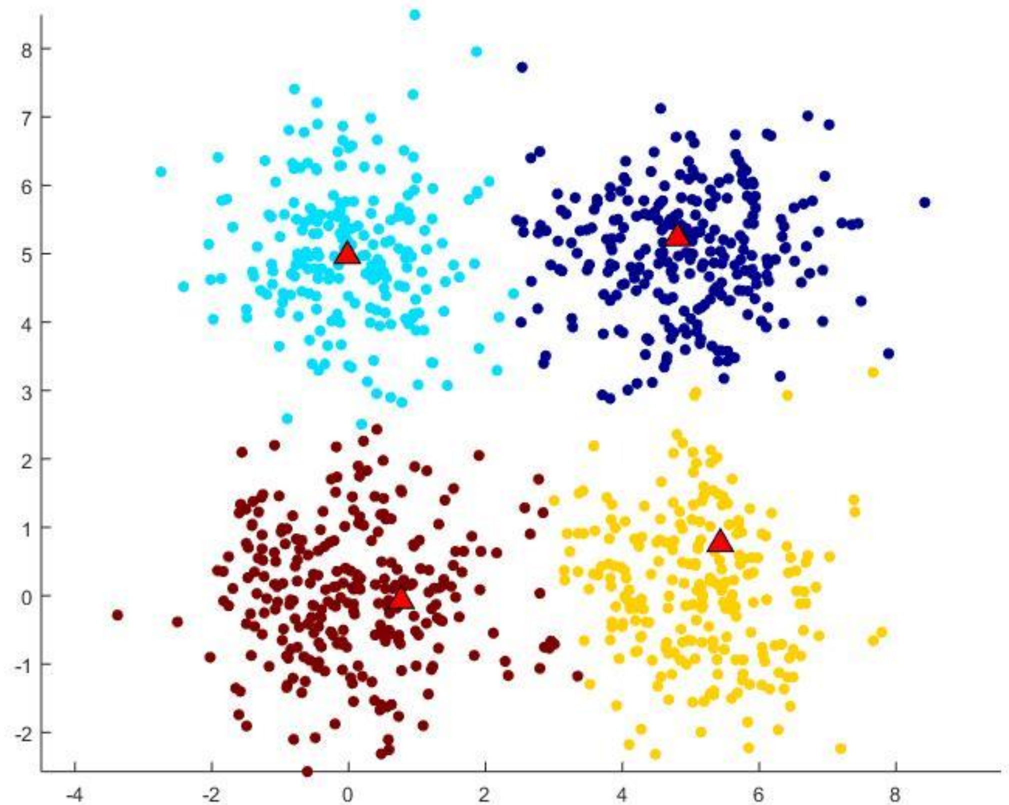
3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

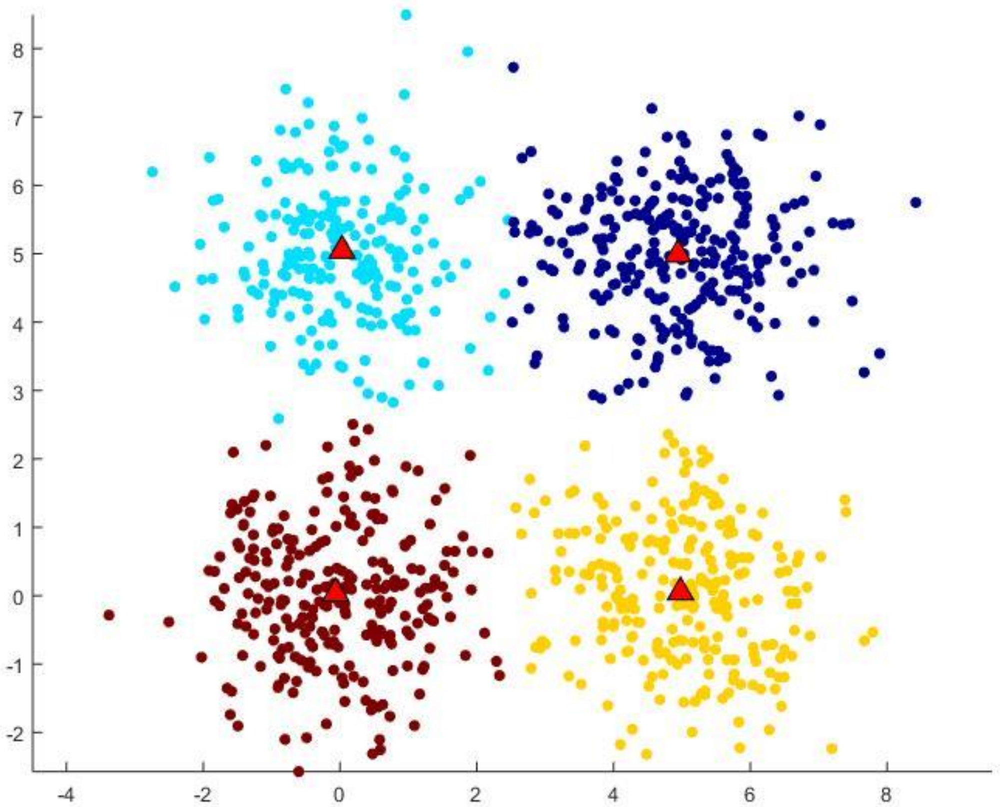
3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

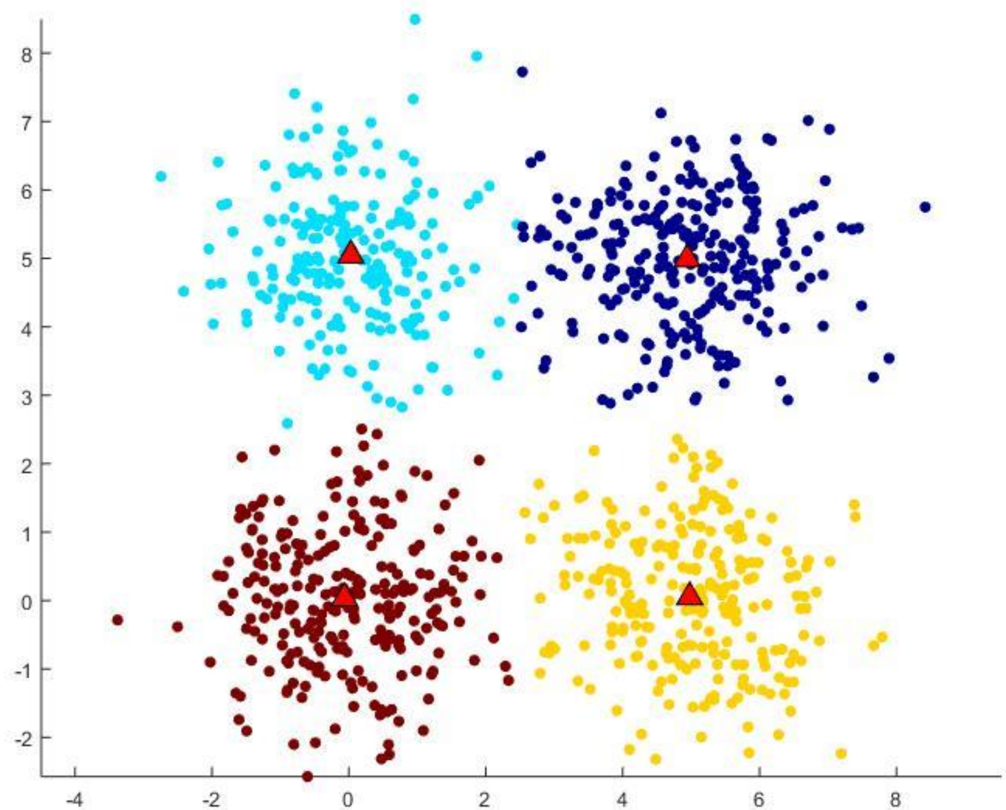
3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^{N} \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random

- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^{K} \subset \mathbb{R}^2$

---

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:

- $\boldsymbol{c}_k = \dfrac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

---

4. Repeat **2.** and **3.** until convergence

The K-means Algorithm

K-means: What does it do?

- It decreases this **total loss** at each iteration:

$$\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) = \sum_{k=1}^{K} \sum_{n \in \mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$$

K-means: What does it do?

- It decreases this **total loss** at each iteration:

$$\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) = \sum_{k=1}^{K} \sum_{n \in \mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$$

▸ Non-Convex
(Depends on init.)

K-means: What does it do?

- It decreases this **total loss** at each iteration:

$$\mathcal{C}(\mathbf{X},\mathcal{G},\theta) = \sum_{k=1}^{K} \sum_{n\in\mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1,\dots,\boldsymbol{c}_K\}$$

▸ Non-Convex
(Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ

K-means: What does it do?

- It decreases this **total loss** at each iteration:

$$\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) = \sum_{k=1}^{K} \sum_{n \in \mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$$

▸ Non-Convex (Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ
- Probabilistic / Generative interpretation *(where is* $\boldsymbol{z}$ *) ?*

K-means: What does it do?

- It decreases this **total loss** at each iteration:

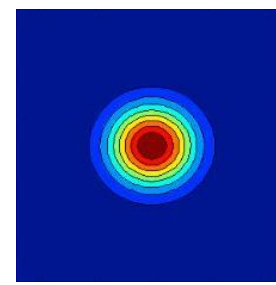
$$\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) = \sum_{k=1}^{K} \sum_{n \in \mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$$

▸ Non-Convex (Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ

- Probabilistic / Generative interpretation *(where is $\boldsymbol{z}$) ?*

$$\begin{cases} p(z_{n,k}=1) = \frac{1}{K} \quad \text{where } \boldsymbol{z}_n = \begin{array}{r} \\ \\ \\ \\ \\ k \rightarrow \\ \\ \end{array}\begin{bmatrix} \\ \\ \\ \\ \\ 1 \\ \\ \end{bmatrix} \in \{0,1\}^K \ ! \\ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{c}_k, \mathbf{I}) \text{ (Gaussian centered on } \boldsymbol{c}_k \text{)} \end{cases}$$



K-means: What does it do?

- It decreases this **total loss** at each iteration:

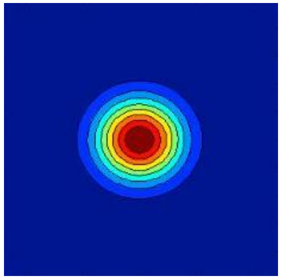
$$\mathcal{C}(\mathbf{X},\mathcal{G},\theta) = \sum_{k=1}^{K} \sum_{n\in\mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1,\dots,\boldsymbol{c}_K\}$$

▸ Non-Convex (Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ

- Probabilistic / Generative interpretation *(where is $\boldsymbol{z}$) ?*

$$\begin{cases} p(z_{n,k}=1) = \dfrac{1}{K} \quad \text{where } \boldsymbol{z}_n = \begin{array}{r} \\ \\ \\ \\ \\ k \rightarrow \\ \\ \end{array}\begin{bmatrix} \\ \\ \\ \\ \\ 1 \\ \\ \end{bmatrix} \in \{0,1\}^K \ ! \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\mathbf{I}) \ \text{(Gaussian centered on } \boldsymbol{c}_k \text{)} \end{cases}$$



$$\Rightarrow p_\theta(\mathbf{X},\mathbf{Z}) = \prod_{n=1}^{N}\sum_{k=1}^{K} z_{n,k}\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\mathbf{I})$$

K-means: What does it do?

- It decreases this **total loss** at each iteration:

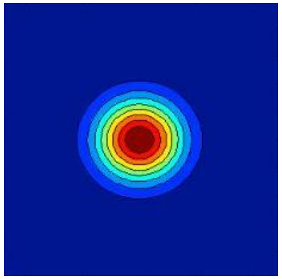
$$\mathcal{C}(\mathbf{X},\mathcal{G},\theta) = \sum_{k=1}^{K}\sum_{n\in\mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1,\dots,\boldsymbol{c}_K\}$$

▸ Non-Convex (Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ

- Probabilistic / Generative interpretation *(where is $\boldsymbol{z}$) ?*

$$\left\{\begin{array}{l} p(z_{n,k}=1) = \dfrac{1}{K} \quad \text{where } \boldsymbol{z}_n = \begin{array}{r} \\ \\ \\ \\ \\ k \rightarrow \\ \\ \end{array}\begin{bmatrix} \\ \\ \\ \\ \\ 1 \\ \\ \end{bmatrix} \in \{0,1\}^K \ ! \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\mathbf{I}) \ \text{(Gaussian centered on } \boldsymbol{c}_k\text{)} \end{array}\right.$$



$$\Rightarrow p_\theta(\mathbf{X},\mathbf{Z}) = \prod_{n=1}^{N}\sum_{k=1}^{K} z_{n,k}\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\mathbf{I})$$

$$\boxed{\mathcal{C}(\mathbf{X},\mathcal{G},\theta) \stackrel{c}{=} -\log p_\theta(\mathbf{Z}|\mathbf{X})}$$

K-means: What does it do?

- It decreases this **total loss** at each iteration:

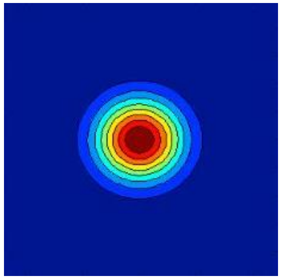
$$\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) = \sum_{k=1}^{K} \sum_{n \in \mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$$

▸ Non-Convex (Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ

- Probabilistic / Generative interpretation *(where is $\boldsymbol{z}$) ?*

$$\begin{cases} p(z_{n,k} = 1) = \dfrac{1}{K} \quad \text{where } \boldsymbol{z}_n = \begin{array}{r} \\ \\ \\ \\ \\ k \rightarrow \\ \\ \end{array} \begin{bmatrix} \\ \\ \\ \\ \\ 1 \\ \\ \end{bmatrix} \in \{0,1\}^K \; ! \\ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{c}_k, \mathbf{I}) \text{ (Gaussian centered on } \boldsymbol{c}_k \text{)} \end{cases}$$



$$\Rightarrow p_\theta(\mathbf{X}, \mathbf{Z}) = \prod_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{c}_k, \mathbf{I})$$

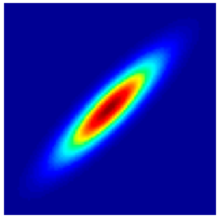
$$\boxed{\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) \overset{c}{=} -\log p_\theta(\mathbf{Z}|\mathbf{X})}$$

K-means can be seen as a *maximum a posteriori* (MAP) approach!

Gaussian mixture models: a generalization

$$\begin{cases} p(z_{n,k}=1)=\pi_k, \quad \sum_{k=1}^K \pi_k = 1 \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{c}_k,\Sigma_k) \end{cases}$$

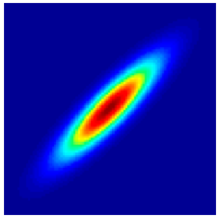
$$\Rightarrow p_\theta(\boldsymbol{x}_n) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\Sigma_k)$$



$$\theta = \{\pi_1,\dots,\pi_K, \boldsymbol{c}_1,\dots,\boldsymbol{c}_K, \Sigma_1,\dots,\Sigma_K\}$$

Gaussian mixture models: a generalization

$$\begin{cases} p(z_{n,k}=1)=\pi_k, \quad \sum_{k=1}^{K}\pi_k = 1 \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{c}_k,\Sigma_k) \end{cases}$$



$$\theta = \{\pi_1,\dots,\pi_K, \boldsymbol{c}_1,\dots,\boldsymbol{c}_K, \Sigma_1,\dots,\Sigma_K\}$$

$$\Rightarrow p_\theta(\boldsymbol{x}_n) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\Sigma_k)$$

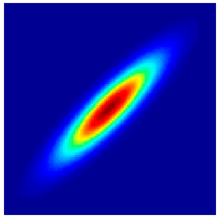
- Find θ that maximizes the *observed data log-likelihood*:

$$\mathcal{L}_\theta(\mathbf{X}) = \log p_\theta(\mathbf{X}) = \sum_n \log p_\theta(\boldsymbol{x}_n)$$

Gaussian mixture models: a generalization

$$\begin{cases} p(z_{n,k}=1)=\pi_k, \quad \sum_{k=1}^K \pi_k = 1 \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{c}_k,\Sigma_k) \end{cases}$$

$$\Rightarrow p_\theta(\boldsymbol{x}_n) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\Sigma_k)$$



$$\theta = \{\pi_1,\ldots,\pi_K, \boldsymbol{c}_1,\ldots,\boldsymbol{c}_K, \Sigma_1,\ldots,\Sigma_K\}$$

- Find θ that maximizes the *observed data log-likelihood*:

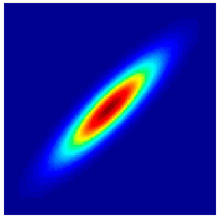
$$\mathcal{L}_\theta(\mathbf{X}) = \log p_\theta(\mathbf{X}) = \sum_n \log p_\theta(\boldsymbol{x}_n)$$

…very hard to solve directly.

Gaussian mixture models: a generalization

$$\begin{cases} p(z_{n,k}=1)=\pi_k, \quad \sum_{k=1}^{K}\pi_k = 1 \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{c}_k,\Sigma_k) \end{cases}$$

$$\Rightarrow p_\theta(\boldsymbol{x}_n) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\Sigma_k)$$



$$\theta = \{\pi_1,\dots,\pi_K, \boldsymbol{c}_1,\dots,\boldsymbol{c}_K, \Sigma_1,\dots,\Sigma_K\}$$

- Find θ that maximizes the *observed data log-likelihood*:

$$\mathcal{L}_\theta(\mathbf{X}) = \log p_\theta(\mathbf{X}) = \sum_n \log p_\theta(\boldsymbol{x}_n)$$

…very hard to solve directly.

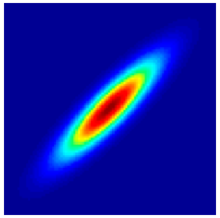
- The **Expectation-Maximization** (EM) algorithm iteratively maximizes the *expected complete-data log-likelihood* instead:

$$\theta^{(i+1)} = \operatorname*{argmax}_\theta \ \mathbb{E}_{p_{\theta^{(i)}}(Z|X)}\{\log p_\theta(\mathbf{X},\mathbf{Z})\}$$

Gaussian mixture models: a generalization

$$\begin{cases} p(z_{n,k}=1)=\pi_k, \quad \sum_{k=1}^{K}\pi_k = 1 \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{c}_k,\Sigma_k) \end{cases}$$

$$\Rightarrow p_\theta(\boldsymbol{x}_n) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\Sigma_k)$$



$$\theta = \{\pi_1,\dots,\pi_K, \boldsymbol{c}_1,\dots,\boldsymbol{c}_K, \Sigma_1,\dots,\Sigma_K\}$$

- Find θ that maximizes the *observed data log-likelihood*:

$$\mathcal{L}_\theta(\mathbf{X}) = \log p_\theta(\mathbf{X}) = \sum_n \log p_\theta(\boldsymbol{x}_n)$$

…very hard to solve directly.

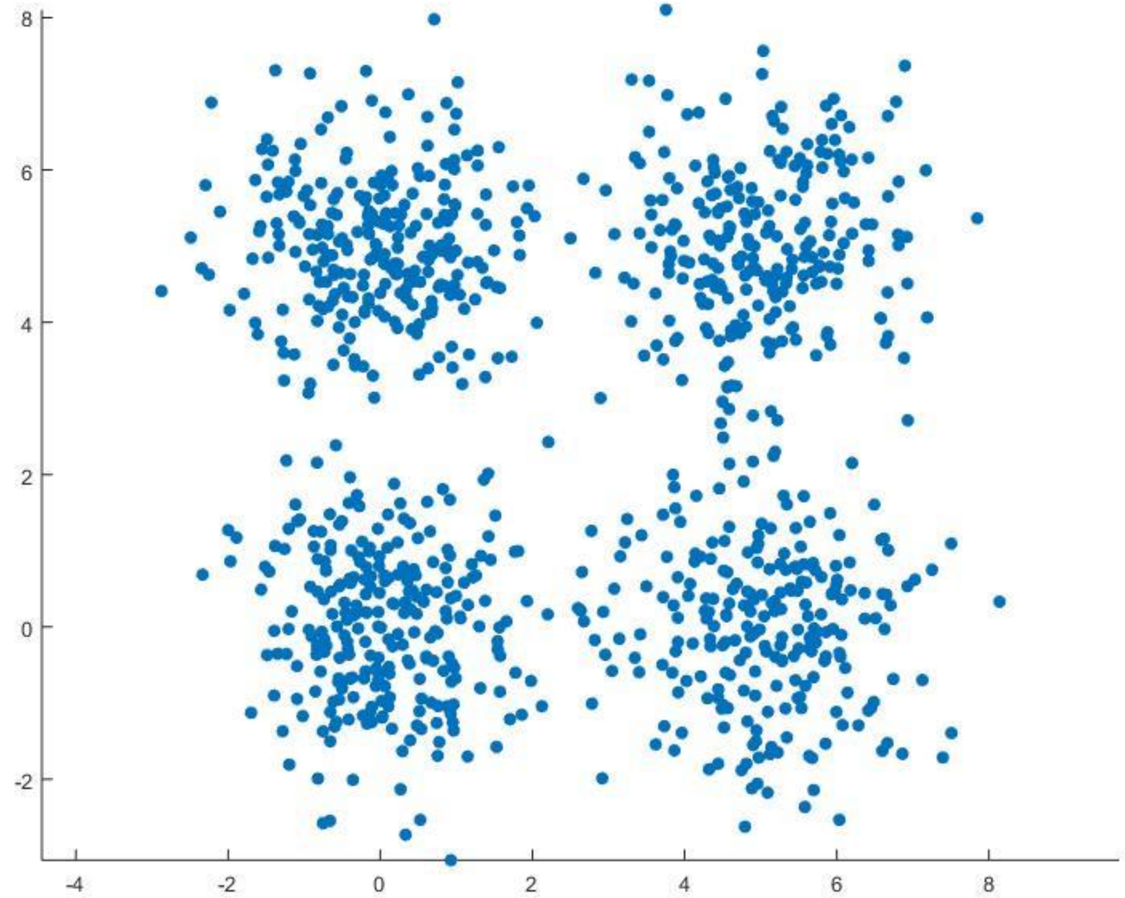
- The **Expectation-Maximization** (EM) algorithm iteratively maximizes the *expected complete-data log-likelihood* instead:

$$\theta^{(i+1)} = \operatorname*{argmax}_\theta \ \mathbb{E}_{p_{\theta^{(i)}}(Z|X)}\{\log p_\theta(\mathbf{X},\mathbf{Z})\}$$

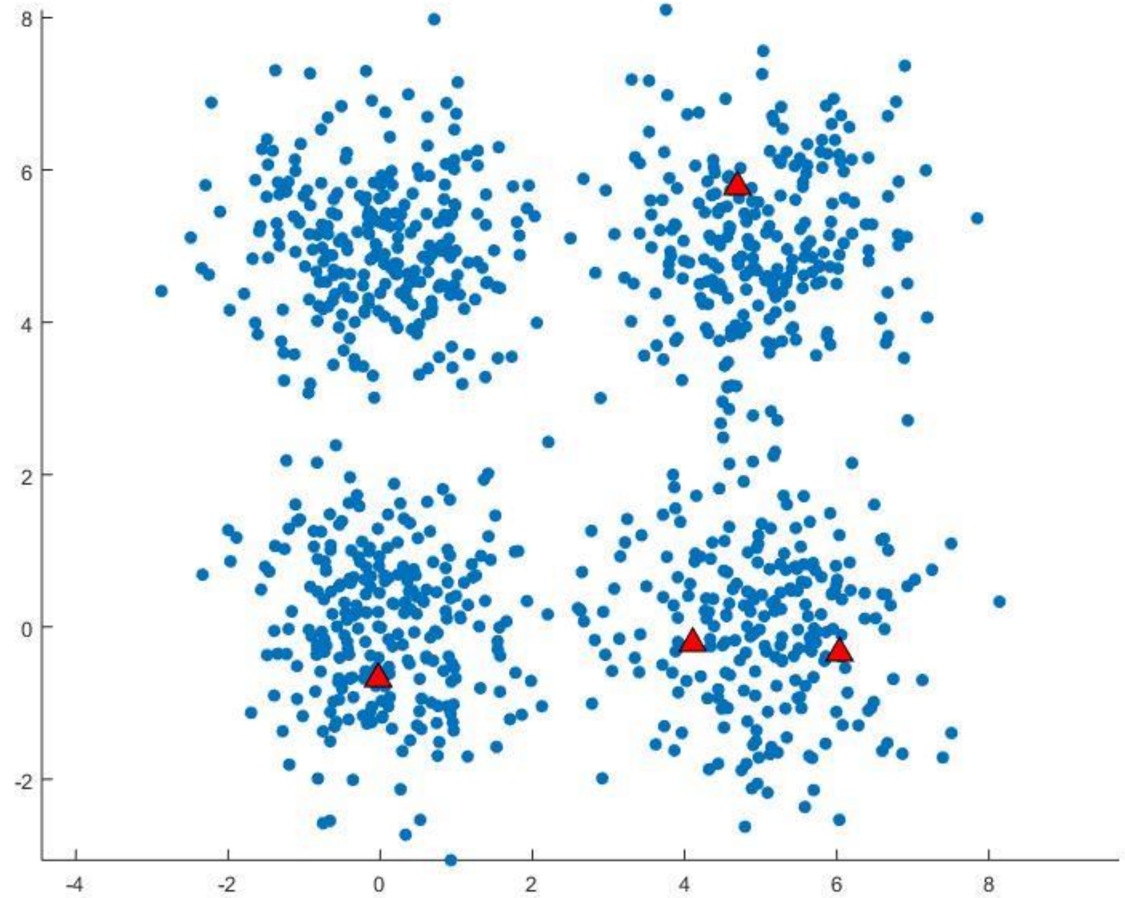
Repeat until convergence:

- $r_{n,k}^{(i)} = p_{\theta^{(i-1)}}(z_{n,k}=1|\boldsymbol{x}_n)$
- $\pi_k^{(i)} = \frac{1}{N}\sum_n r_{n,k}^{(i)} = \frac{\bar{r}_k}{N}$
- $\boldsymbol{c}_k^{(i)} = \frac{1}{\bar{r}_k}\sum_n r_{n,k}^{(i)}\boldsymbol{x}_n$
- $\Sigma_k^{(i)} = \frac{1}{\bar{r}_k}\sum_n r_{n,k}^{(i)}(\boldsymbol{x}_n - \boldsymbol{c}_k^{(i)}) \cdot (\boldsymbol{x}_n - \boldsymbol{c}_k^{(i)})^\top$

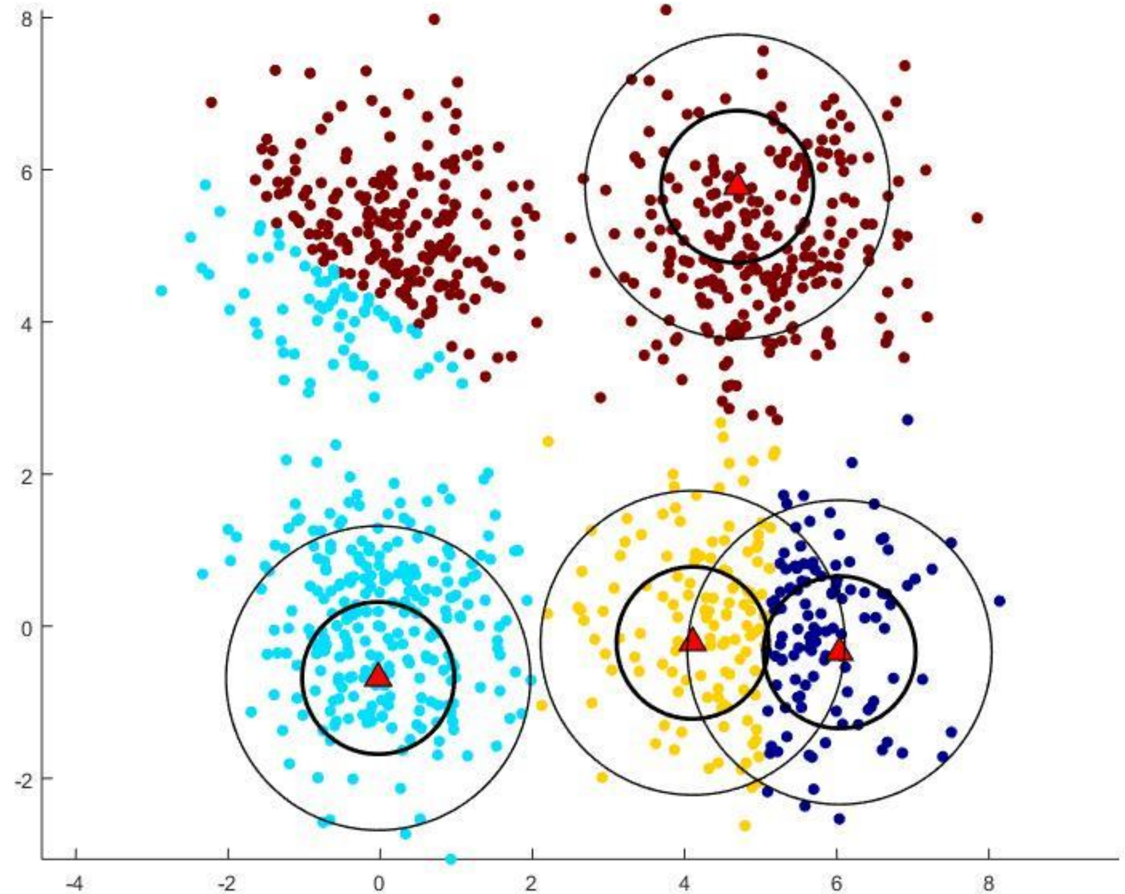
Gaussian Mixture Model Expectation-Maximization (GMM EM)



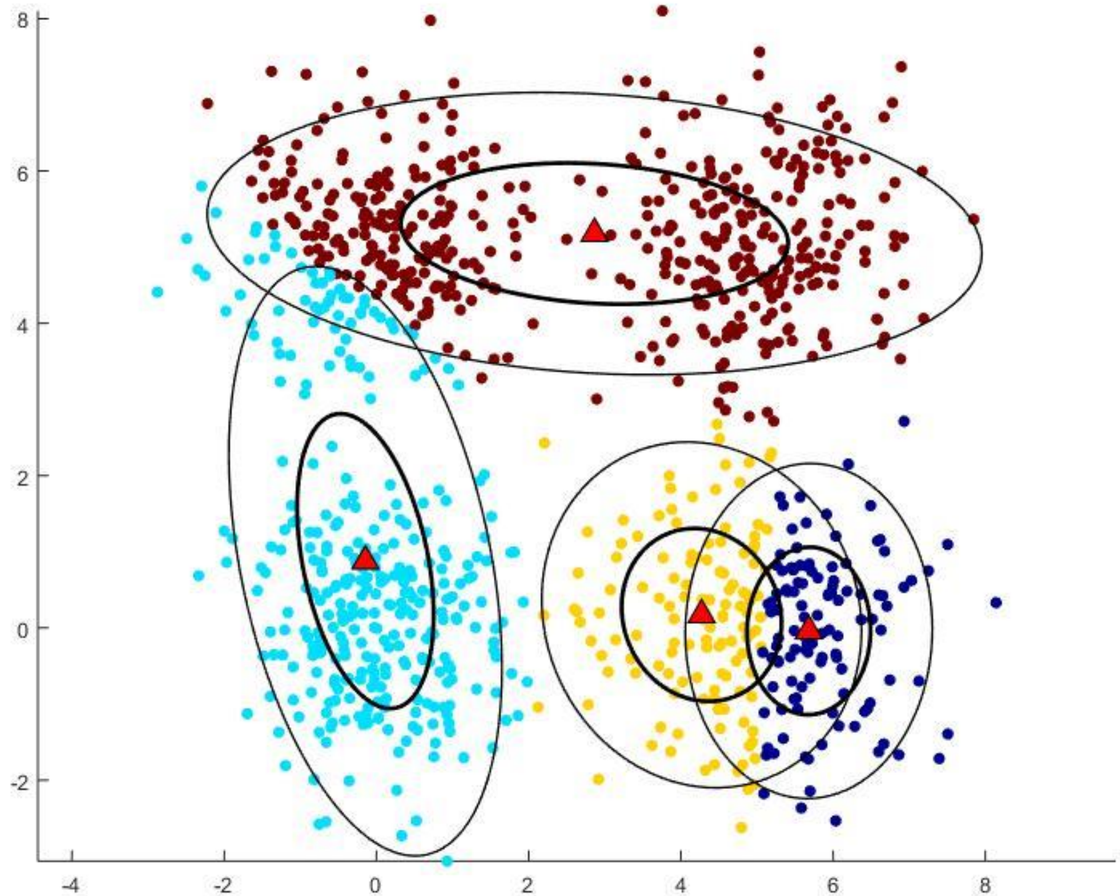
Gaussian Mixture Model Expectation-Maximization (GMM EM)



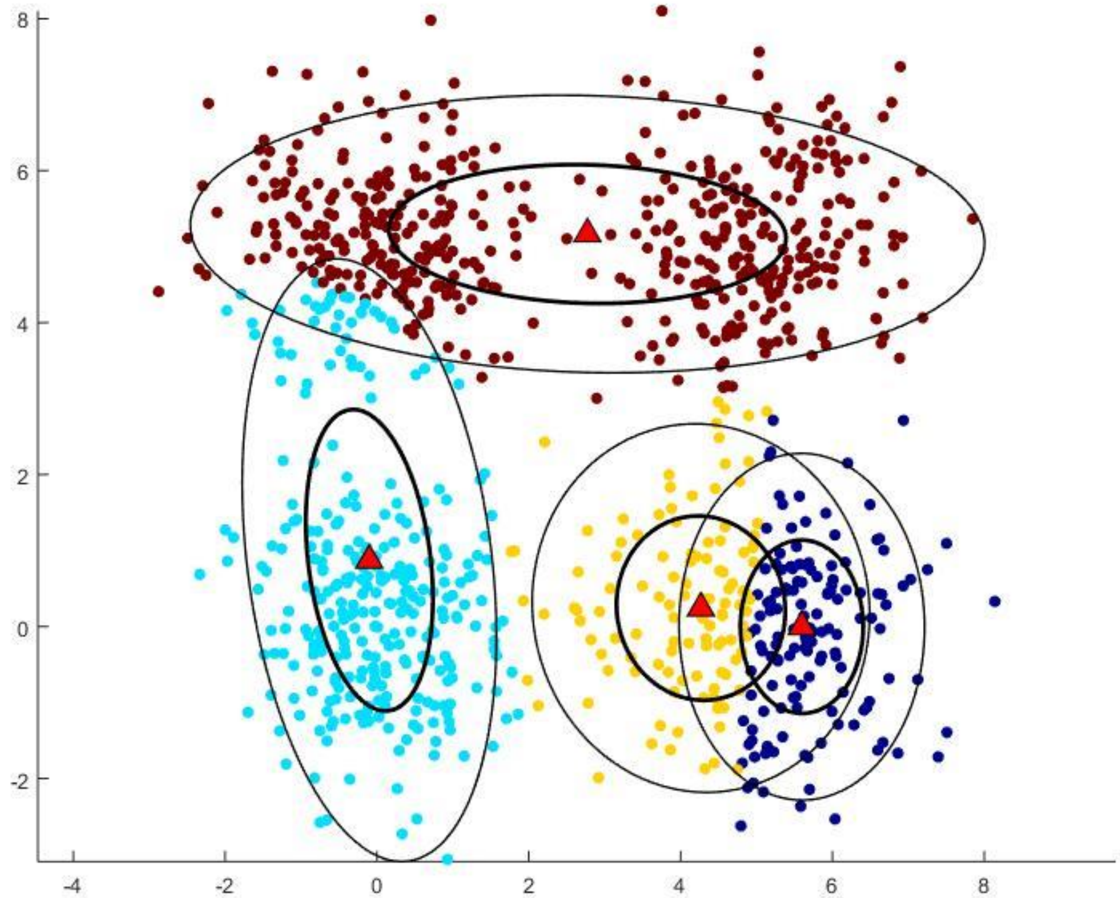
Gaussian Mixture Model Expectation-Maximization (GMM EM)



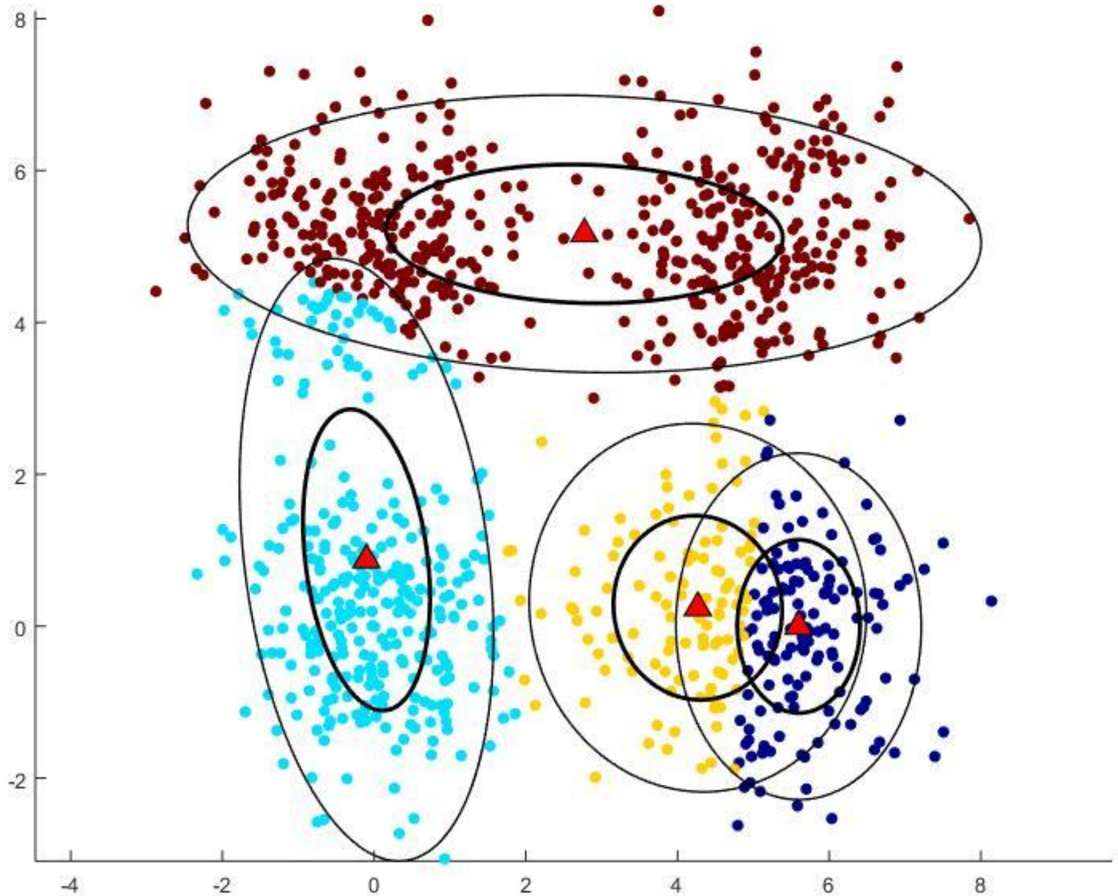
Gaussian Mixture Model Expectation-Maximization (GMM EM)



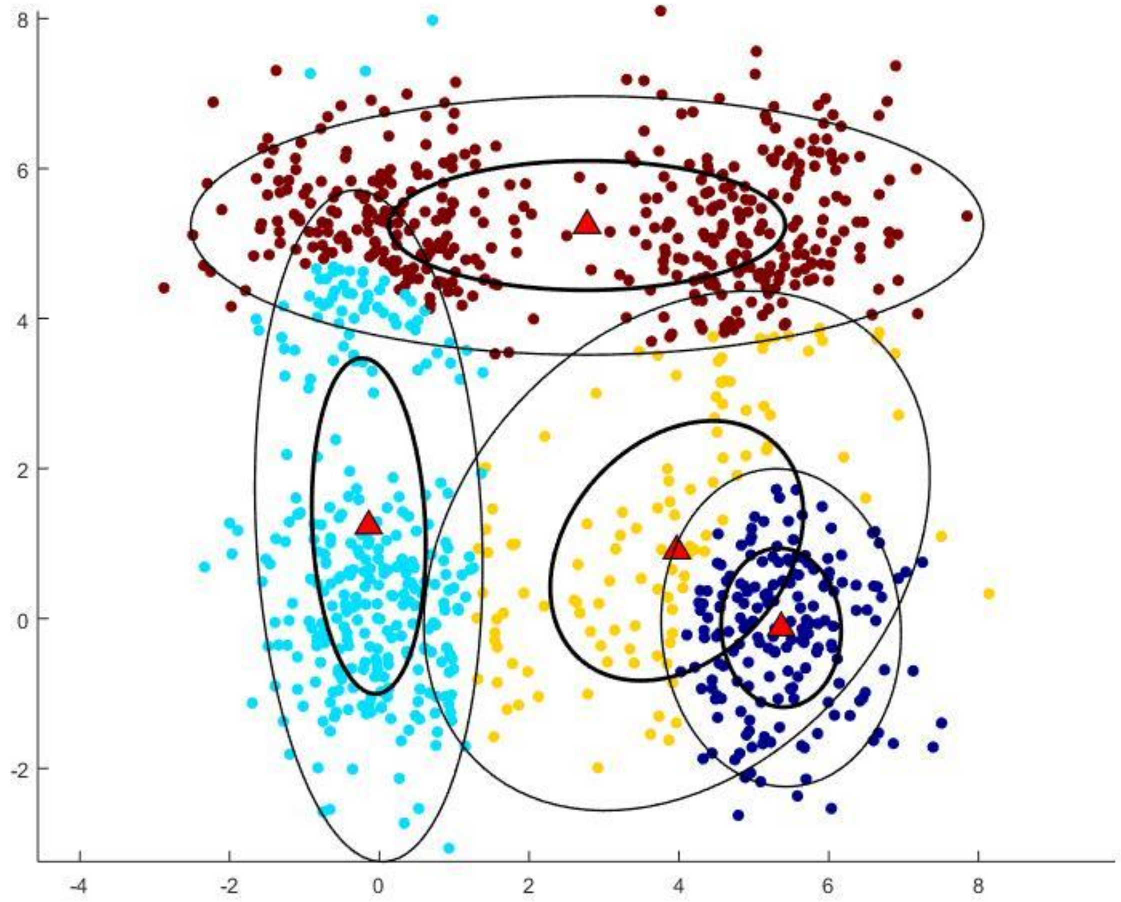
Gaussian Mixture Model Expectation-Maximization (GMM EM)



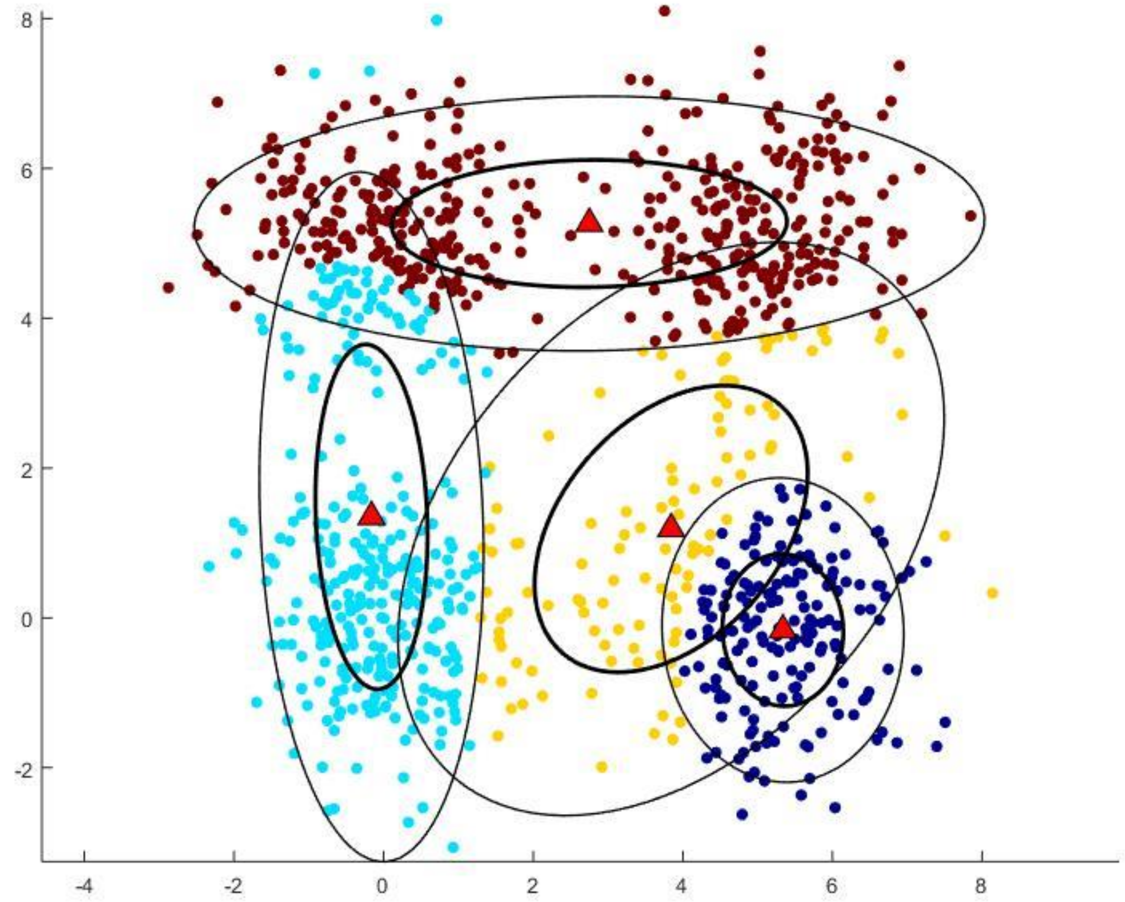
Gaussian Mixture Model Expectation-Maximization (GMM EM)



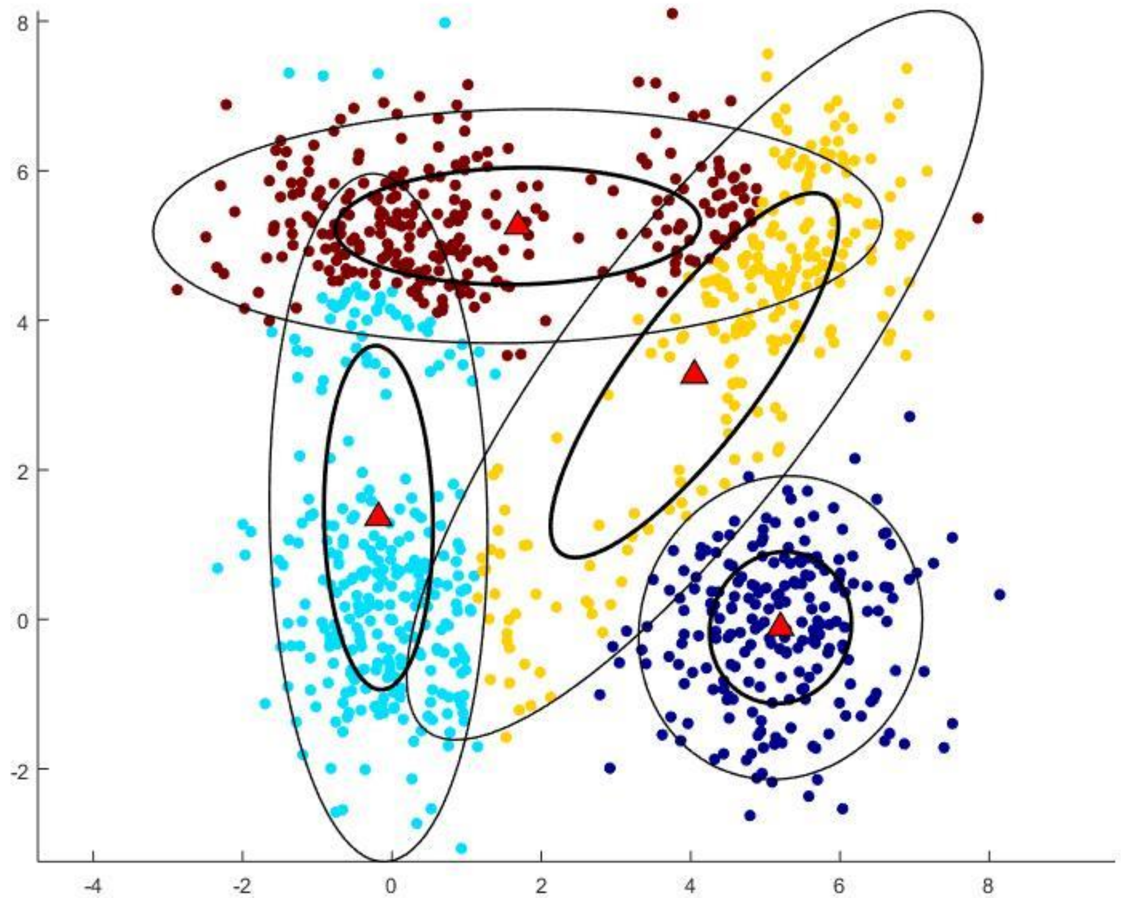
Gaussian Mixture Model Expectation-Maximization (GMM EM)



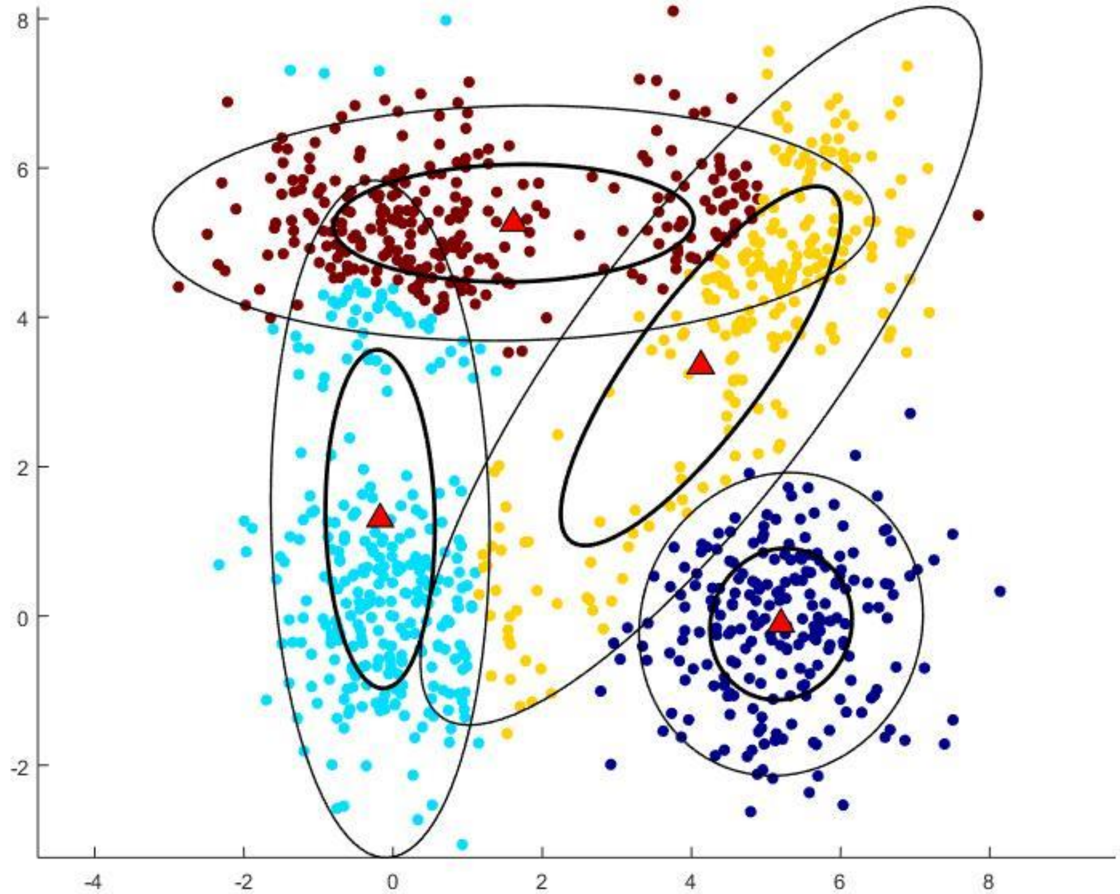
Gaussian Mixture Model Expectation-Maximization (GMM EM)



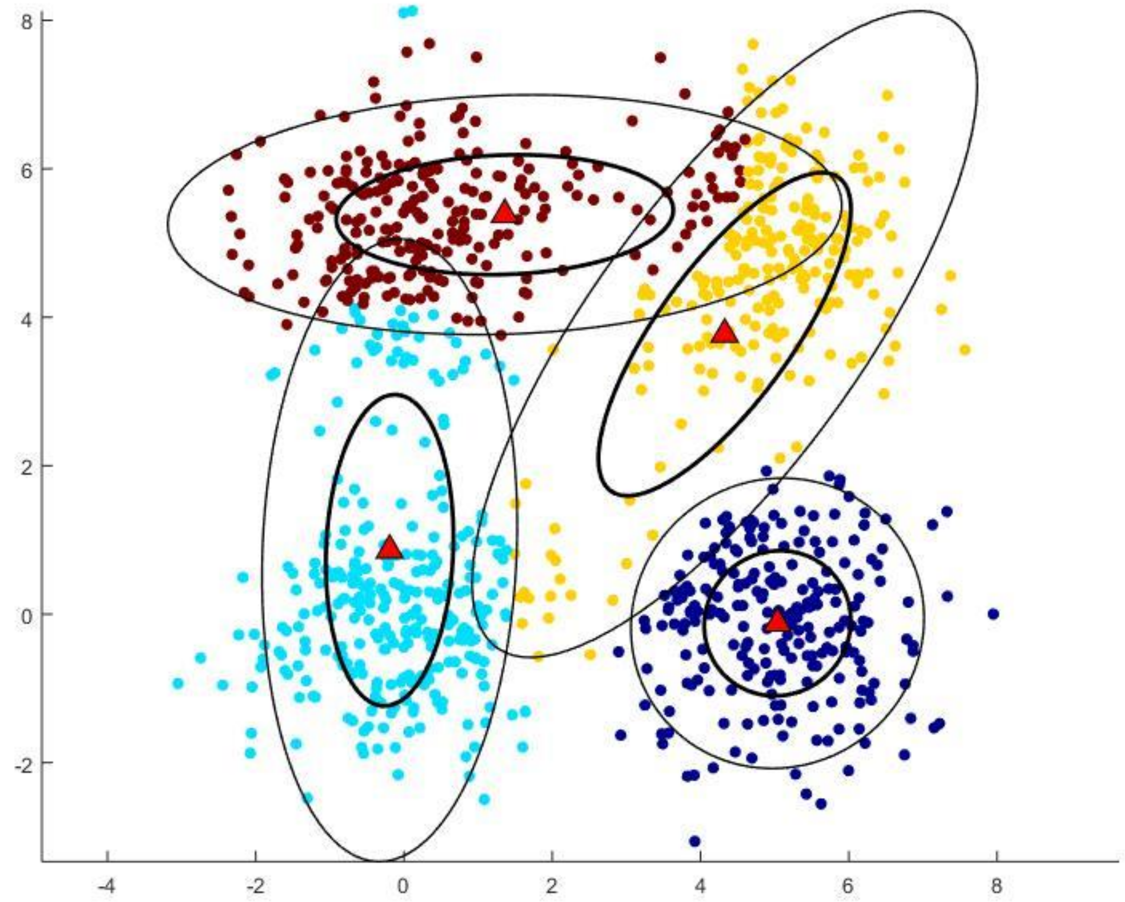
Gaussian Mixture Model Expectation-Maximization (GMM EM)



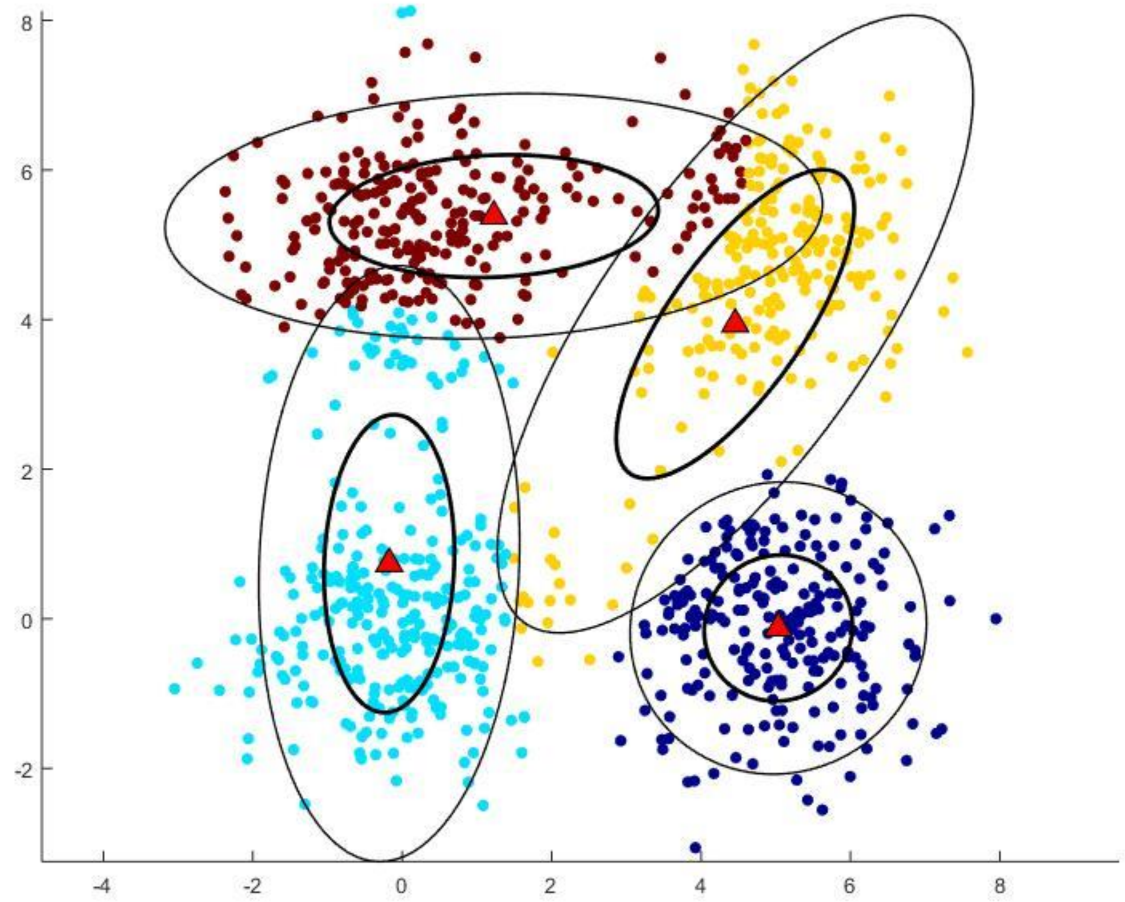
Gaussian Mixture Model Expectation-Maximization (GMM EM)



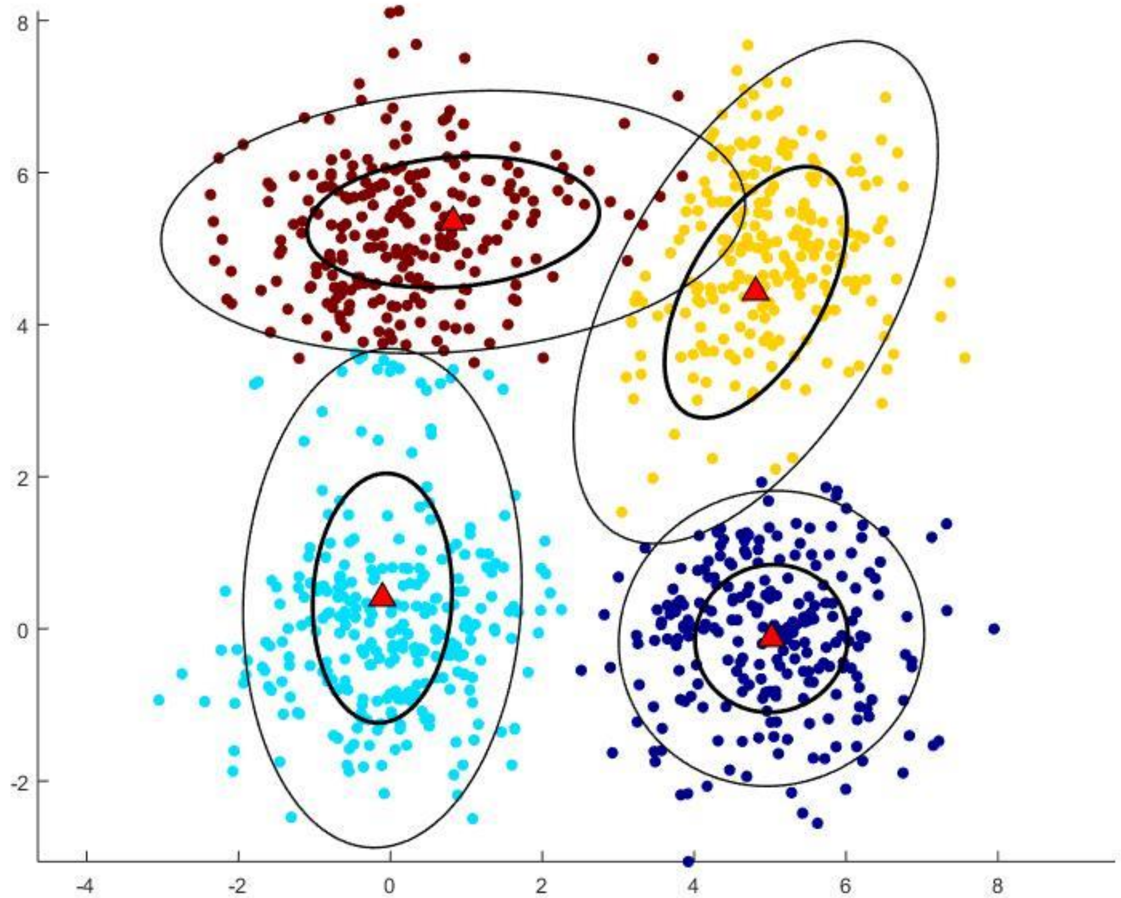
Gaussian Mixture Model Expectation-Maximization (GMM EM)



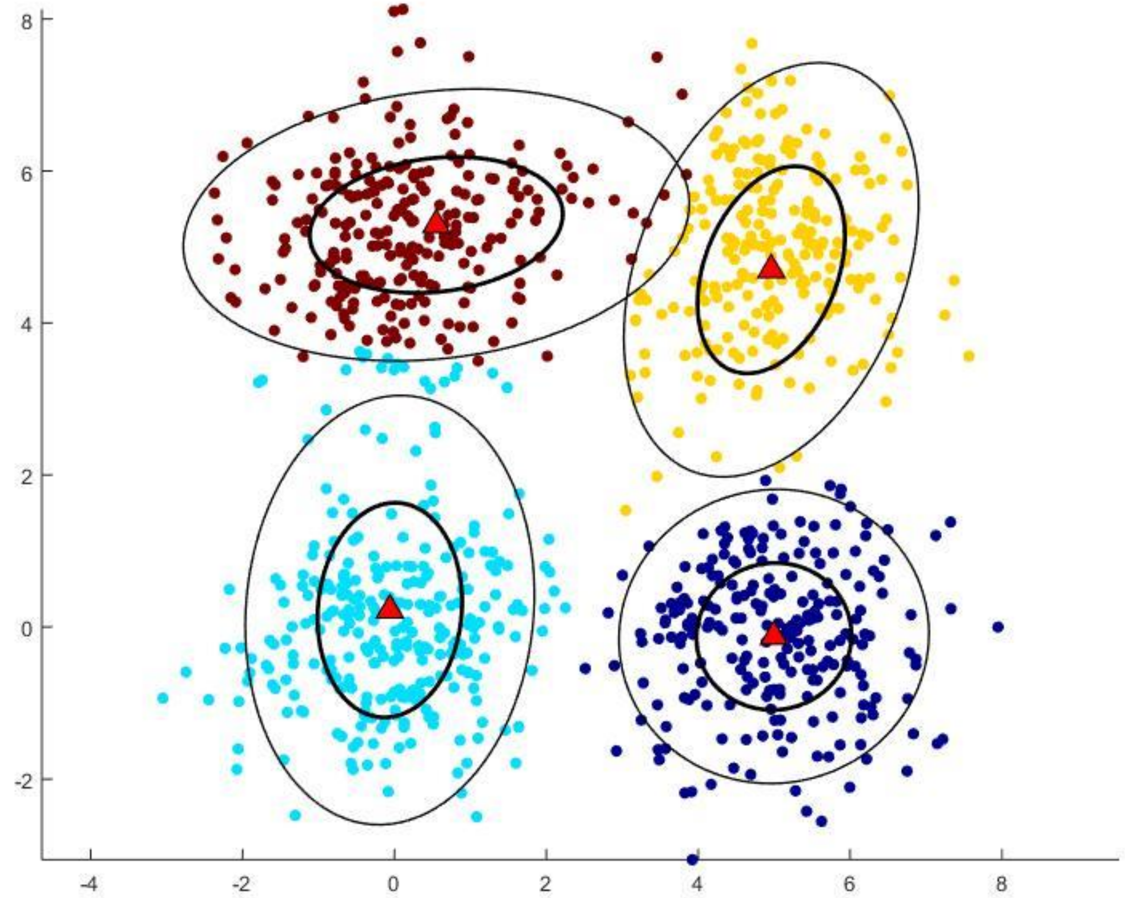
Gaussian Mixture Model Expectation-Maximization (GMM EM)



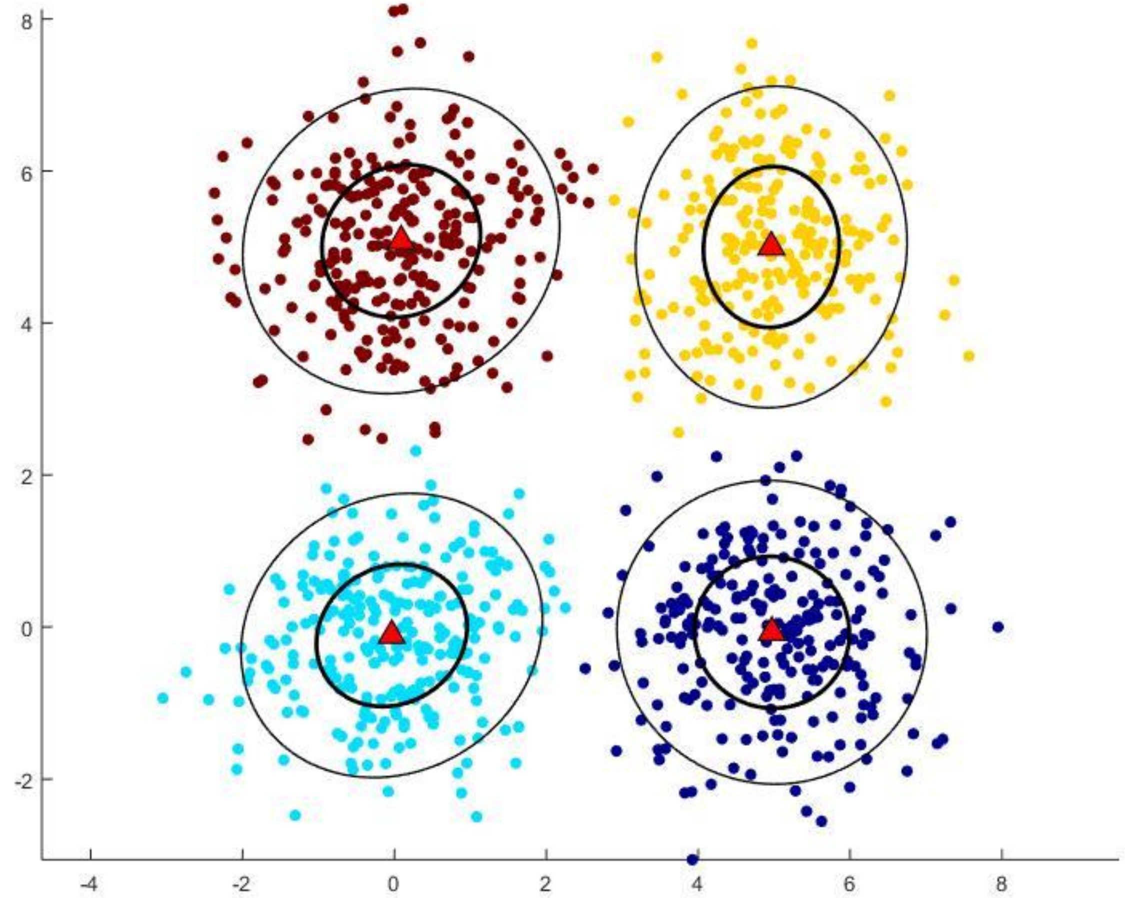
Gaussian Mixture Model Expectation-Maximization (GMM EM)



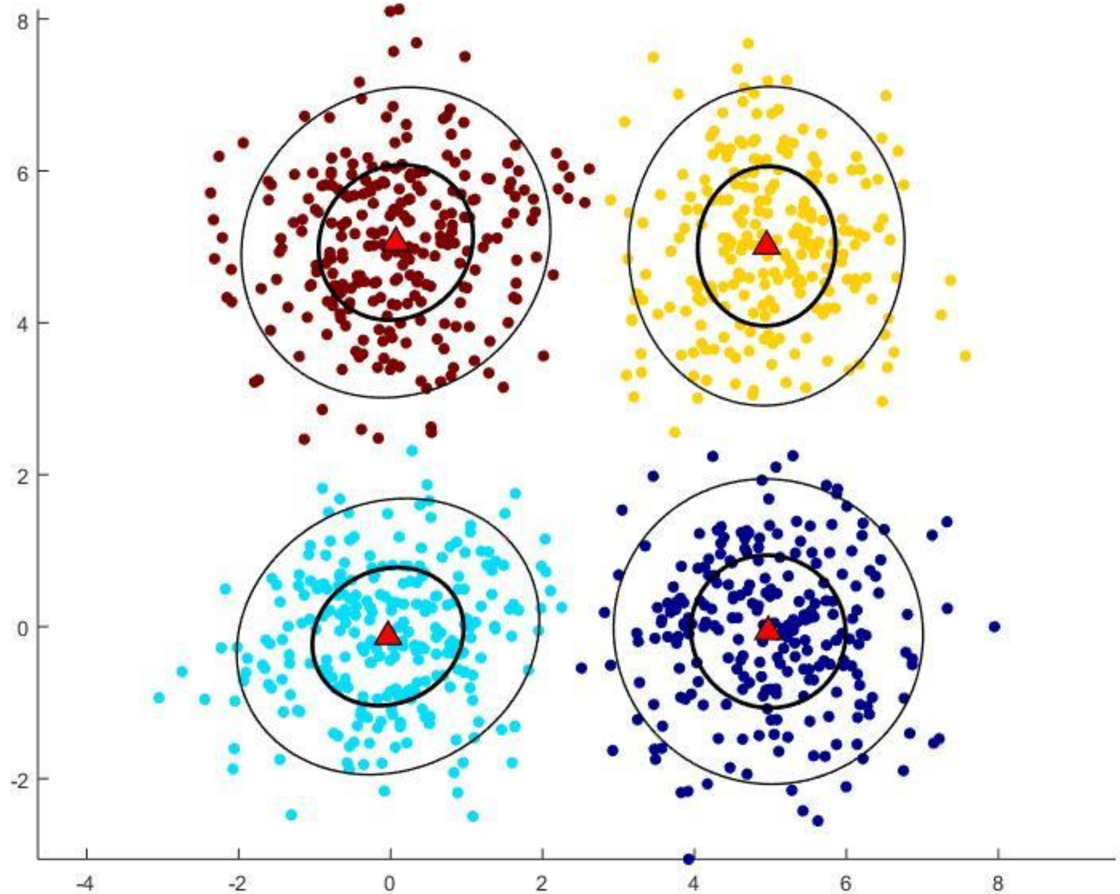
Gaussian Mixture Model Expectation-Maximization (GMM EM)



Gaussian Mixture Model Expectation-Maximization (GMM EM)



Gaussian Mixture Model Expectation-Maximization (GMM EM)



And in the *Deep Learning Era* …

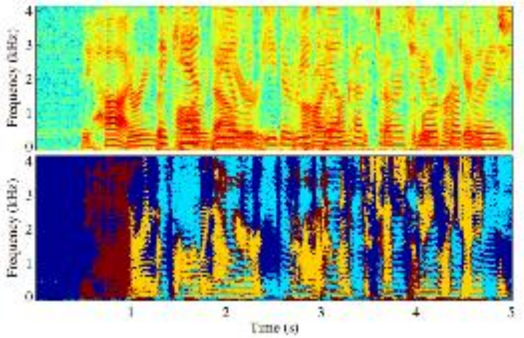
- ***Deep clustering:*** trains a **DNN** to project data to a feature space where K-means can be optimally used

And in the *Deep Learning Era* …

- ***Deep clustering:*** trains a **DNN** to project data to a feature space where K-means can be optimally used

- Application to blind speech source separation:

John R. Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. **"Deep clustering: Discriminative embeddings for segmentation and separation."** In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 31-35. IEEE, 2016.

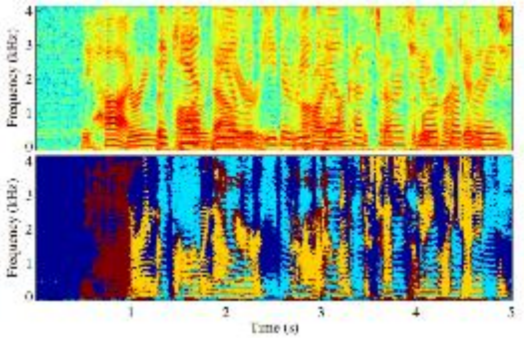


And in the *Deep Learning Era* …

- ***Deep clustering:*** trains a **DNN** to project data to a feature space where K-means can be optimally used

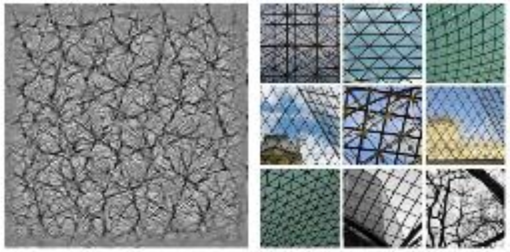
- Application to blind speech source separation:

John R. Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe. **"Deep clustering: Discriminative embeddings for segmentation and separation."** In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 31-35. IEEE, 2016.



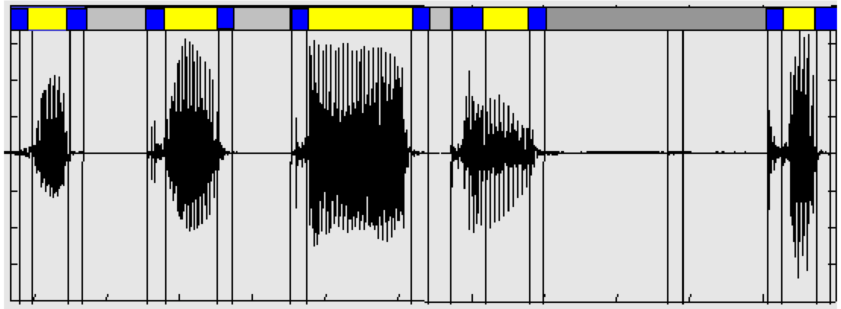
- Application to feature learning from images:

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. **"Deep clustering for unsupervised learning of visual features."** In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132-149. 2018.



Typical applications of clustering

Image segmentation ($\{x_n\}_{n=1}^N$ are local descriptors)



Audio segmentation ($\{x_n\}_{n=1}^N$ are sound segment)

Anti-spam filters

Data Quantification

DNA sequence analysis

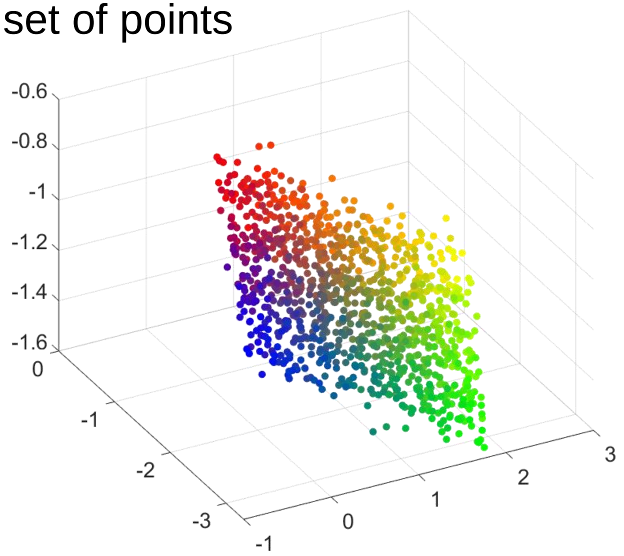
Medical imaging

Speech diarization

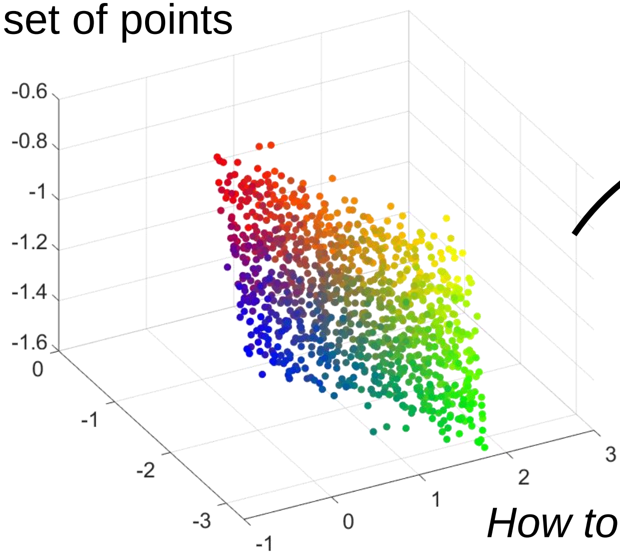
Species classificaton (biology)

Social network analysis

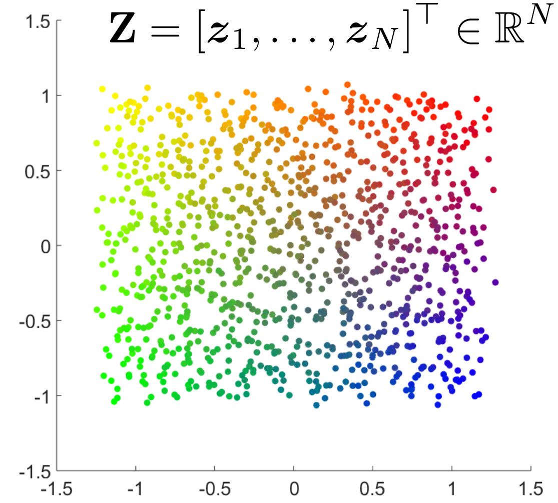
Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times 3}$ be a 3D set of points



Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times 3}$ be a 3D set of points

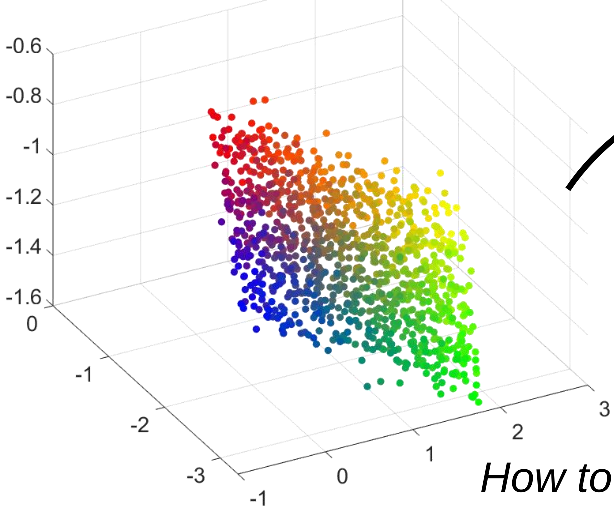


$\mathbf{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]^\top \in \mathbb{R}^{N \times 2}$

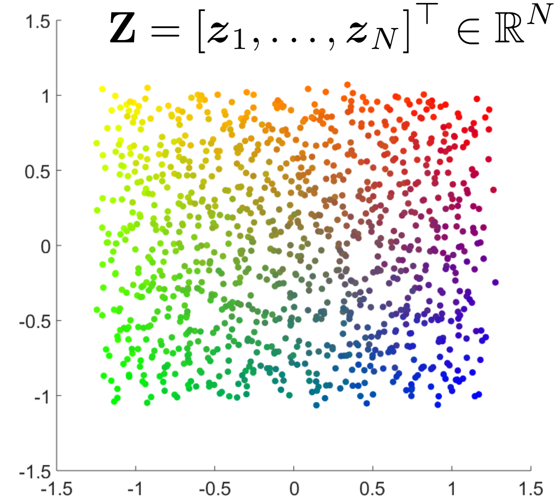


How to reduce its dimensionality while preserving most of its information?

Let $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times 3}$ be a 3D set of points



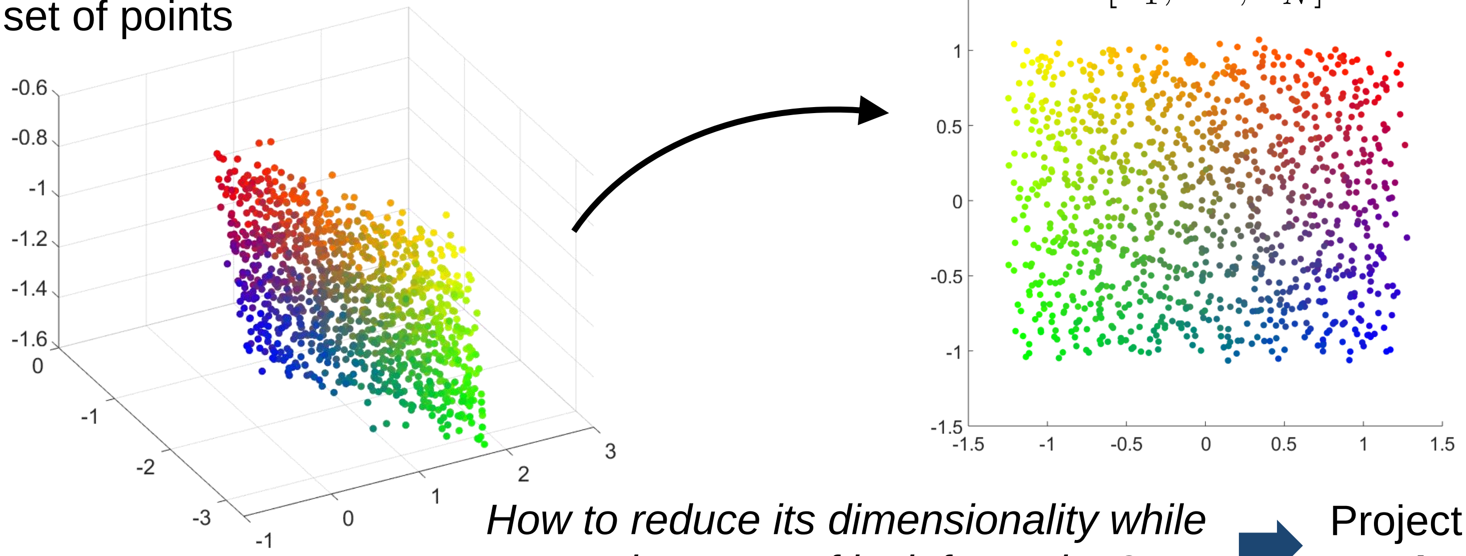
$\mathbf{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_N]^\top \in \mathbb{R}^{N \times 2}$



How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times 3}$ be a 3D set of points

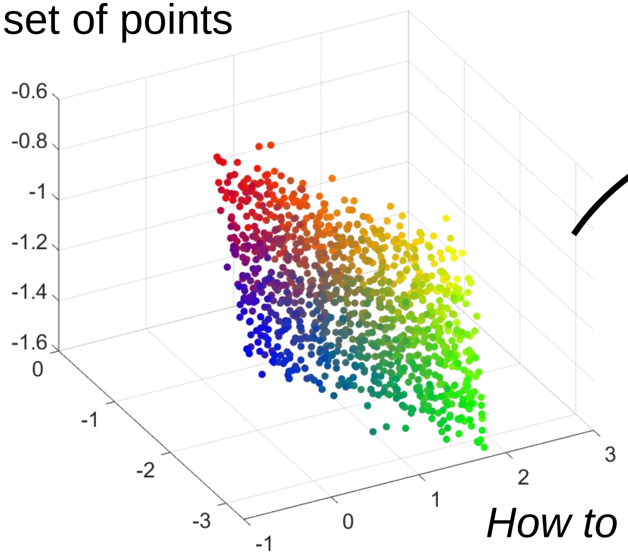
$\mathbf{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]^\top \in \mathbb{R}^{N\times 2}$



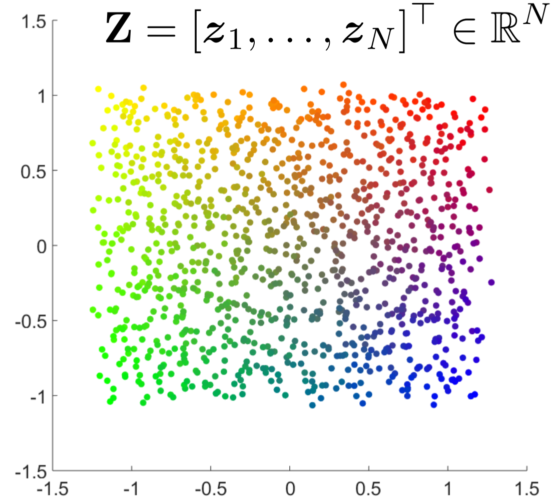
How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \ldots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

Let $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times 3}$ be a 3D set of points



$\mathbf{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_N]^\top \in \mathbb{R}^{N\times 2}$

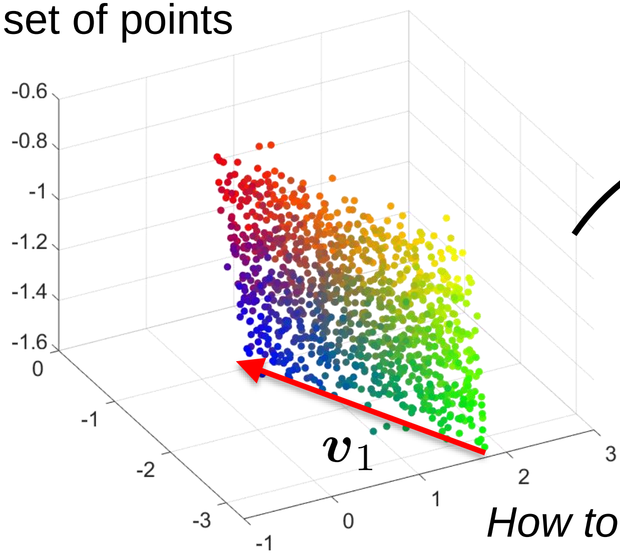


How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

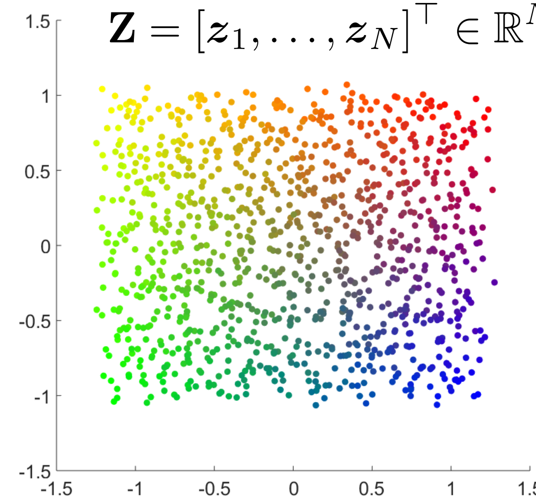
1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \dots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

- The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3\times 3}$.

Let $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times 3}$ be a 3D set of points



$\mathbf{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_N]^\top \in \mathbb{R}^{N\times 2}$

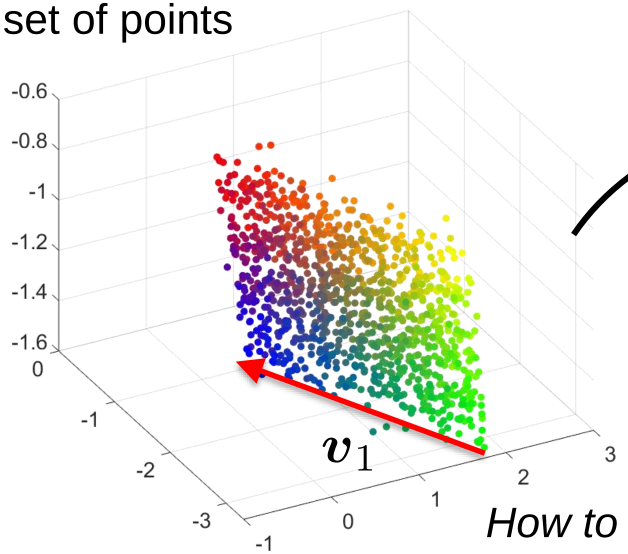


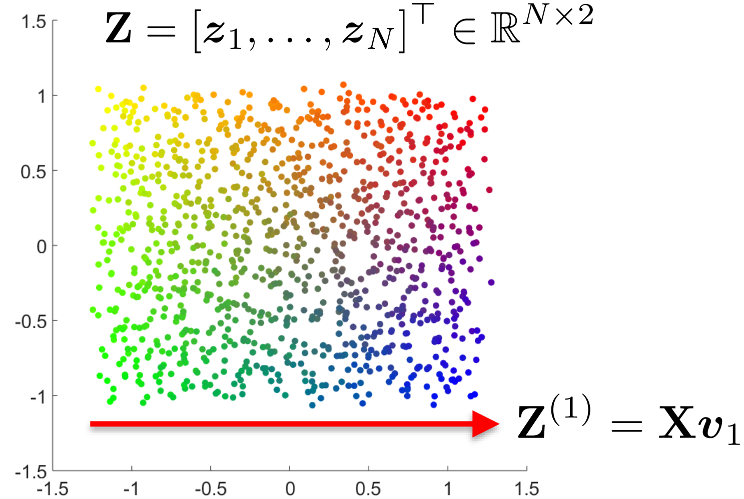
How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \dots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

- The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3\times 3}$.

Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times 3}$ be a 3D set of points



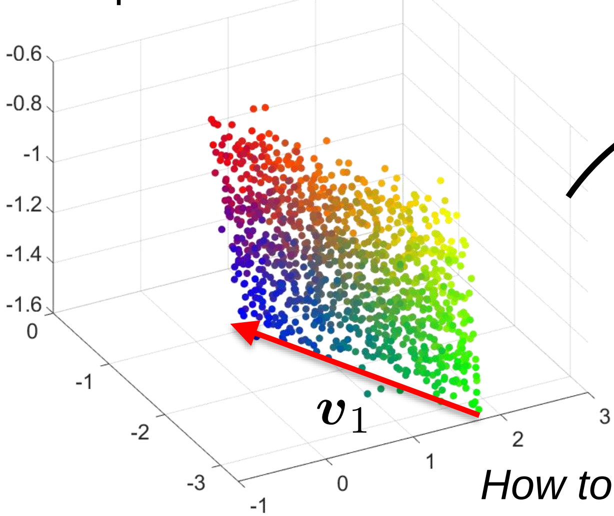


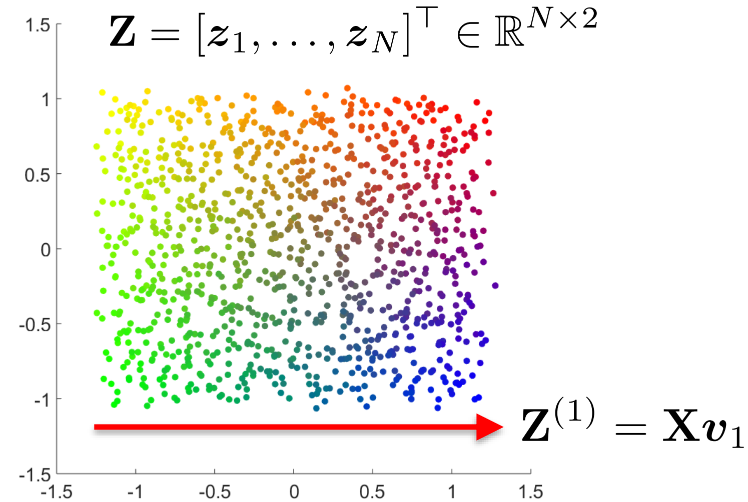
How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \ldots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

- The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3\times 3}$.

Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times 3}$ be a 3D set of points





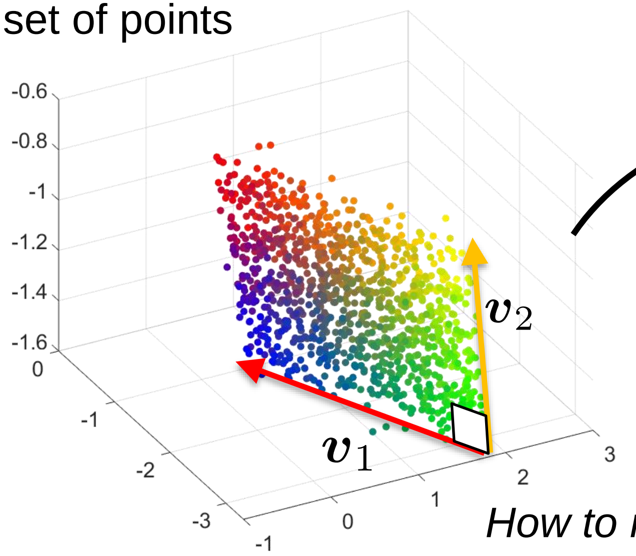
How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

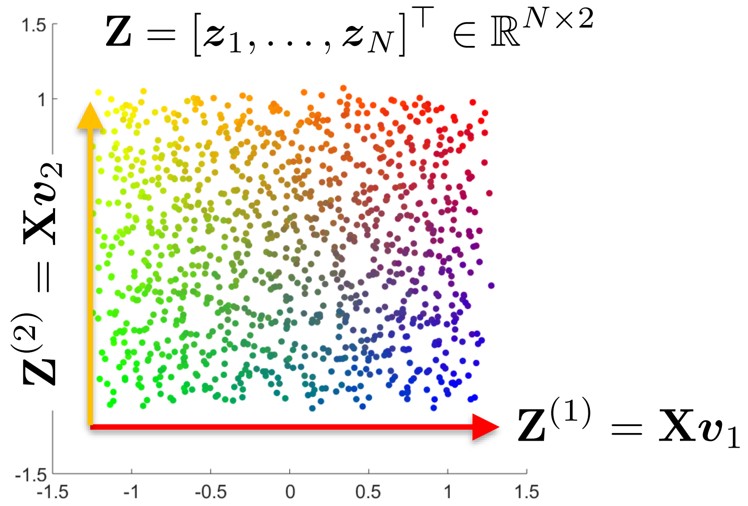
1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \ldots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^{N} |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

- ▸ The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3 \times 3}$.

2. Find $\boldsymbol{v}_2 \perp \boldsymbol{v}_1$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_2)$ is largest. ▸ Second dominant eigenvector of $\mathbf{C}$.

Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times 3}$ be a 3D set of points





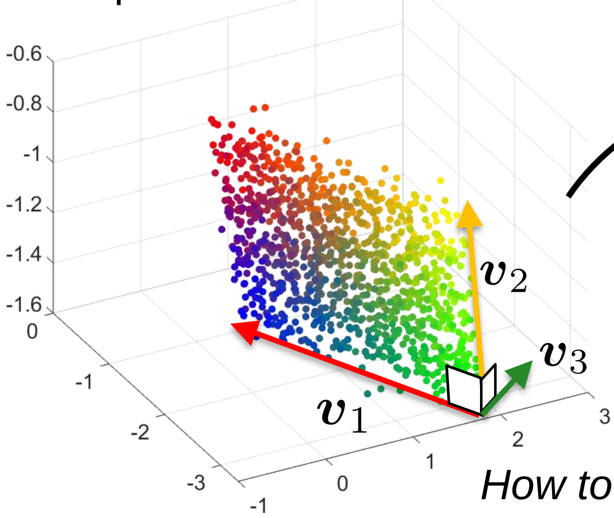
How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

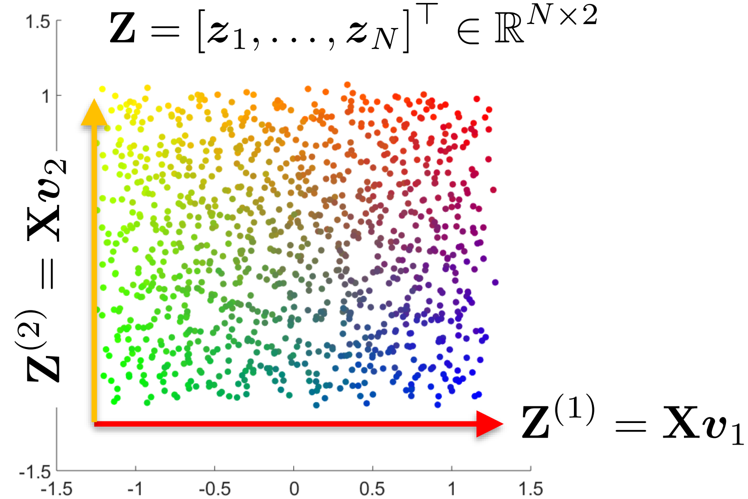
1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top\boldsymbol{x}_1, \ldots, \boldsymbol{v}_1^\top\boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top\boldsymbol{x}_n|^2$ is largest

▸ The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3\times 3}$.

2. Find $\boldsymbol{v}_2 \perp \boldsymbol{v}_1$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_2)$ is largest. ▸ Second dominant eigenvector of $\mathbf{C}$.

Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times 3}$ be a 3D set of points





How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \ldots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

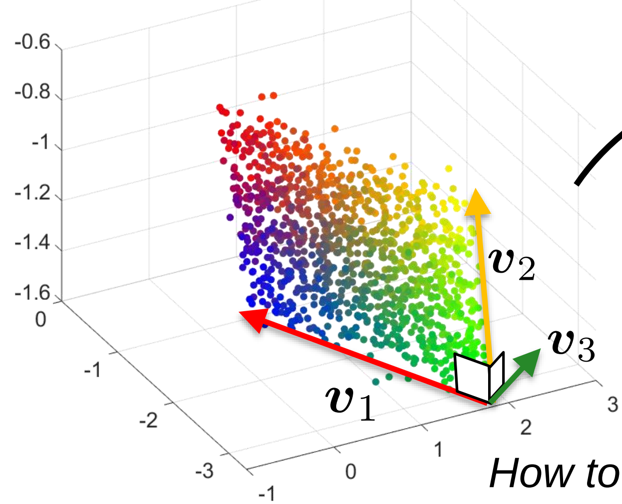
- The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3 \times 3}$.

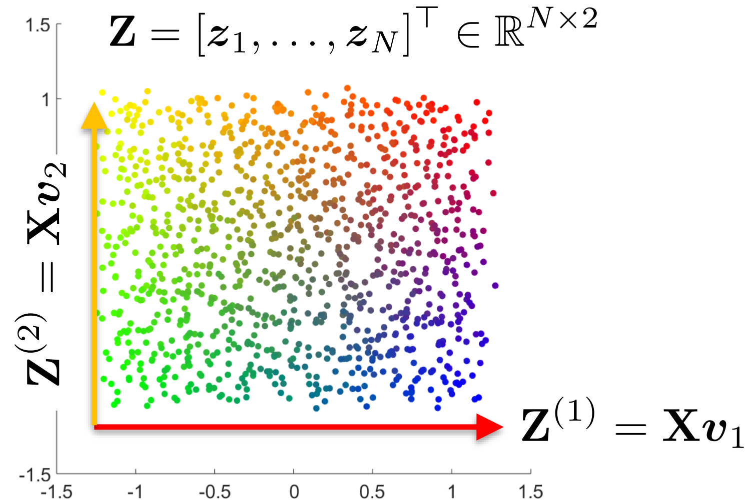
2. Find $\boldsymbol{v}_2 \perp \boldsymbol{v}_1$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_2)$ is largest. ▸ Second dominant eigenvector of $\mathbf{C}$.

3. Find $\boldsymbol{v}_3 \perp [\boldsymbol{v}_1, \boldsymbol{v}_2]$ … *etc.*

- The **Principal Axes** of $\mathbf{X}$ are the P dominant eigenvectors of $\mathbf{C}$.

Let $\mathbf{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times 3}$ be a 3D set of points





How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \ldots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

- The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3\times 3}$.

2. Find $\boldsymbol{v}_2 \perp \boldsymbol{v}_1$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_2)$ is largest. ▸ Second dominant eigenvector of $\mathbf{C}$.

3. Find $\boldsymbol{v}_3 \perp [\boldsymbol{v}_1, \boldsymbol{v}_2]$ … *etc.*

- The **Principal Axes** of $\mathbf{X}$ are the P dominant eigenvectors of $\mathbf{C}$.

Principal
Component
Anaysis

Principal Component Analysis

- Probabilistic / Generative interpretation:

$$\begin{cases} p_\theta(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n; \mathbf{0}_P, \mathbf{\Lambda} - \sigma^2 \mathbf{I}_P), & \mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \dots, \lambda_P), \quad \sigma^2 \leq \lambda_P \\ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n; \mathbf{V}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_D), & \mathbf{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_P] \in \mathbb{R}^{D \times P}, \end{cases}$$

$$\theta = \{\mathbf{\Lambda}, \sigma^2, \mathbf{V}\}$$

Principal Component Analysis

- Probabilistic / Generative interpretation:

$$\begin{cases} p_\theta(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n; \mathbf{0}_P, \boldsymbol{\Lambda} - \sigma^2 \mathbf{I}_P), \quad \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \dots, \lambda_P), \quad \sigma^2 \leq \lambda_P \\ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n; \mathbf{V}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_D), \quad \mathbf{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_P] \in \mathbb{R}^{D \times P}, \end{cases}$$

$$\theta = \{\boldsymbol{\Lambda}, \sigma^2, \mathbf{V}\}$$

PCA is **equivalent** to:

- $\hat{\theta} = \underset{\theta}{\mathrm{argmax}} \log p_\theta(\mathbf{X})$ (Maximum Likelihood)
- $\hat{\boldsymbol{z}}_n = \underset{\boldsymbol{z}_n}{\mathrm{argmax}}\, p_{\hat{\theta}}(\boldsymbol{z}_n | \boldsymbol{x}_n)$ (Maximum a posteriori)

M.E. Tipping and C.M. Bishop. **"Probabilistic PCA."** *Journal of the Royal Statistical Society: Series B,* 61, no. 3 (1999): 611-622.

Principal Component Analysis

- Probabilistic / Generative interpretation:

$$\begin{cases} p_\theta(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n; \mathbf{0}_P, \boldsymbol{\Lambda} - \sigma^2\mathbf{I}_P), \quad \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \dots, \lambda_P), \quad \sigma^2 \leq \lambda_P \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n; \mathbf{V}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2\mathbf{I}_D), \quad \mathbf{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_P] \in \mathbb{R}^{D\times P}, \end{cases}$$

$$\theta = \{\boldsymbol{\Lambda}, \sigma^2, \mathbf{V}\}$$

PCA is **equivalent** to:

- $\hat{\theta} = \underset{\theta}{\mathrm{argmax}} \log p_\theta(\mathbf{X})$ (Maximum Likelihood)
- $\hat{\boldsymbol{z}}_n = \underset{\boldsymbol{z}_n}{\mathrm{argmax}}\, p_{\hat{\theta}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$ (Maximum a posteriori)

M.E. Tipping and C.M. Bishop. **"Probabilistic PCA."** *Journal of the Royal Statistical Society: Series B,* 61, no. 3 (1999): 611-622.

Variational Autoencoders

Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *ICLR* 2014.

Principal Component Analysis

- Probabilistic / Generative interpretation:

$$\begin{cases} p_\theta(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n; \mathbf{0}_P, \boldsymbol{\Lambda} - \sigma^2\mathbf{I}_P), & \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1,\dots,\lambda_P), \quad \sigma^2 \leq \lambda_P \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n; \boxed{\mathbf{V}\boldsymbol{z}_n + \boldsymbol{\mu}}, \sigma^2\mathbf{I}_D), & \mathbf{V} = [\boldsymbol{v}_1,\dots,\boldsymbol{v}_P] \in \mathbb{R}^{D\times P}, \end{cases}$$

$$\theta = \{\boldsymbol{\Lambda}, \sigma^2, \mathbf{V}\}$$

PCA is **equivalent** to:

- $\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \log p_\theta(\mathbf{X})$ (Maximum Likelihood)
- $\hat{\boldsymbol{z}}_n = \underset{\boldsymbol{z}_n}{\operatorname{argmax}}\, p_{\hat{\theta}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$ (Maximum a posteriori)

M.E. Tipping and C.M. Bishop. **"Probabilistic PCA."** *Journal of the Royal Statistical Society: Series B,* 61, no. 3 (1999): 611-622.

Variational Autoencoders

- Generalize PCA by replacing this by a **DNN** (the *decoder*)

Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *ICLR* 2014.

Principal Component Analysis

- Probabilistic / Generative interpretation:

$$\begin{cases} p_\theta(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n; \mathbf{0}_P, \boldsymbol{\Lambda} - \sigma^2 \mathbf{I}_P), & \boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_P), \quad \sigma^2 \leq \lambda_P \\ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n; \boxed{\mathbf{V}\boldsymbol{z}_n + \boldsymbol{\mu}}, \sigma^2 \mathbf{I}_D), & \mathbf{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_P] \in \mathbb{R}^{D \times P}, \end{cases}$$

$$\theta = \{\boldsymbol{\Lambda}, \sigma^2, \mathbf{V}\}$$

PCA is **equivalent** to:

- $\hat{\theta} = \operatorname*{argmax}_{\theta} \log p_\theta(\mathbf{X})$ (Maximum Likelihood)
- $\hat{\boldsymbol{z}}_n = \operatorname*{argmax}_{\boldsymbol{z}_n} p_{\hat{\theta}}(\boldsymbol{z}_n | \boldsymbol{x}_n)$ (Maximum a posteriori)

M.E. Tipping and C.M. Bishop. **"Probabilistic PCA."** *Journal of the Royal Statistical Society: Series B,* 61, no. 3 (1999): 611-622.

Variational Autoencoders

- Generalize PCA by replacing $\boxed{\text{this}}$ by a **DNN** (the *decoder*)
- Optimized using a variational approximation of $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ by another neural network (the *encoder*)

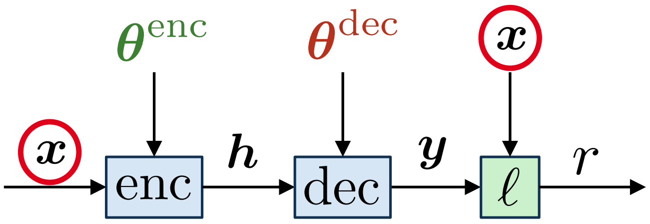
Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *ICLR* 2014.

Autoencoder

- A neural network trained to predict its input: a **pretext task**

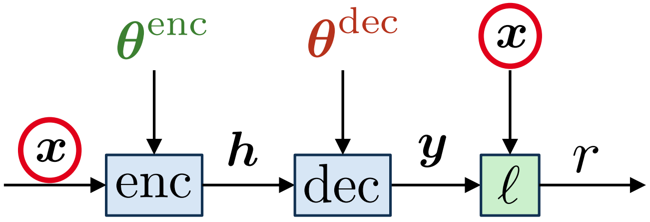
Autoencoder

- A neural network trained to predict its input: a **pretext task**
- Consists of two parts:
  - An **encoder** function $\boldsymbol{h} = \mathrm{enc}(\boldsymbol{x})$
  - A **decoder** function $\boldsymbol{y} = \mathrm{dec}(\boldsymbol{h})$



Autoencoder

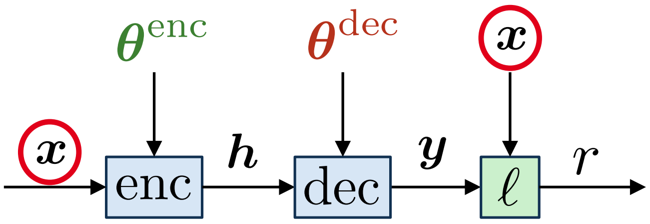
- A neural network trained to predict its input: a **pretext task**
- Consists of two parts:
  - An **encoder** function $\boldsymbol{h} = \mathrm{enc}(\boldsymbol{x})$
  - A **decoder** function $\boldsymbol{y} = \mathrm{dec}(\boldsymbol{h})$



- The task is non-trivial if the encoder is **dimensionality-reducing**

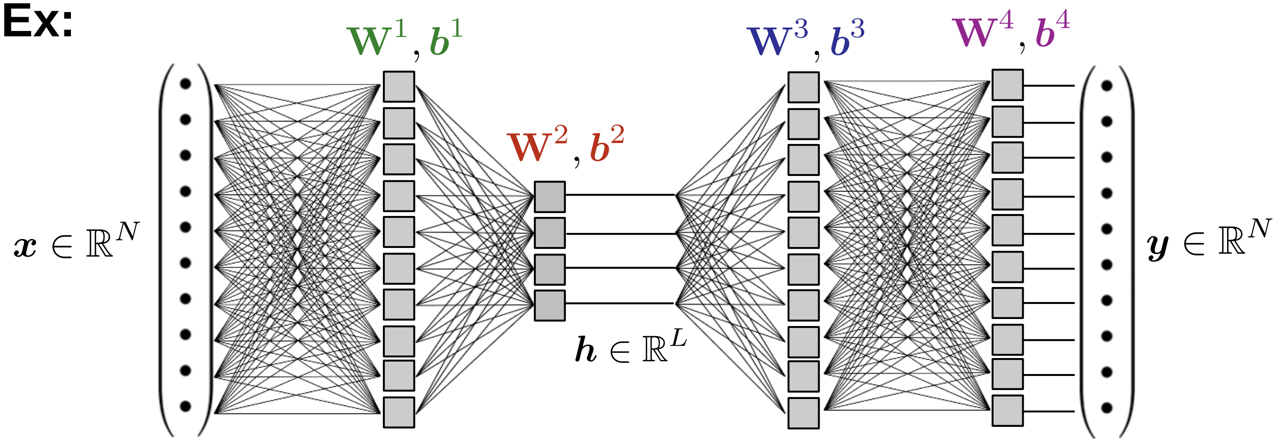
Autoencoder

- A neural network trained to predict its input: a **pretext task**
- Consists of two parts:
  - An **encoder** function $\boldsymbol{h} = \mathrm{enc}(\boldsymbol{x})$
  - A **decoder** function $\boldsymbol{y} = \mathrm{dec}(\boldsymbol{h})$



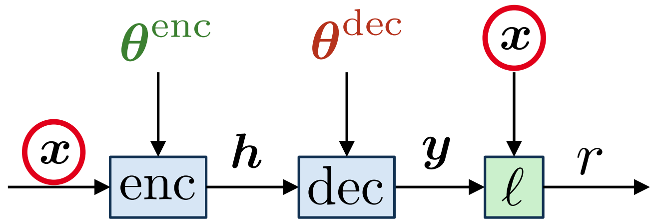
- The task is non-trivial if the encoder is **dimensionality-reducing**

Ex:



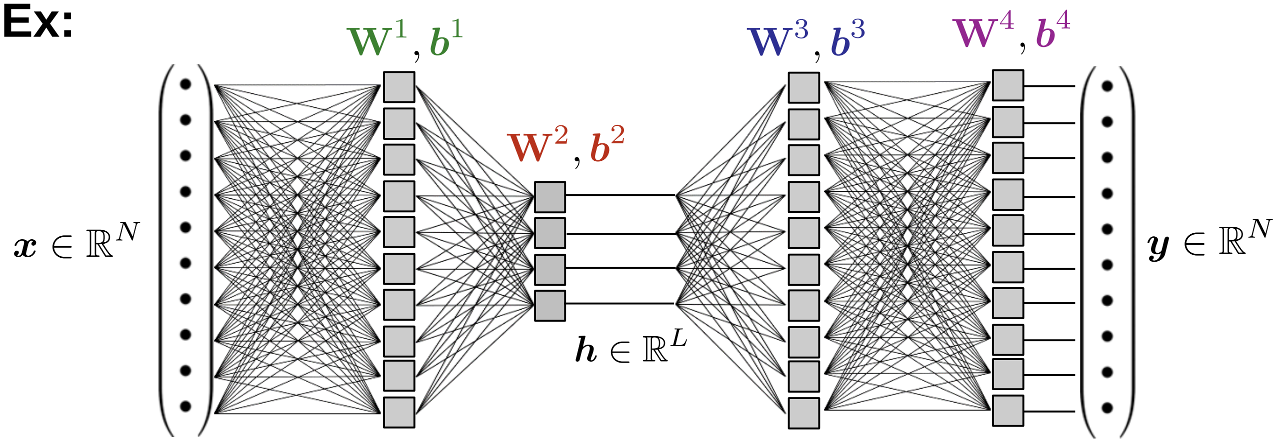
Autoencoder

- A neural network trained to predict its input: a **pretext task**
- Consists of two parts:
  - An **encoder** function $\boldsymbol{h} = \mathrm{enc}(\boldsymbol{x})$
  - A **decoder** function $\boldsymbol{y} = \mathrm{dec}(\boldsymbol{h})$



- The task is non-trivial if the encoder is **dimensionality-reducing**

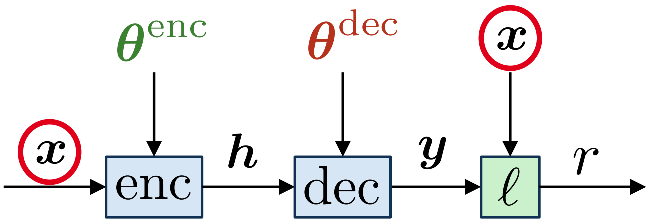
Ex:



- $\boldsymbol{h}$ is called an ***embedding*** of $\boldsymbol{x}$, i.e., a nonlinear representation of the input.

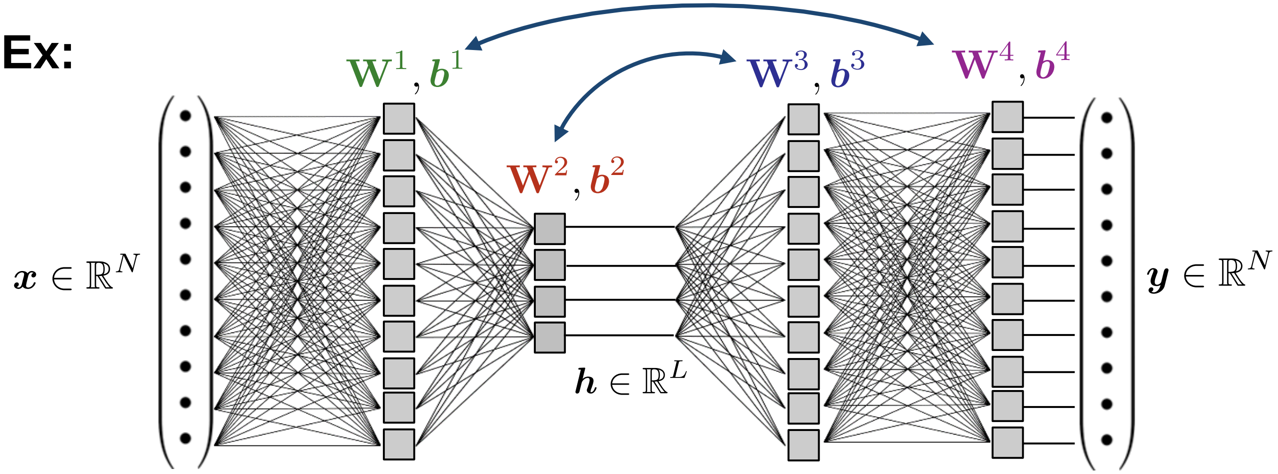
Autoencoder

- A neural network trained to predict its input: a **pretext task**
- Consists of two parts:
  - An **encoder** function $\boldsymbol{h} = \mathrm{enc}(\boldsymbol{x})$
  - A **decoder** function $\boldsymbol{y} = \mathrm{dec}(\boldsymbol{h})$



- The task is non-trivial if the encoder is **dimensionality-reducing**

Ex:

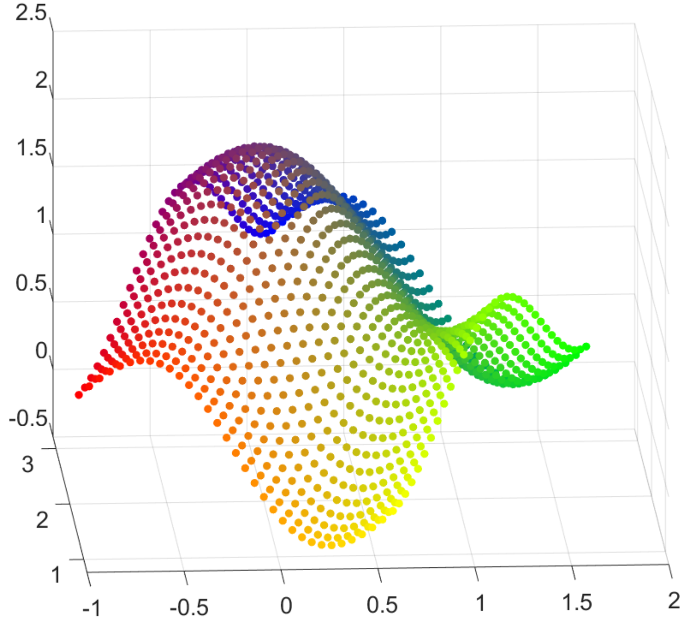


The encoder and decoder parameters can be ***tied*** together

- $\boldsymbol{h}$ is called an ***embedding*** of $\boldsymbol{x}$, i.e., a nonlinear representation of the input.

Manifold Learning

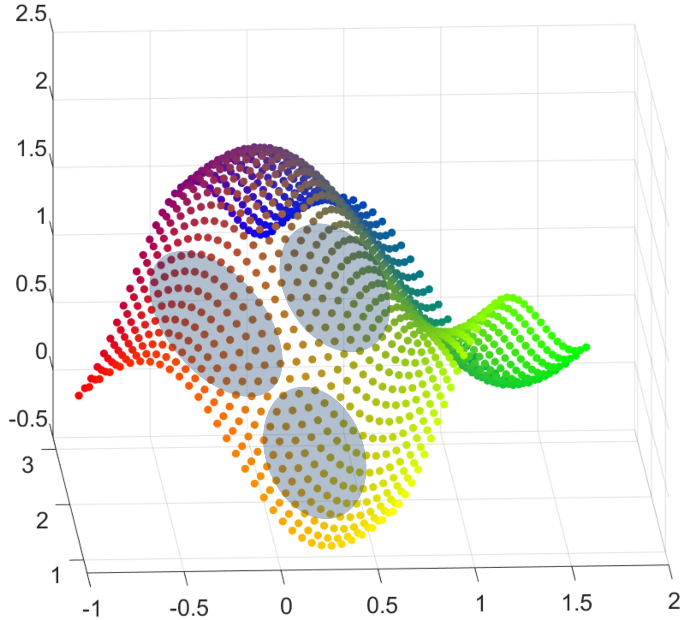
- Local Tangent Space Alignment (LTSA)



Zhang, Zhenyue, and Hongyuan Zha. **"Principal manifolds and nonlinear dimensionality reduction via tangent space alignment."** *SIAM journal on scientific computing* 26, no. 1 (2004): 313-338.

Manifold Learning

- Local Tangent Space Alignment (LTSA)

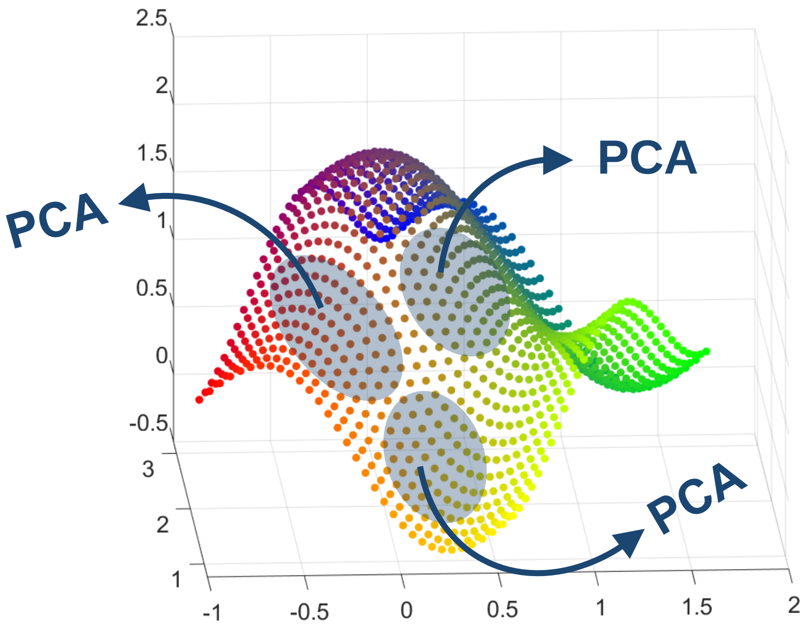


1. Builds local k-nearest neighborhoods on the data

Zhang, Zhenyue, and Hongyuan Zha. **"Principal manifolds and nonlinear dimensionality reduction via tangent space alignment."** *SIAM journal on scientific computing* 26, no. 1 (2004): 313-338.

Manifold Learning

- Local Tangent Space Alignment (LTSA)



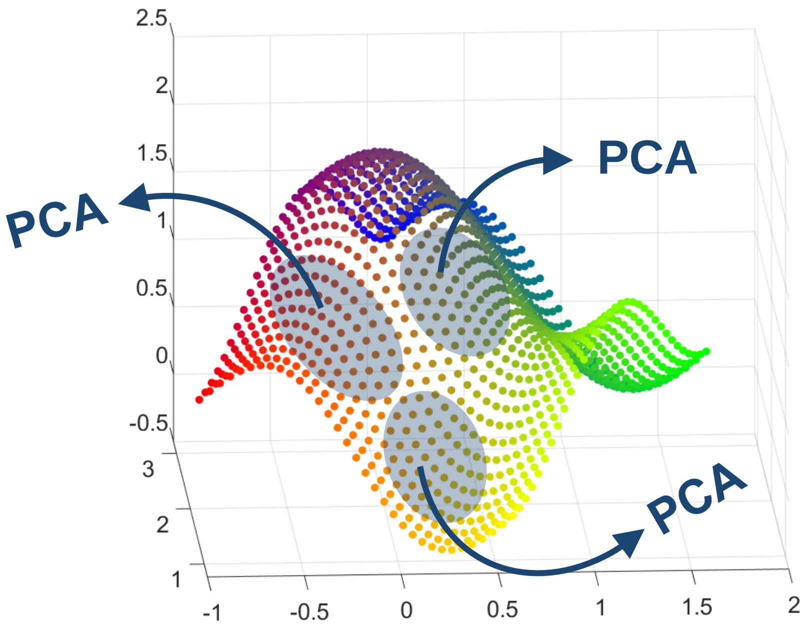
1. Builds local k-nearest neighborhoods on the data

2. Applies PCA to each neighborhood

Zhang, Zhenyue, and Hongyuan Zha. **"Principal manifolds and nonlinear dimensionality reduction via tangent space alignment."** *SIAM journal on scientific computing* 26, no. 1 (2004): 313-338.

Manifold Learning

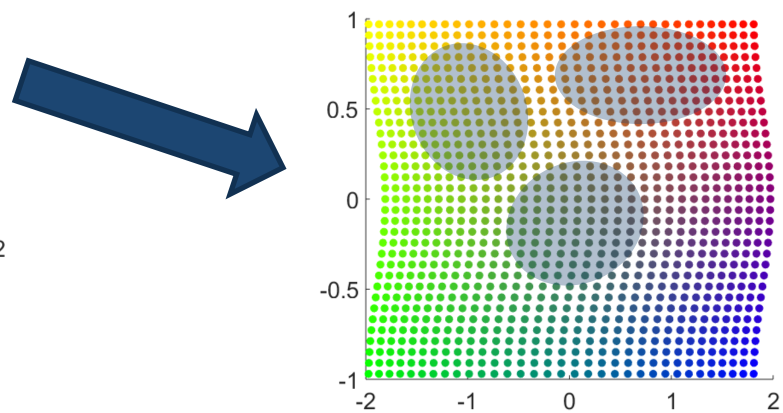
- Local Tangent Space Alignment (LTSA)



1. Builds local k-nearest neighborhoods on the data

2. Applies PCA to each neighborhood

3. Patch the local PCAs together



Zhang, Zhenyue, and Hongyuan Zha. **"Principal manifolds and nonlinear dimensionality reduction via tangent space alignment."** *SIAM journal on scientific computing* 26, no. 1 (2004): 313-338.

Manifold Learning

- Graph-Based Methods: *Isomap, LLE, Laplacian Eigenmap,…*

Ghodsi, Ali. **"Dimensionality reduction a short tutorial."** *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, no. 38 (2006): 2006.

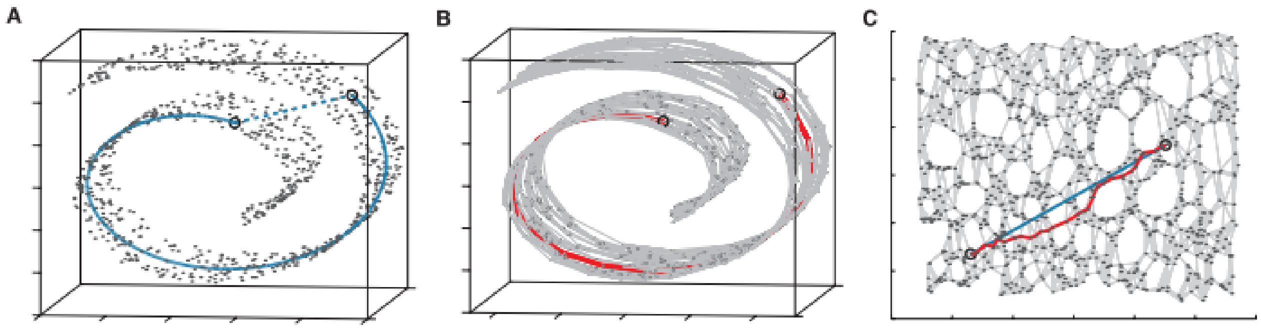
Manifold Learning

- Graph-Based Methods: *Isomap, LLE, Laplacian Eigenmap,…*
  - Build a **neighborhood graph** from the data
  - « *Unroll* » the graph to a lower dimensional space

Ghodsi, Ali. **"Dimensionality reduction a short tutorial."** *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, no. 38 (2006): 2006.

Manifold Learning

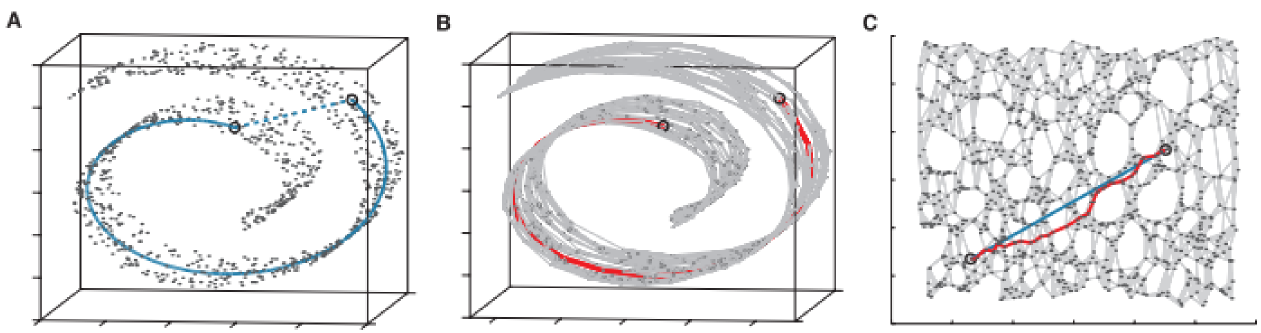
- Graph-Based Methods: *Isomap, LLE, Laplacian Eigenmap,…*
  - Build a **neighborhood graph** from the data
  - « *Unroll* » the graph to a lower dimensional space
  - Ex: **Isomap**. Compute all ***geodesic distances*** (shortest paths) on the graph



Ghodsi, Ali. **"Dimensionality reduction a short tutorial."** *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, no. 38 (2006): 2006.

Manifold Learning

- Graph-Based Methods: *Isomap, LLE, Laplacian Eigenmap,…*
  - Build a **neighborhood graph** from the data
  - « *Unroll* » the graph to a lower dimensional space
  - Ex: **Isomap**. Compute all ***geodesic distances*** (shortest paths) on the graph



- ***Stochastic neighbor embedding*** (**SNE**): match *neighborhood probabilities* in the high- and low-dim. spaces

Hinton, Geoffrey E., and Sam Roweis. "Stochastic neighbor embedding." *Advances in neural information processing systems* 15 (2002).

Ghodsi, Ali. **"Dimensionality reduction a short tutorial."** *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, no. 38 (2006): 2006.

Manifold Learning

- Ex: **t-SNE**: uses a Gaussian model for similarity between data points and a *Student's t* (Cauchy) model for similarity in the latent space (2D or 3D)

Manifold Learning

- Ex: **t-SNE**: uses a Gaussian model for similarity between data points and a *Student's t* (Cauchy) model for similarity in the latent space (2D or 3D)
- The variances of the Gaussians are fixed to attain a given ***entropy*** value.

Manifold Learning

- Ex: **t-SNE**: uses a Gaussian model for similarity between data points and a *Student's t* (Cauchy) model for similarity in the latent space (2D or 3D)
- The variances of the Gaussians are fixed to attain a given ***entropy*** value.
- The latent representation is found by minimizing the ***Kullback-Leibler divergence*** wrt. these distributions using **gradient descent**.

Manifold Learning

- Ex: **t-SNE**: uses a Gaussian model for similarity between data points and a *Student's t* (Cauchy) model for similarity in the latent space (2D or 3D)
- The variances of the Gaussians are fixed to attain a given ***entropy*** value.
- The latent representation is found by minimizing the ***Kullback-Leibler divergence*** wrt. these distributions using **gradient descent**.
- Easy to use thanks to the Scikit Learn implementation

sklearn.manifold.TSNE

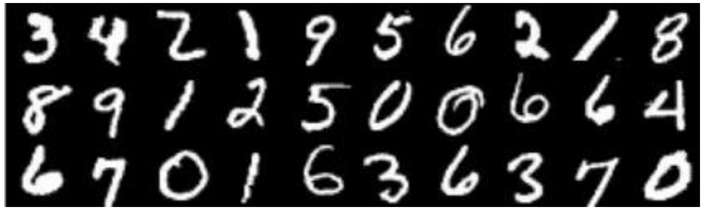
class `sklearn.manifold.TSNE`(*n_components=2, *, perplexity=30.0, early_exaggeration=12.0, learning_rate='auto', n_iter=1000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', metric_params=None, init='pca', verbose=0, random_state=None, method='barnes_hut', angle=0.5, n_jobs=None, square_distances='deprecated'*) [source]

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

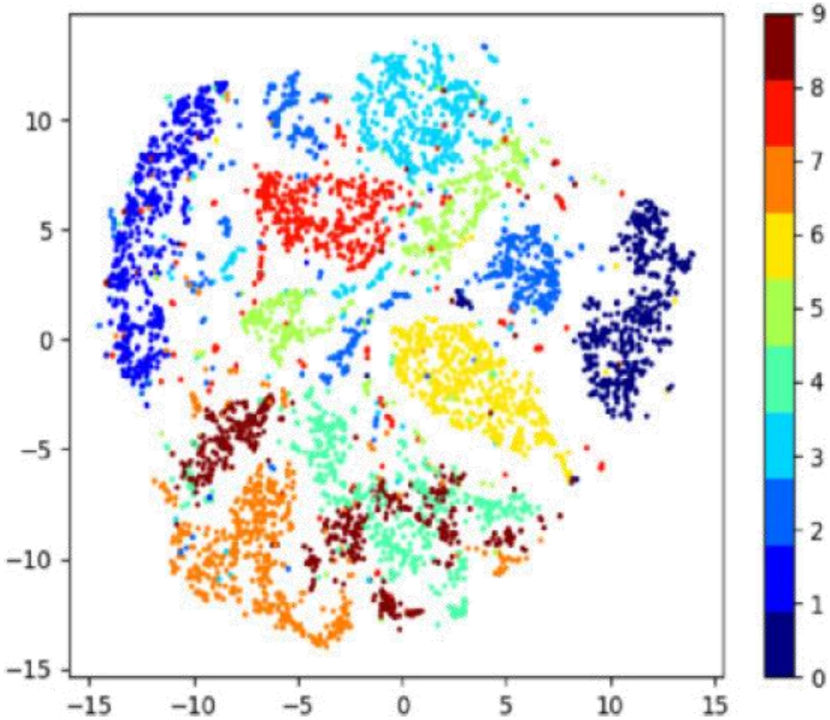
Manifold Learning

Examples of t-SNE visualizations

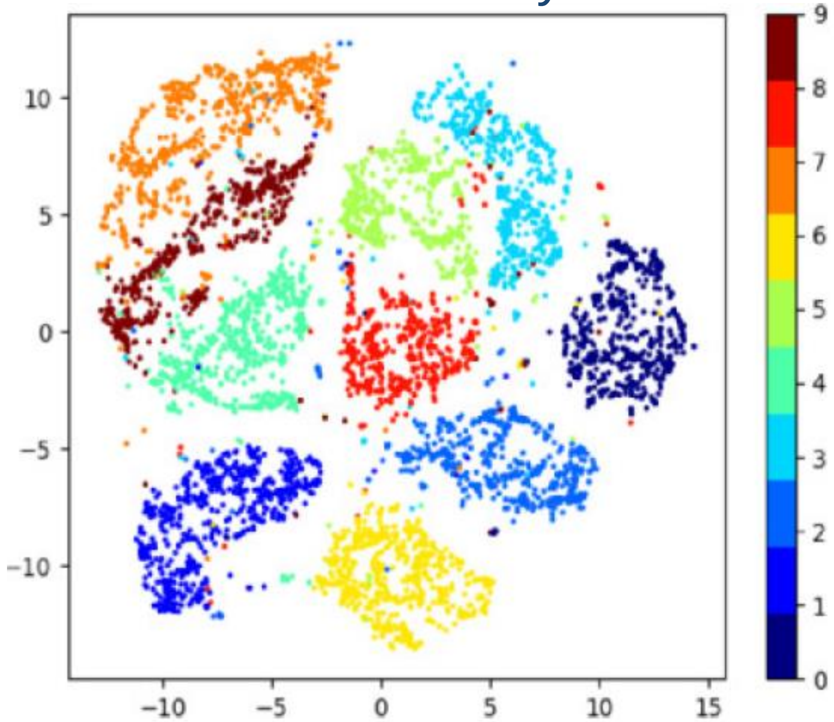
- MNIST dataset



T-SNE on image pixels



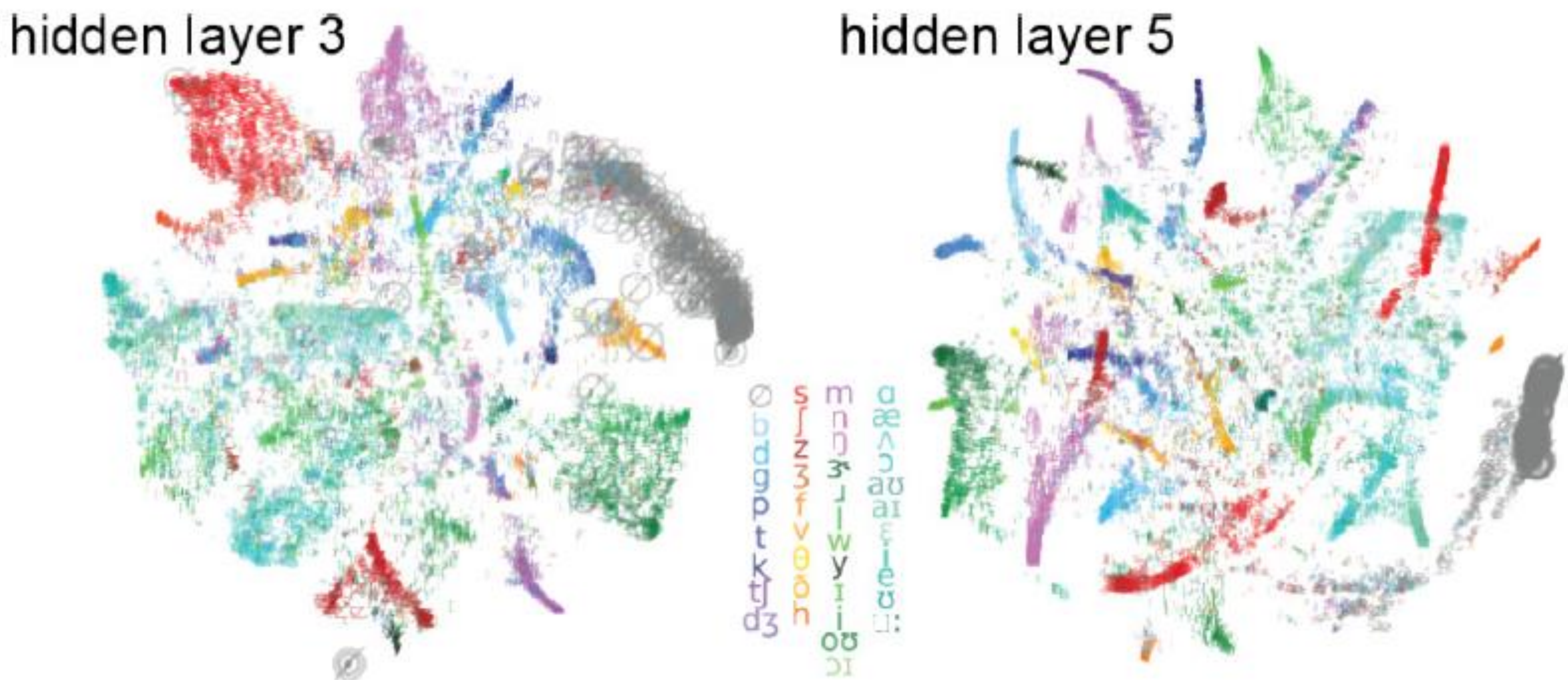
T-SNE on auto-encoder bottleneck layer



Manifold Learning

Examples of t-SNE visualizations

- Speech recognition (Wall Street Journal Dataset)



And in the Deep Learning era?

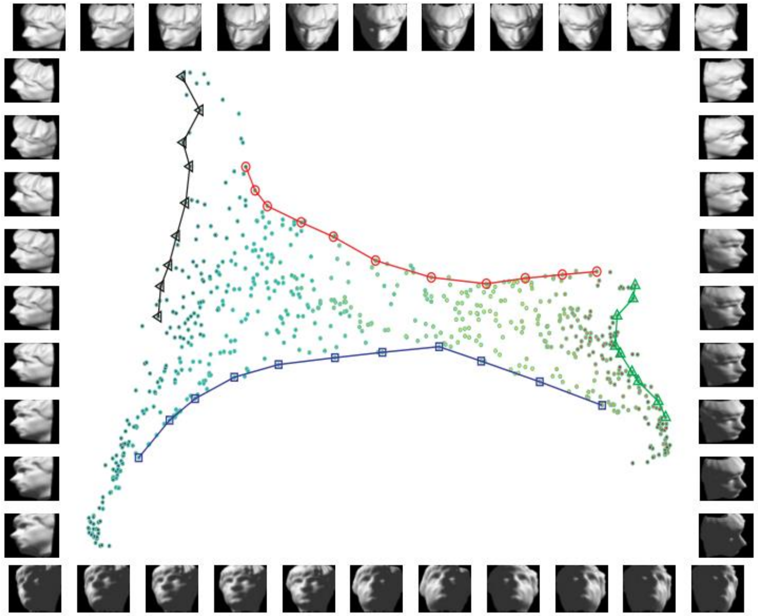
Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *arXiv:1312.6114* (2013).

And in the Deep Learning era?

Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *arXiv:1312.6114* (2013).

Typical applications:

Dataset Visualization



Data generation (Glow)

Pre-processing to speed up learning

Compression

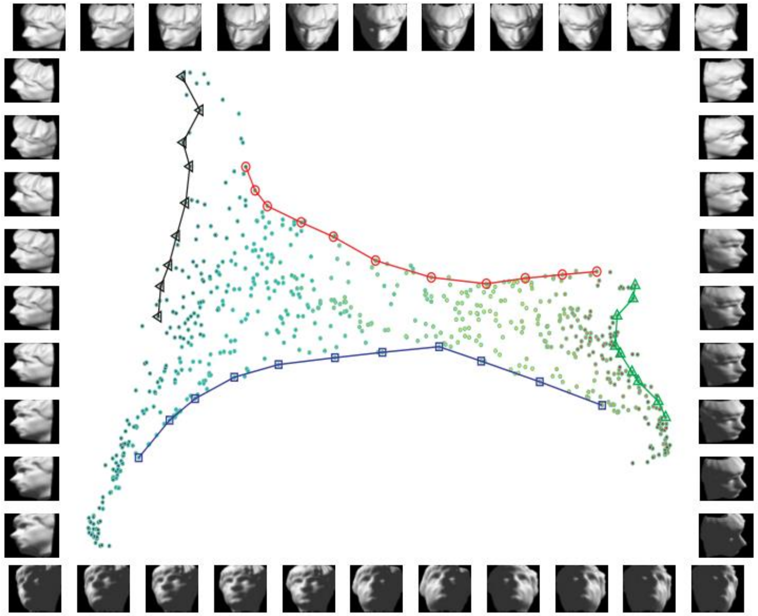
And in the Deep Learning era?

Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *arXiv:1312.6114* (2013).

Dinh, Laurent, David Krueger, and Yoshua Bengio. **"Nice: Non-linear independent components estimation."** *arXiv preprint arXiv:1410.8516* (2014). ***=> Invertible Neural Networks***

Typical applications:

Dataset Visualization



Data generation (Glow)

Pre-processing to speed up learning

Compression

And in the Deep Learning era?

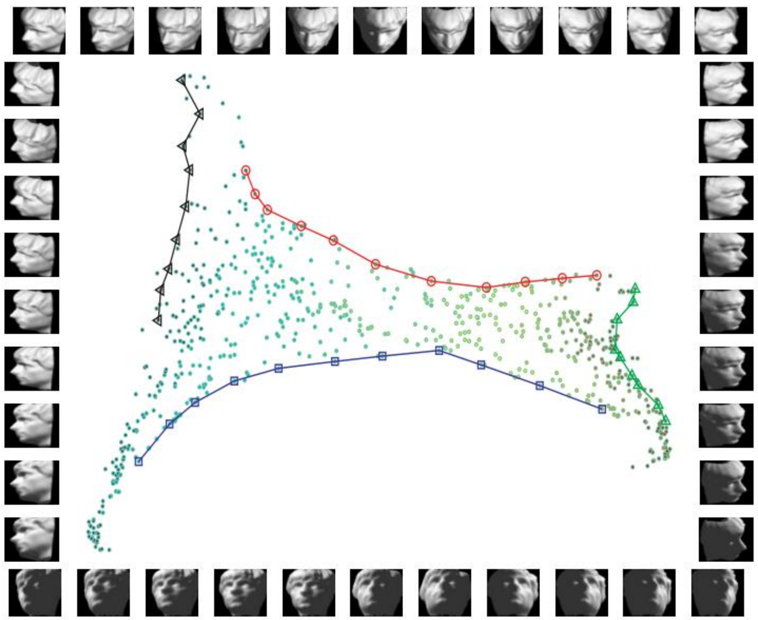
Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *arXiv:1312.6114* (2013).

Dinh, Laurent, David Krueger, and Yoshua Bengio. **"Nice: Non-linear independent components estimation."** *arXiv preprint arXiv:1410.8516* (2014). ***=> Invertible Neural Networks***

+ extensions (Normalizing Flows, Glow…)

Typical applications:

Dataset Visualization



Data generation (Glow)

Pre-processing to speed up learning

Compression

Supervised Learning

Labeled training data

$$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$$

$[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ Input ⟷ $[\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_N]$ Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → **Estimate** $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- Classification

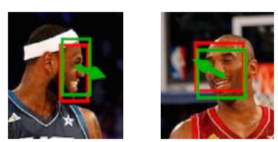
Ex. application: *dog breed*

$\boldsymbol{y} =$ 1

2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1 0.6 0.3

Unsupervised Learning

Unlabeled training data

$$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$$

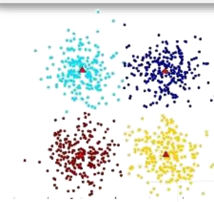
$[\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N]$ Input

Training

Learned Model f_θ → **Representation** $[\boldsymbol{z}_1, \boldsymbol{z}_2, \dots, \boldsymbol{z}_N]$

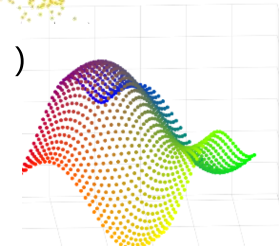
1. Discrete case:
- Clustering

$\boldsymbol{z} =$



2. Continuous case: ($\dim(\boldsymbol{z}) \ll \dim(\boldsymbol{x})$)
- Dimensionality Reduction

$\boldsymbol{z} =$



3. Sparse case:
- Dictionary Learning

$\boldsymbol{z} =$