10-701: Introduction to Machine Learning Lecture 17: Unsupervised Learning

Henry Chai & Zack Lipton

11/1/23

Front Matter

- Announcements
  - Project Proposals due 11/3 (Friday!)
    - Each group should only submit one PDF to Gradescope (see Piazza for instructions on making group submissions)
- Recommended Readings
  - Murphy, Chapters 12.2.1 - 12.2.3
  - Murphy, Chapters 25.5.1 - 25.5.2
  - Daumé III, Chapter 15: Unsupervised Learning

Learning Paradigms

- Supervised learning - $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}$
  - Regression - $y^{(n)} \in \mathbb{R}$
  - Classification - $y^{(n)} \in \{1, \ldots, C\}$
- Reinforcement learning - $\mathcal{D} = \left\{\left(\boldsymbol{s}^{(n)}, \boldsymbol{a}^{(n)}, r^{(n)}\right)\right\}_{n=1}^{N}$
- Unsupervised learning - $\mathcal{D} = \left\{\boldsymbol{x}^{(n)}\right\}_{n=1}^{N}$
  - Clustering
  - Dimensionality reduction

Unsupervised Learning

- Clustering: split an unlabeled data set into groups or partitions of “similar” data points
  - Use cases:
    - Organizing data
    - Discovering patterns or structure
    - Preprocessing for downstream tasks
- Dimensionality Reduction: given some unlabeled data set, learn a latent (typically lower-dimensional) representation
  - Use cases:
    - Decreasing computational costs
    - Improving generalization
    - Visualizing data

Recall: Similarity for kNN

- Intuition: ~~predict the label of a data point to be the label of the "most similar" training point~~ two points are "similar" if the distance between them is small
  - Euclidean distance: $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2$
- Partition-based clustering: Given a desired number of clusters, K, return a partition of the data set into K groups or clusters, $\{C_1, \dots, C_K\}$, that optimize some objective function

Recipe for K-means

- Define a model and model parameters
  - Assume K clusters, and use the Euclidean distance
  - Cluster centers: $\mu_1, \dots, \mu_K$ and assignments: $z^{(1)}, \dots, z^{(N)} \in \{1, 2, \dots, K\}$
- Write down an objective function

$$\min \sum_{n=1}^{N} \| x^{(n)} - \mu_{z^{(n)}} \|_2^2$$

- Optimize the objective w.r.t. the model parameters

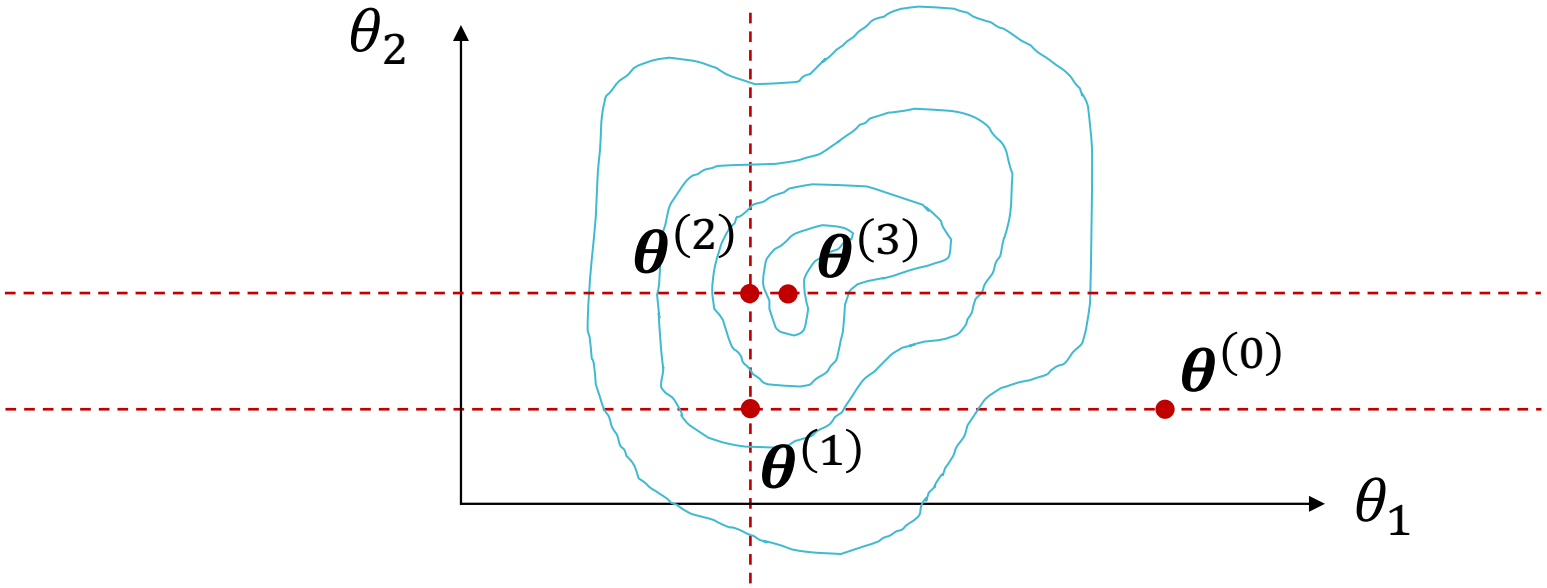
(Block) Coordinate descent

Coordinate Descent

- Goal: minimize some objective

$$\widehat{\boldsymbol{\theta}} = \text{argmin } J(\boldsymbol{\theta})$$

- Idea: iteratively pick one variable and minimize the objective w.r.t. just that variable, *keeping all others fixed*.



Block Coordinate Descent

- Goal: minimize some objective

$$\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}} = \operatorname{argmin} J(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

- Idea: iteratively pick one *block* of variables ($\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$) and minimize the objective w.r.t. that block, keeping the other(s) fixed.
  - Ideally, blocks should be the largest possible set of variables *that can be efficiently optimized simultaneously*

Optimizing the K-means objective

$$\hat{\boldsymbol{\mu}}_1, \ldots, \hat{\boldsymbol{\mu}}_K, z^{(1)}, \ldots, z^{(\mathrm{N})} = \operatorname{argmin} \sum_{n=1}^{N} \left\| \boldsymbol{x}^{(n)} - \boldsymbol{\mu}_{z^{(n)}} \right\|_2$$

- If $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ are fixed

$$\hat{z}^{(n)} = \underset{k \in \{1, \ldots, K\}}{\operatorname{argmin}} \| x^{(n)} - \mu_k \|_2$$

- If $z^{(1)}, \ldots, z^{(N)}$ are fixed

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{n : z^{(n)} = k} x^{(n)}$$

where N_k = # of data points assigned to cluster k

K-means Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}\right)\right\}_{n=1}^{N}, K$

1. Initialize cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$
2. While NOT CONVERGED
   a. Assign each data point to the cluster with the nearest cluster center:
   $$z^{(n)} = \underset{k}{\operatorname{argmin}} \left\|\boldsymbol{x}^{(n)} - \boldsymbol{\mu}_k\right\|_2$$
   b. Recompute the cluster centers:
   $$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n\,:z^{(n)}=k} \boldsymbol{x}^{(n)}$$
   where N_k is the number of data points in cluster k

- Output: cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and cluster assignments $z^{(1)}, \ldots, z^{(N)}$

K-means: Example ($K = 3$)

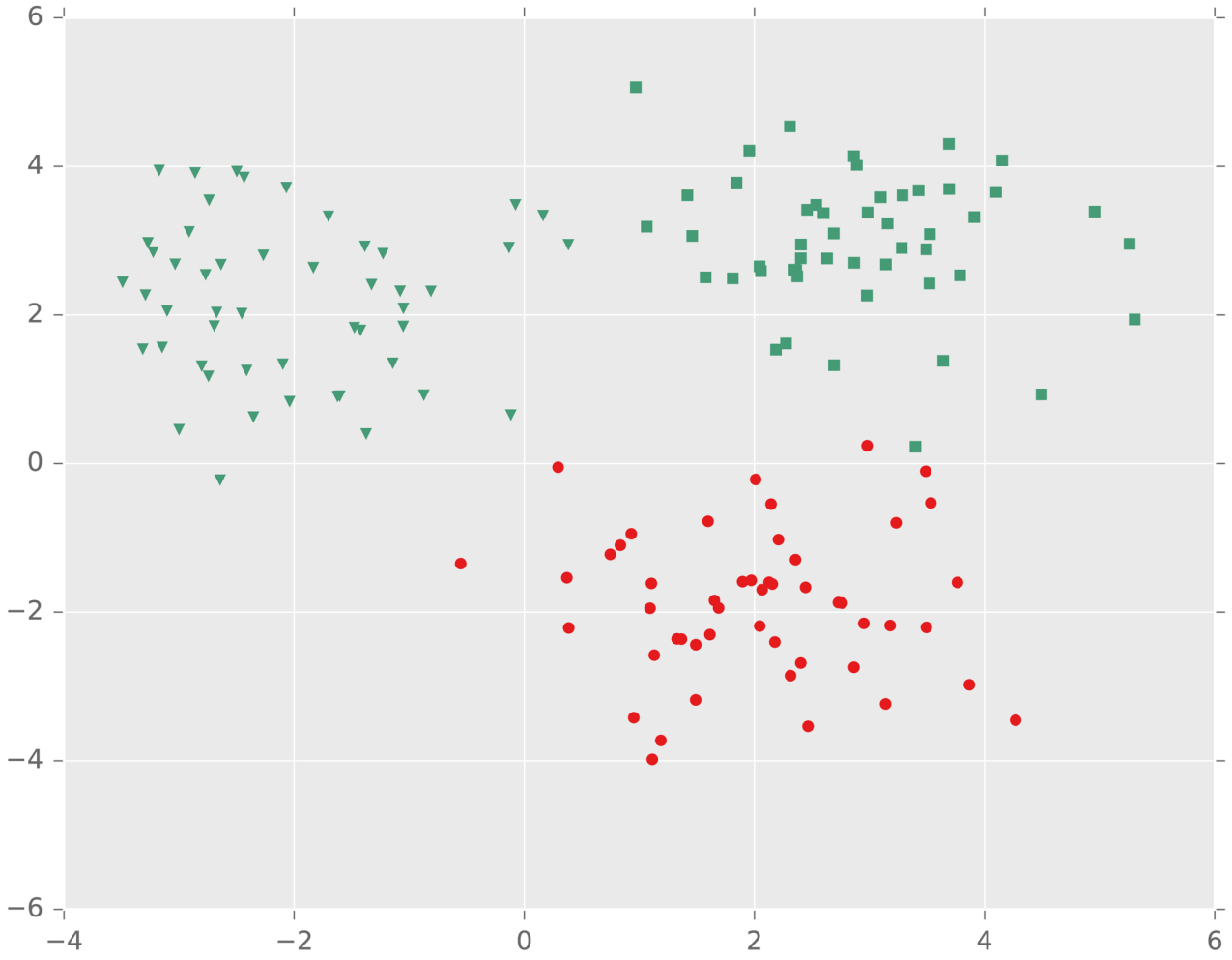


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

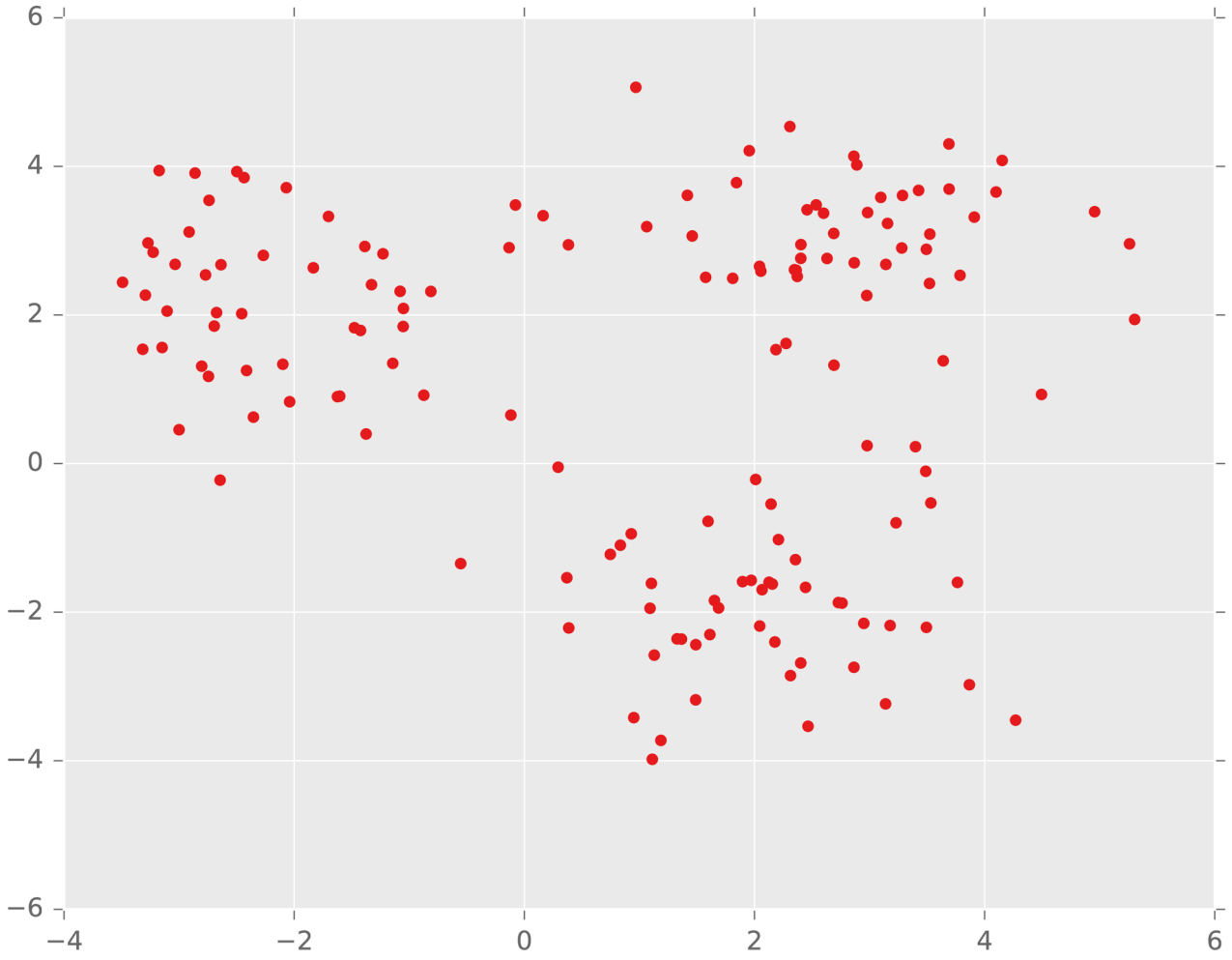


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

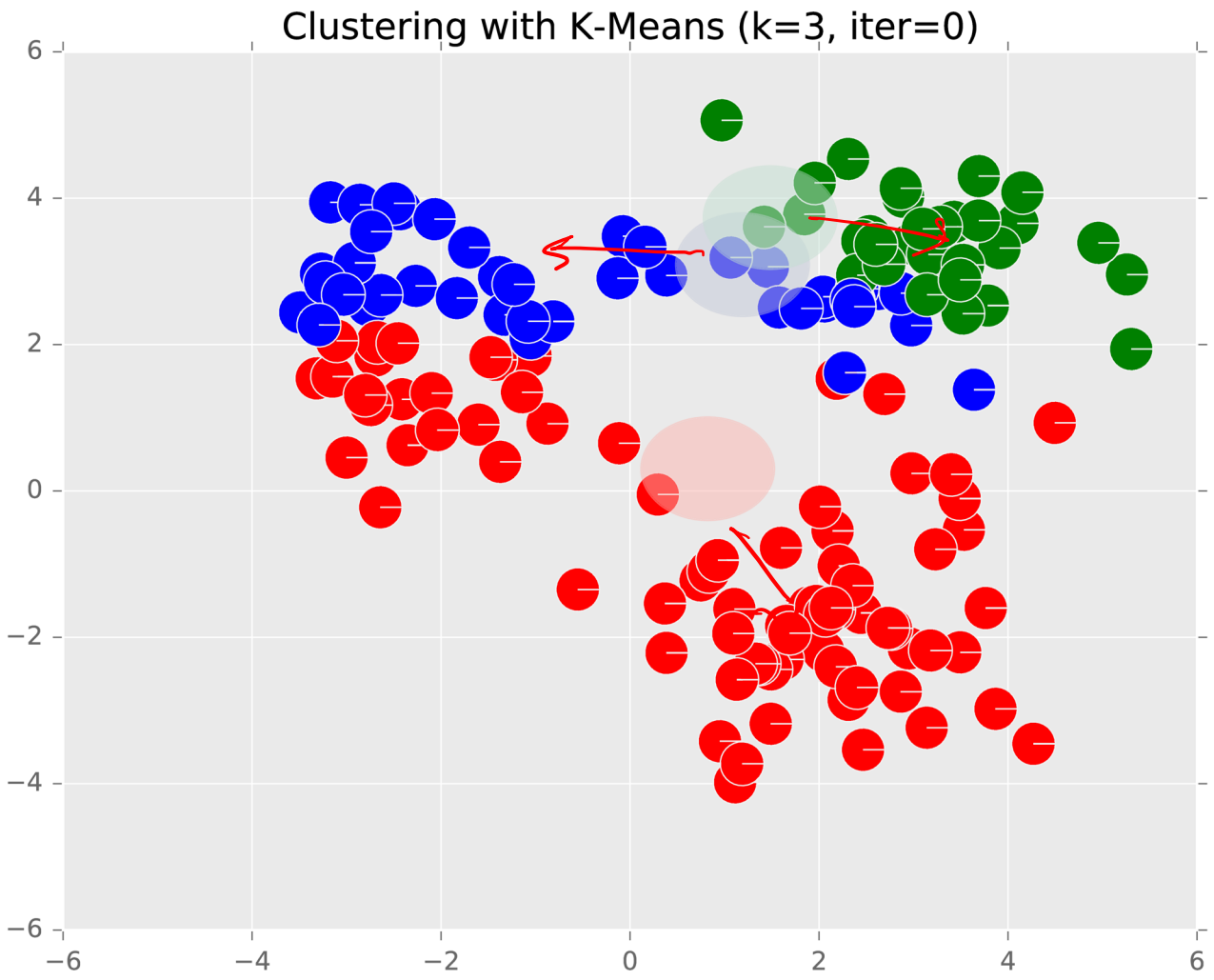


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

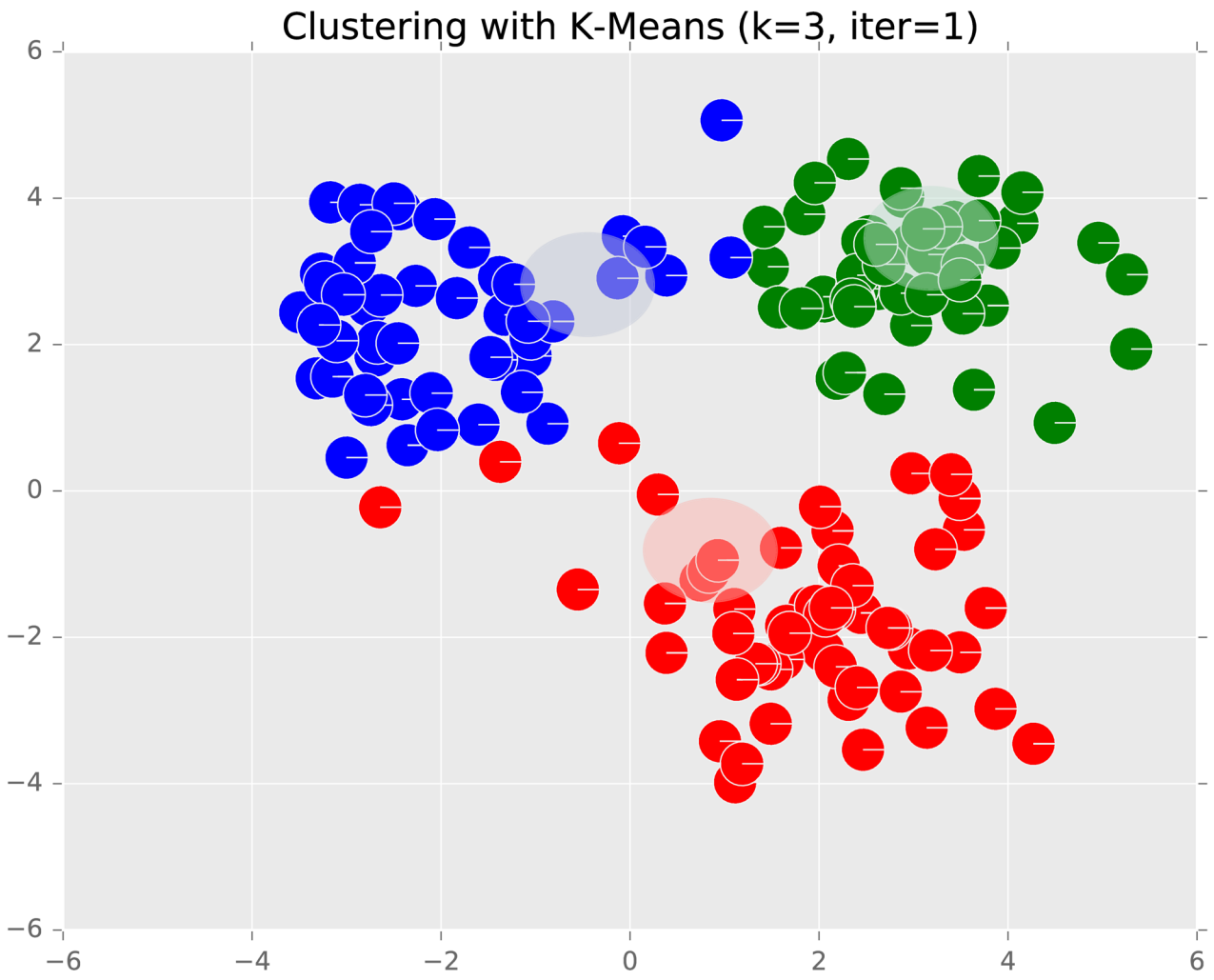


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

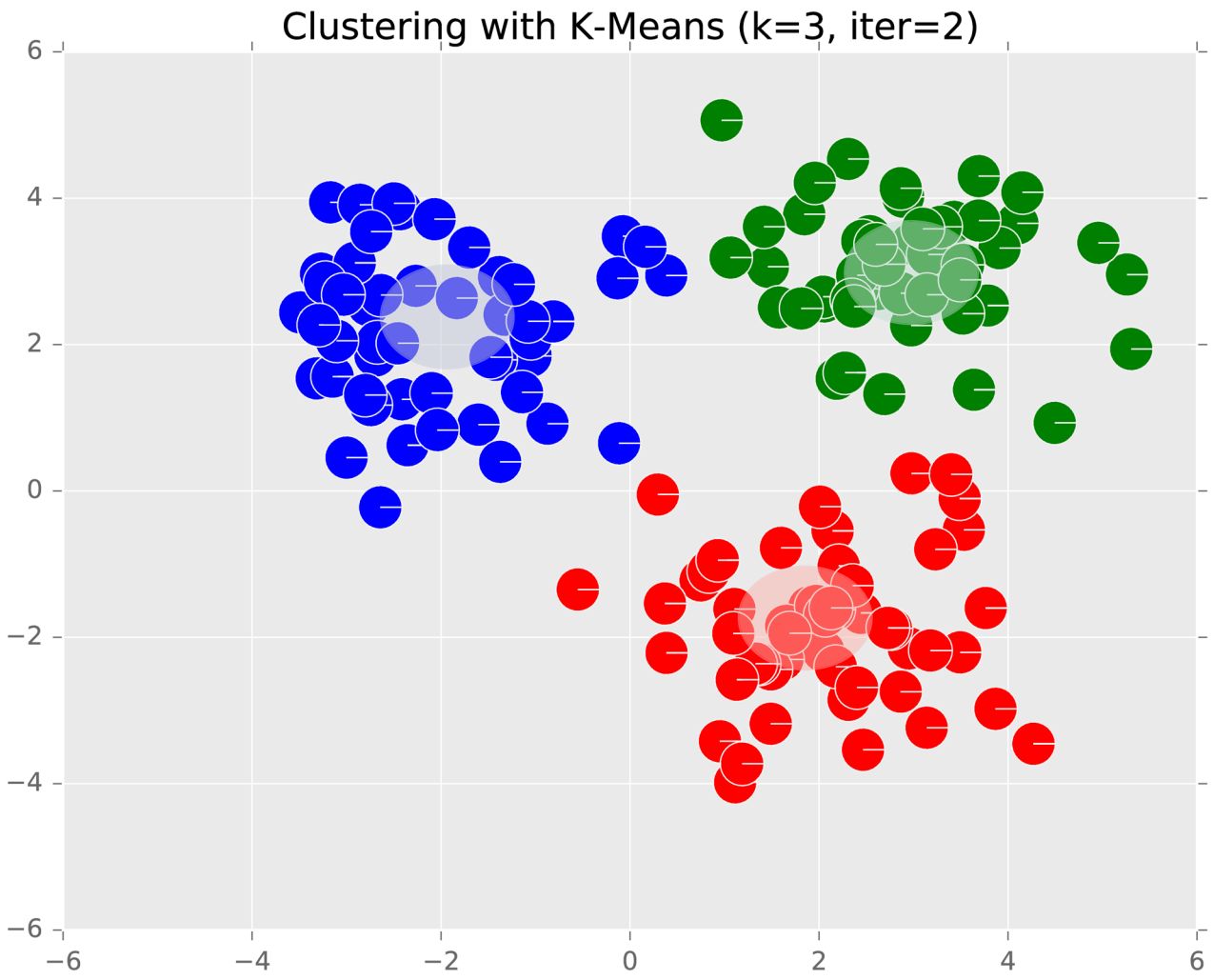


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

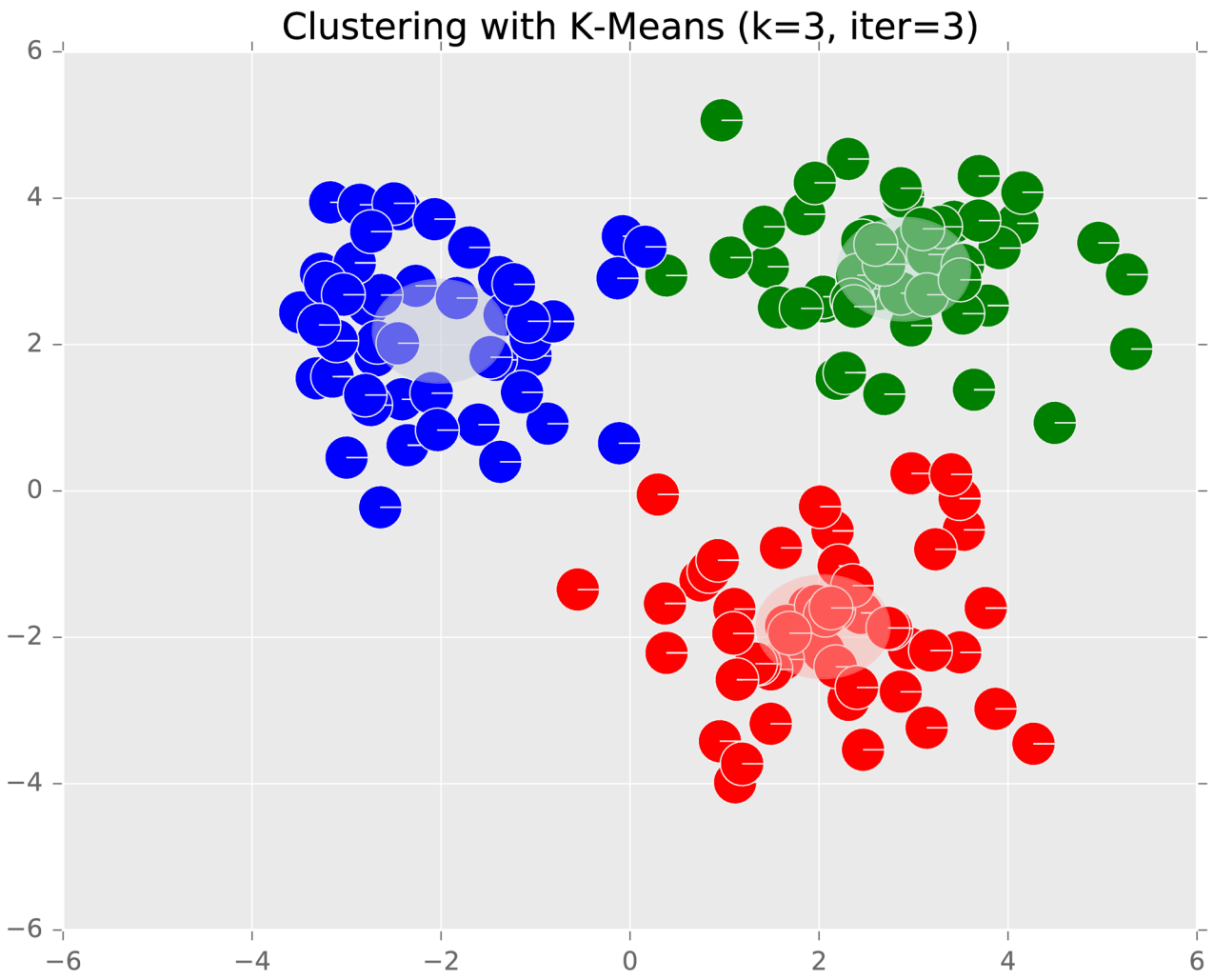


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

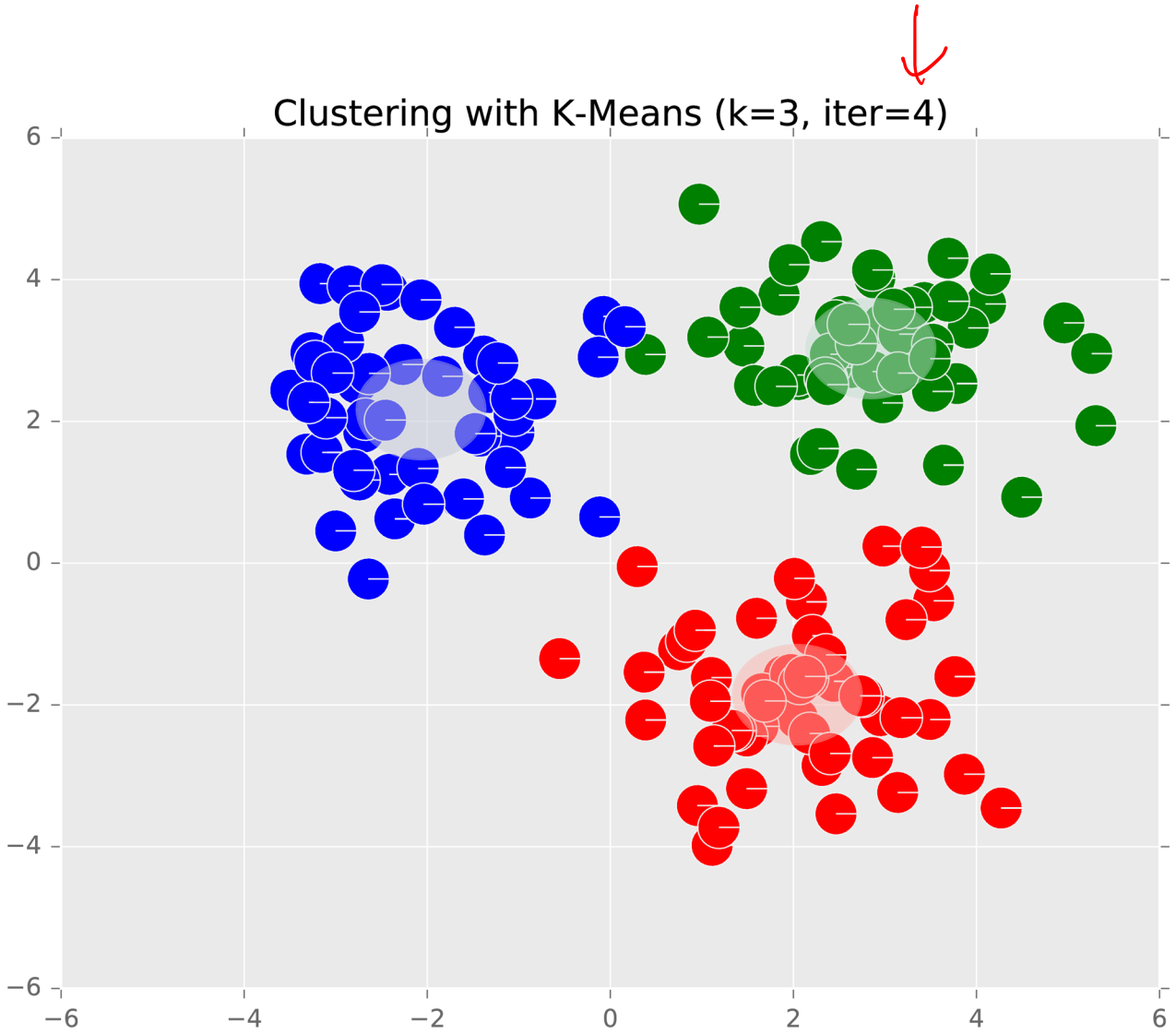


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

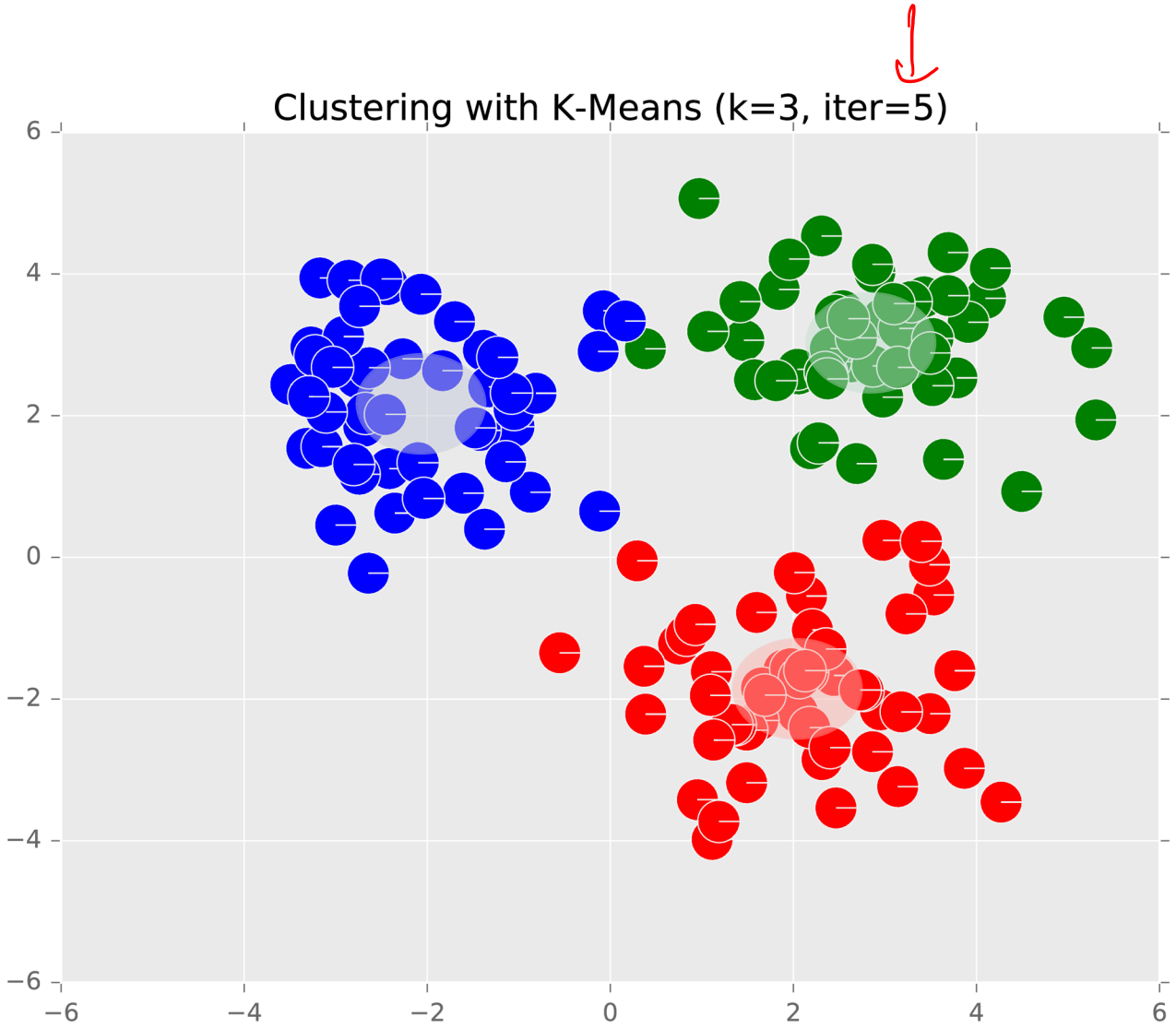


Figure courtesy of Matt Gormley

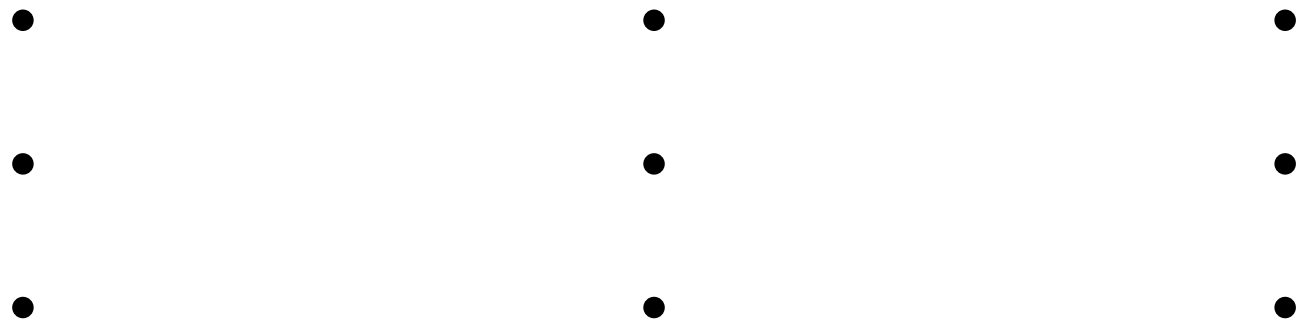
Setting K

- Idea: choose the value of K that minimizes the objective function

- look for largest dips – ("elbow")
- evaluate on downstream metric/task

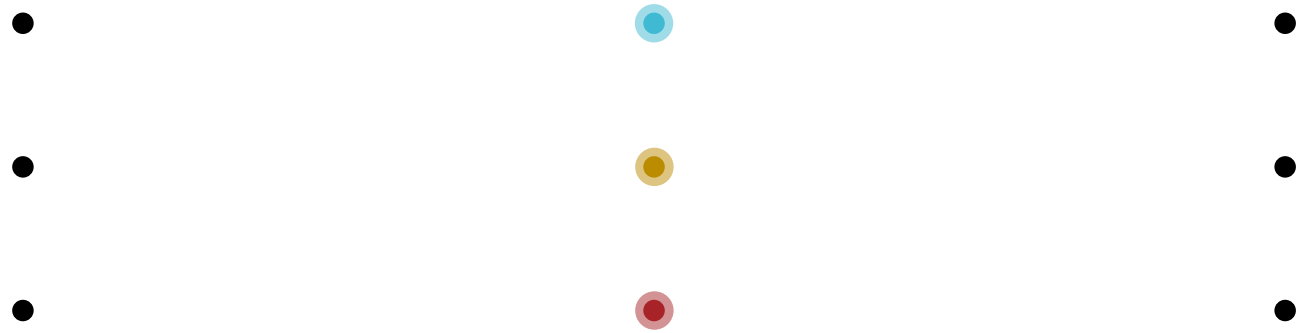
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



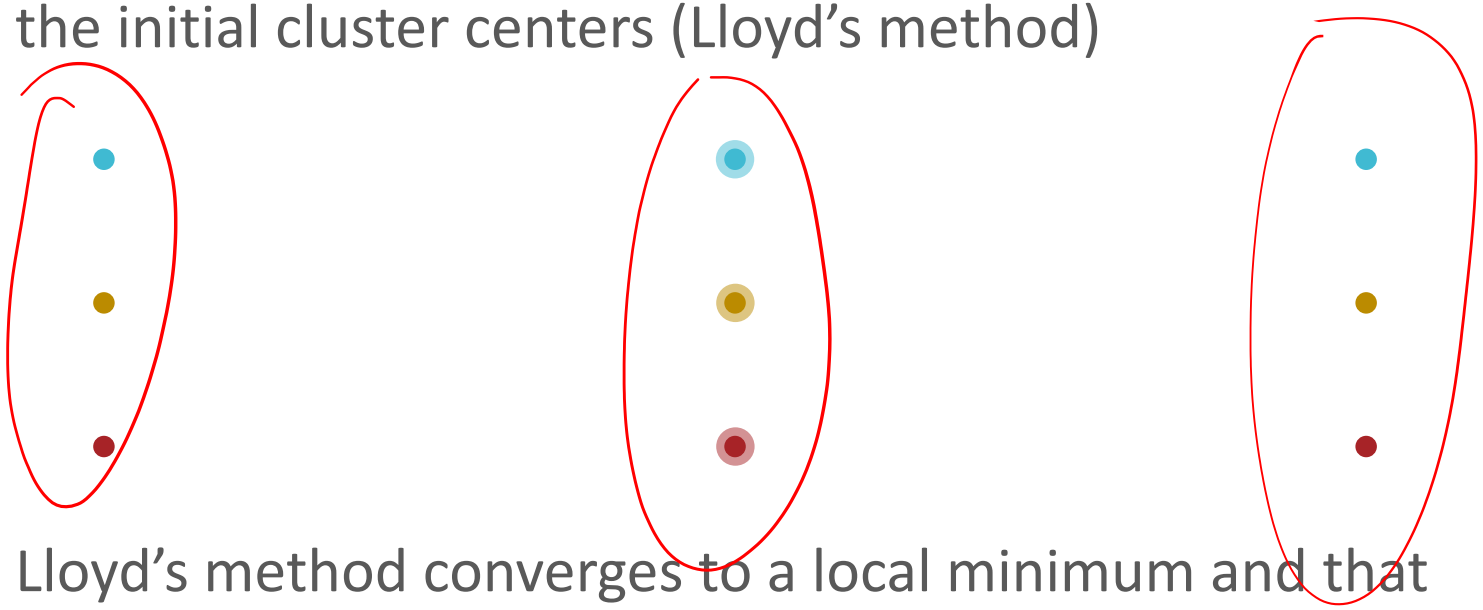
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



- Lloyd's method converges to a local minimum and that local minimum can be arbitrarily bad (relative to the optimal clusters)
- Intuition: want initial cluster centers to be far apart from one another

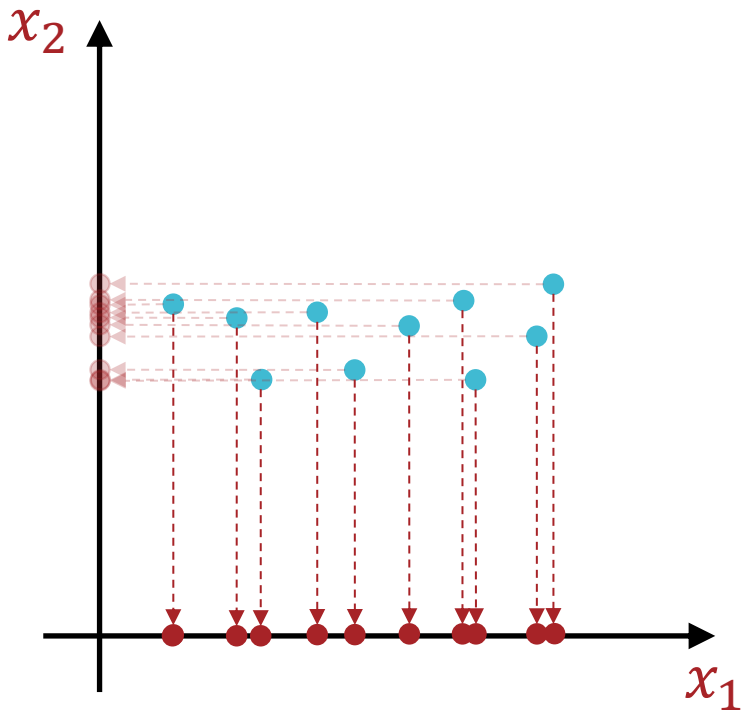
K-means++ (Arthur and Vassilvitskii, 2007)

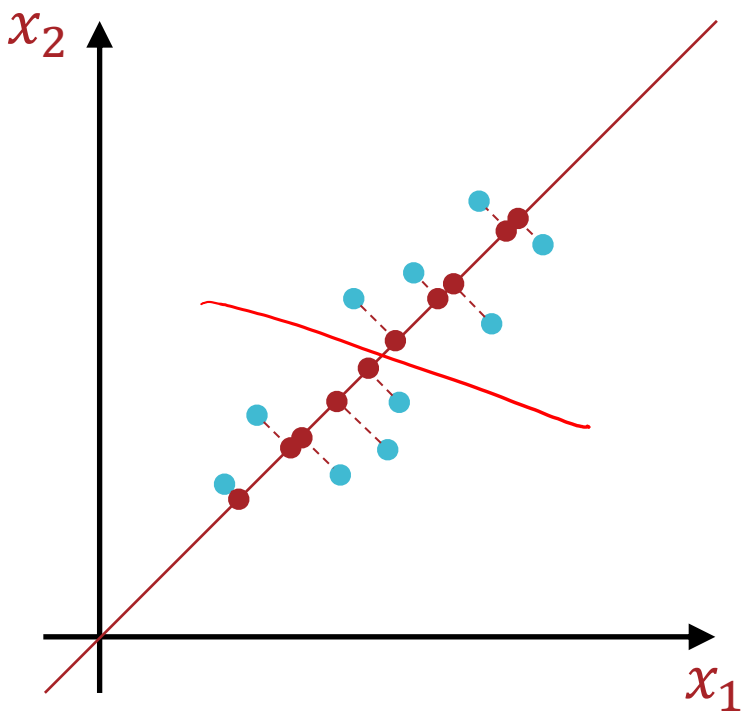
1. Choose the first cluster center randomly from the data points.
2. For each other data point $\boldsymbol{x}$, compute $D(\boldsymbol{x})$, the distance between $\boldsymbol{x}$ and the closest cluster center.
3. Select the next cluster center proportional to $D(\boldsymbol{x})^2$.
4. Repeat 2 and 3 $K-1$ times.

- K-means++ achieves a $O(\log K)$ approximation to the optimal clustering in expectation
- Both Lloyd's method and K-means++ can benefit from multiple random restarts.

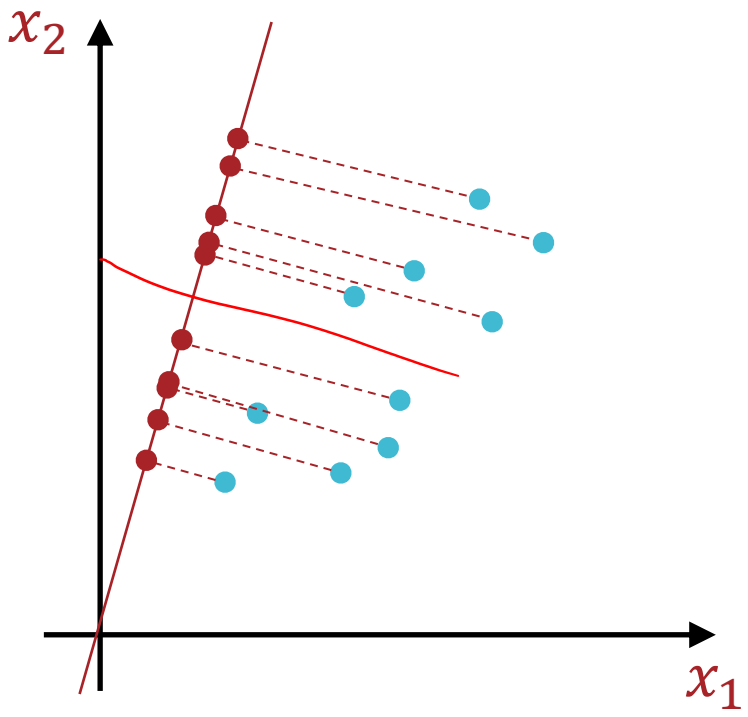
Unsupervised Learning

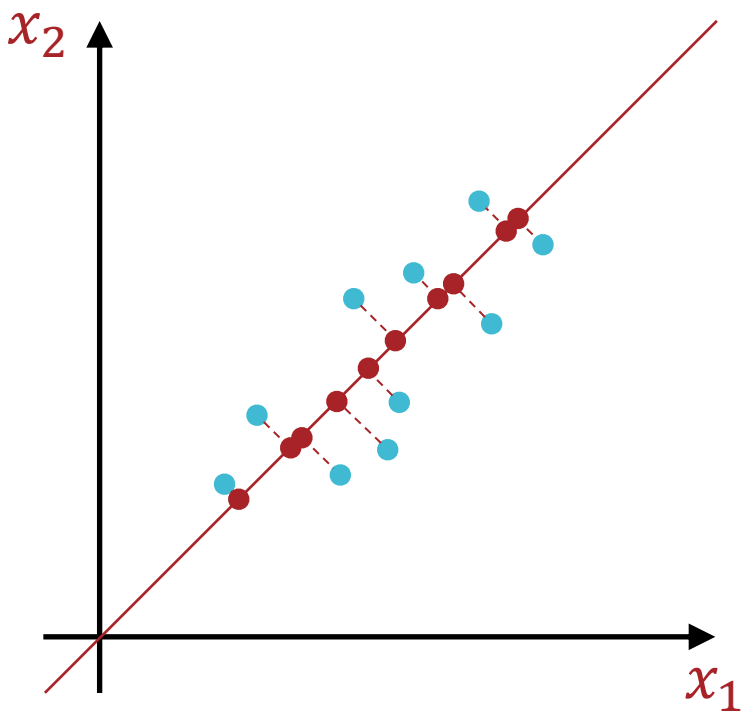
- Clustering: split an unlabeled data set into groups or partitions of “similar” data points
  - Use cases:
    - Organizing data
    - Discovering patterns or structure
    - Preprocessing for downstream tasks
- Dimensionality Reduction: given some unlabeled data set, learn a latent (typically lower-dimensional) representation
  - Use cases:
    - Decreasing computational costs
    - Improving generalization
    - Visualizing data



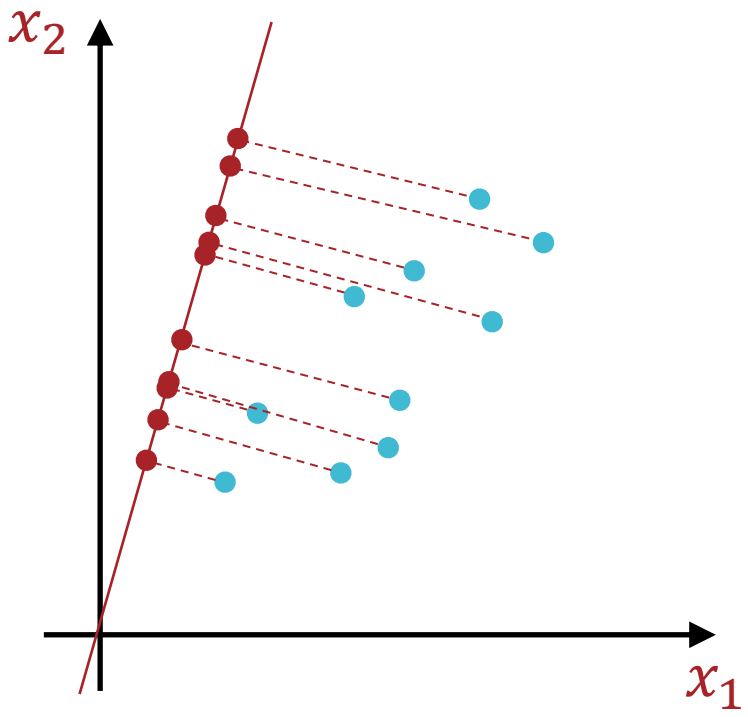


Feature Elimination

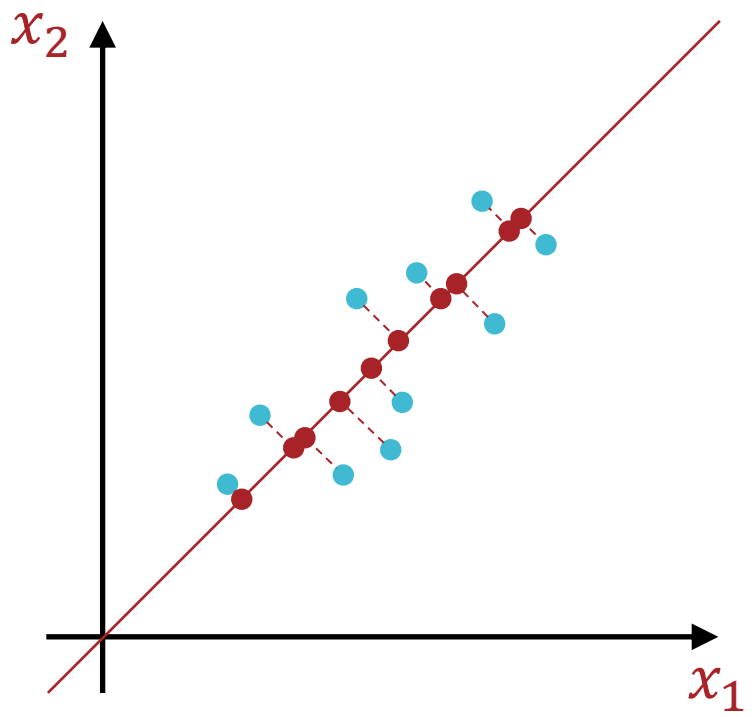




Feature Reduction



Option A



Option B

Which projection do you prefer?

Centering the Data

- To be consistent, we will constrain principal components to be *orthogonal unit vectors* that begin at the origin
- Preprocess data to be centered around the origin:

1. $\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}^{(n)}$
2. $\tilde{\boldsymbol{x}}^{(n)} = \boldsymbol{x}^{(n)} - \boldsymbol{\mu}\ \forall\ n$
3. $X = \begin{bmatrix} \tilde{\boldsymbol{x}}^{(1)^T} \\ \tilde{\boldsymbol{x}}^{(2)^T} \\ \vdots \\ \tilde{\boldsymbol{x}}^{(N)^T} \end{bmatrix}$

Reconstruction Error

- The projection of $\tilde{\boldsymbol{x}}^{(n)}$ onto a vector $\boldsymbol{v}$ is

$$\boldsymbol{z}^{(n)} = \left(\frac{\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}}{\|\boldsymbol{v}\|_2} \right) \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2}$$

Length of projection

Direction of projection

Reconstruction Error

- The projection of $\tilde{\boldsymbol{x}}^{(n)}$ onto a unit vector $\boldsymbol{v}$ is

$$\boldsymbol{z}^{(n)} = \left(\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}\right)\boldsymbol{v}$$

$$\hat{\boldsymbol{v}} = \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmin}} \sum_{n=1}^{N} \left\| \tilde{\boldsymbol{x}}^{(n)} - \left(\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}\right)\boldsymbol{v} \right\|_2^2$$

$$\|\tilde{x}^{(n)} - (v^T\tilde{x}^{(n)})v\|_2^2 = (\tilde{x}^{(n)} - (v^T\tilde{x}^{(n)})v)^T(\ \dots\ )$$

$$= \tilde{x}^{(n)T}\tilde{x}^{(n)} - 2(v^T\tilde{x}^{(n)})(v^T\tilde{x}^{(n)}) + (v^T\tilde{x}^{(n)})(v^T\tilde{x}^{(n)})$$

$v^Tv = (1)$

$$= \tilde{x}^{(n)T}\tilde{x}^{(n)} - (v^T\tilde{x}^{(n)})^2$$

$$= \|\tilde{x}^{(n)}\|_2^2 - (v^T\tilde{x}^{(n)})^2$$

Minimizing the Reconstruction Error

⇕

Maximizing the Variance

$$\hat{\boldsymbol{v}} = \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmin}} \sum_{n=1}^{N} \left\| \tilde{\boldsymbol{x}}^{(n)} - \left(\boldsymbol{v}^T \tilde{\boldsymbol{x}}^{(n)}\right)\boldsymbol{v} \right\|_2^2$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmin}} \sum_{n=1}^{N} \left(\|\tilde{x}^{(n)}\|_2^2 - \left(v^T \tilde{x}^{(n)}\right)^2 \right)$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} \sum_{n=1}^{N} \left(v^T \tilde{x}^{(n)}\right)^2$$

variance of centered projections

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} \sum_{n=1}^{N} v^T \tilde{x}^{(n)} \tilde{x}^{(n)T} v$$

$$= \underset{v:\|v\|_2^2=1}{\operatorname{argmax}} \; v^T \left(\sum_{n=1}^{N} \tilde{x}^{(n)} \tilde{x}^{(n)T} \right) v$$

$\left(X^T X\right)$

Maximizing the Variance

$$\hat{\boldsymbol{v}} = \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmax}} \boldsymbol{v}^T (X^T X)\boldsymbol{v}$$

$$\mathcal{L}(v, \lambda) = v^T (X^T X) v - \lambda (v^T v - 1)$$

$$\frac{\partial \mathcal{L}}{\partial v} = (X^T X) v - 2\lambda v$$

$$\Rightarrow (X^T X)\hat{v} - 2\lambda \hat{v} = 0$$

$$\Rightarrow (X^T X)\hat{v} = 2\lambda \hat{v}$$

$\hat{v}$ is an eigenvector of $X^T X$

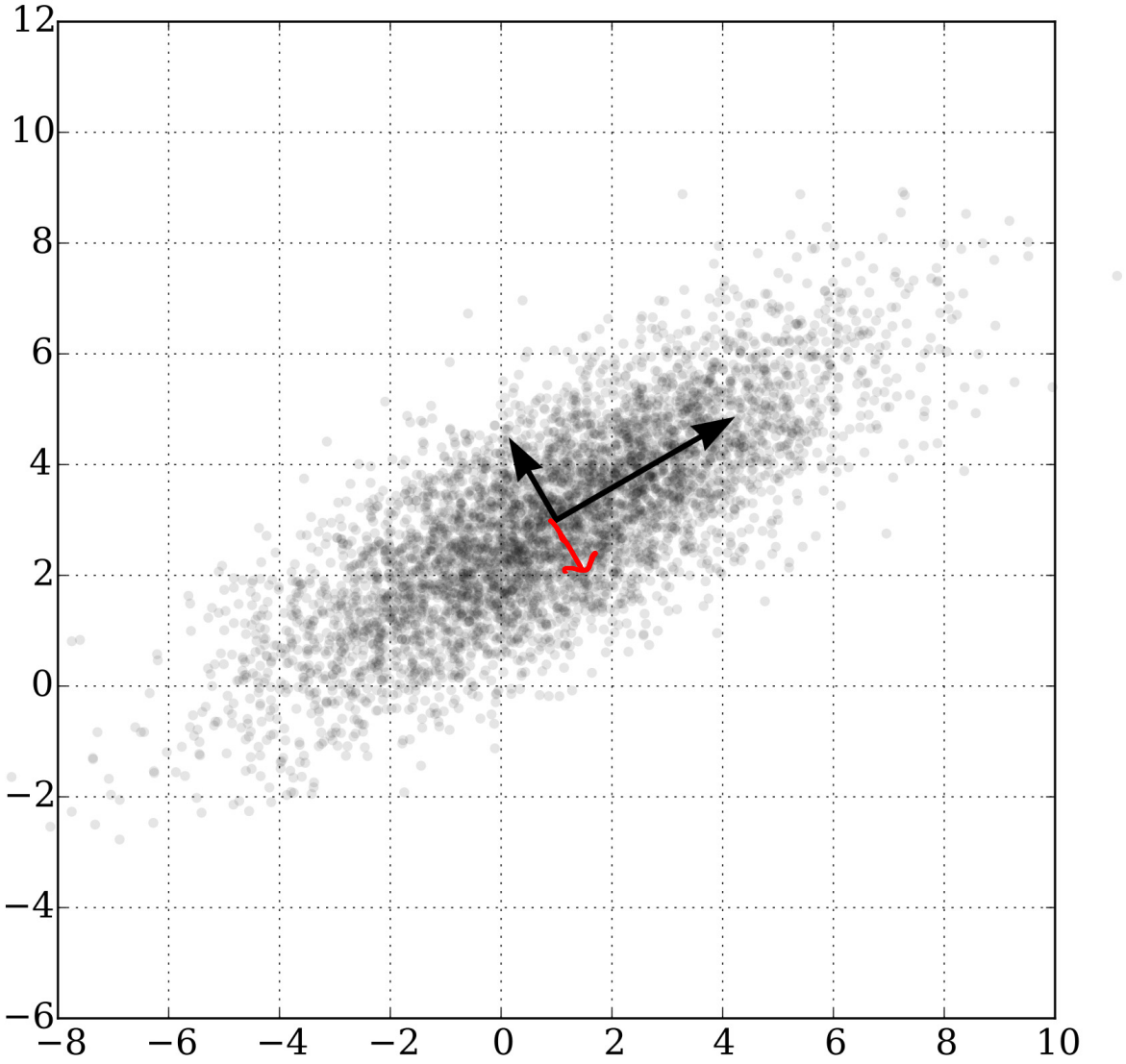
Maximizing the Variance

$$\hat{\boldsymbol{v}} = \underset{\boldsymbol{v}:\|\boldsymbol{v}\|_2^2=1}{\operatorname{argmax}} \boldsymbol{v}^T(X^TX)\boldsymbol{v}$$

$$(X^TX)\hat{\boldsymbol{v}} = \lambda\hat{\boldsymbol{v}} \rightarrow \hat{\boldsymbol{v}}^T(X^TX)\hat{\boldsymbol{v}} = \lambda\hat{\boldsymbol{v}}^T\hat{\boldsymbol{v}} = \lambda$$

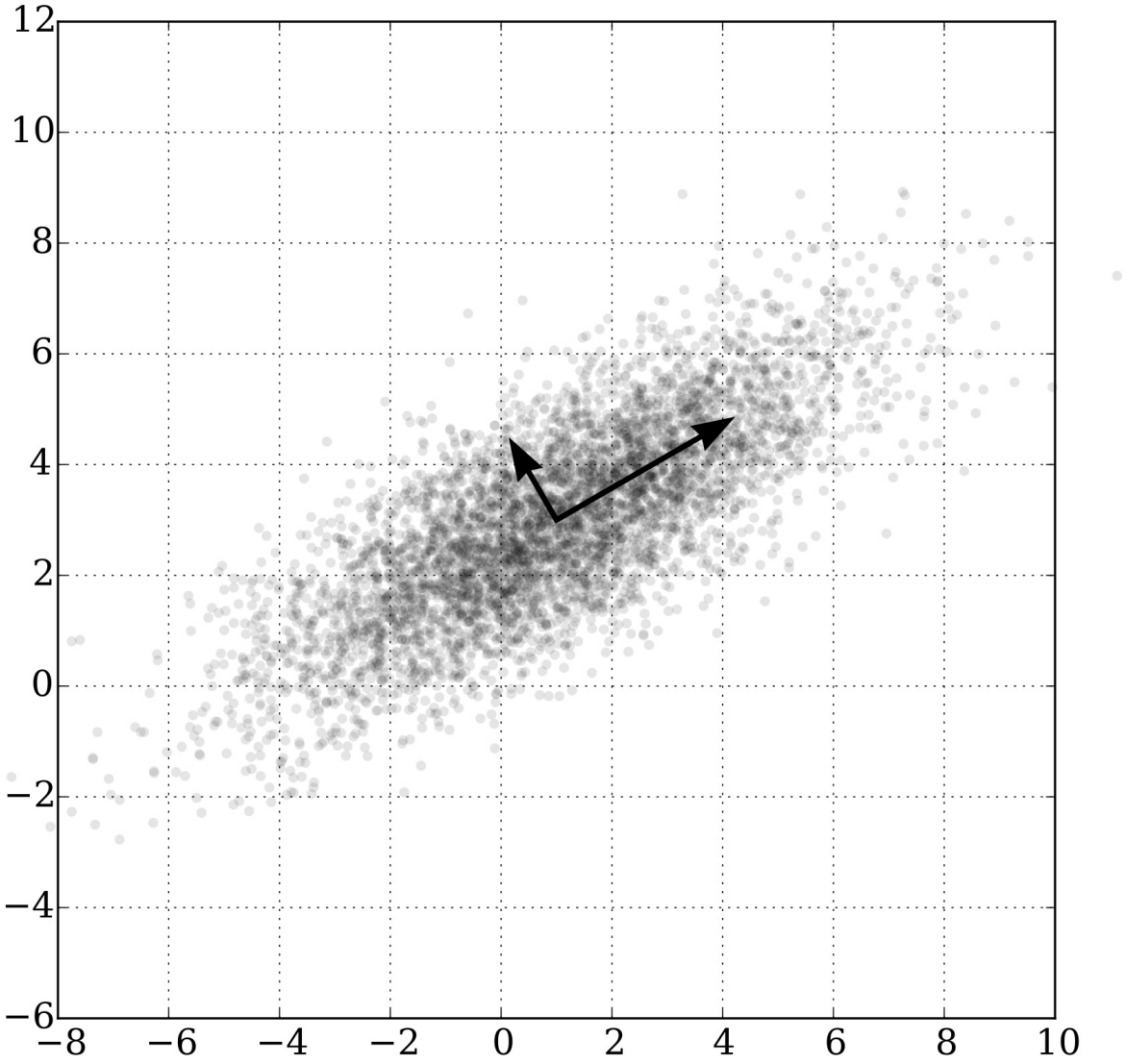
- The first principal component is the eigenvector $\hat{\boldsymbol{v}}_1$ that corresponds to the largest eigenvalue λ_1
- The second principal component is the eigenvector $\hat{\boldsymbol{v}}_2$ that corresponds to the second largest eigenvalue λ_1
  - $\hat{\boldsymbol{v}}_1$ and $\hat{\boldsymbol{v}}_2$ are orthogonal
- Etc …
- λ_i is a measure of how much variance falls along $\hat{\boldsymbol{v}}_i$

Principal Components: Example



Source: https://en.wikipedia.org/wiki/Principal_component_analysis#/media/File:GaussianScatterPCA.svg

How can we efficiently find principal components (eigenvectors)?



Source: https://en.wikipedia.org/wiki/Principal_component_analysis#/media/File:GaussianScatterPCA.svg

Singular Value Decomposition (SVD) for PCA

- Every real-valued matrix $X \in \mathbb{R}^{N \times D}$ can be expressed as

$$X = USV^T$$

where:

1. $U \in \mathbb{R}^{N \times N}$ - columns of U are eigenvectors of XX^T
2. $V \in \mathbb{R}^{D \times D}$ - columns of V are eigenvectors of $X^T X$
3. $S \in \mathbb{R}^{N \times D}$ - diagonal matrix whose entries are the eigenvalues of $X \rightarrow$ squared entries are the eigenvalues of XX^T and $X^T X$

PCA Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}\right)\right\}_{n=1}^{N}, \rho$

1. Center the data
2. Use SVD to compute the eigenvalues and eigenvectors of $X^T X$
3. Collect the top ρ eigenvectors (corresponding to the ρ largest eigenvalues), $V_\rho \in \mathbb{R}^{D \times \rho}$
4. Project the data into the space defined by V_ρ, $Z = XV_\rho$

- Output: Z, the transformed (potentially lower-dimensional) data

How many PCs should we use?

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}\right)\right\}_{n=1}^{N}, \rho$

1. Center the data
2. Use SVD to compute the eigenvalues and eigenvectors of $X^T X$
3. Collect the top ρ eigenvectors (corresponding to the ρ largest eigenvalues), $V_\rho \in \mathbb{R}^{D \times \rho}$
4. Project the data into the space defined by V_ρ, $Z = XV_\rho$

- Output: Z, the transformed (potentially lower-dimensional) data

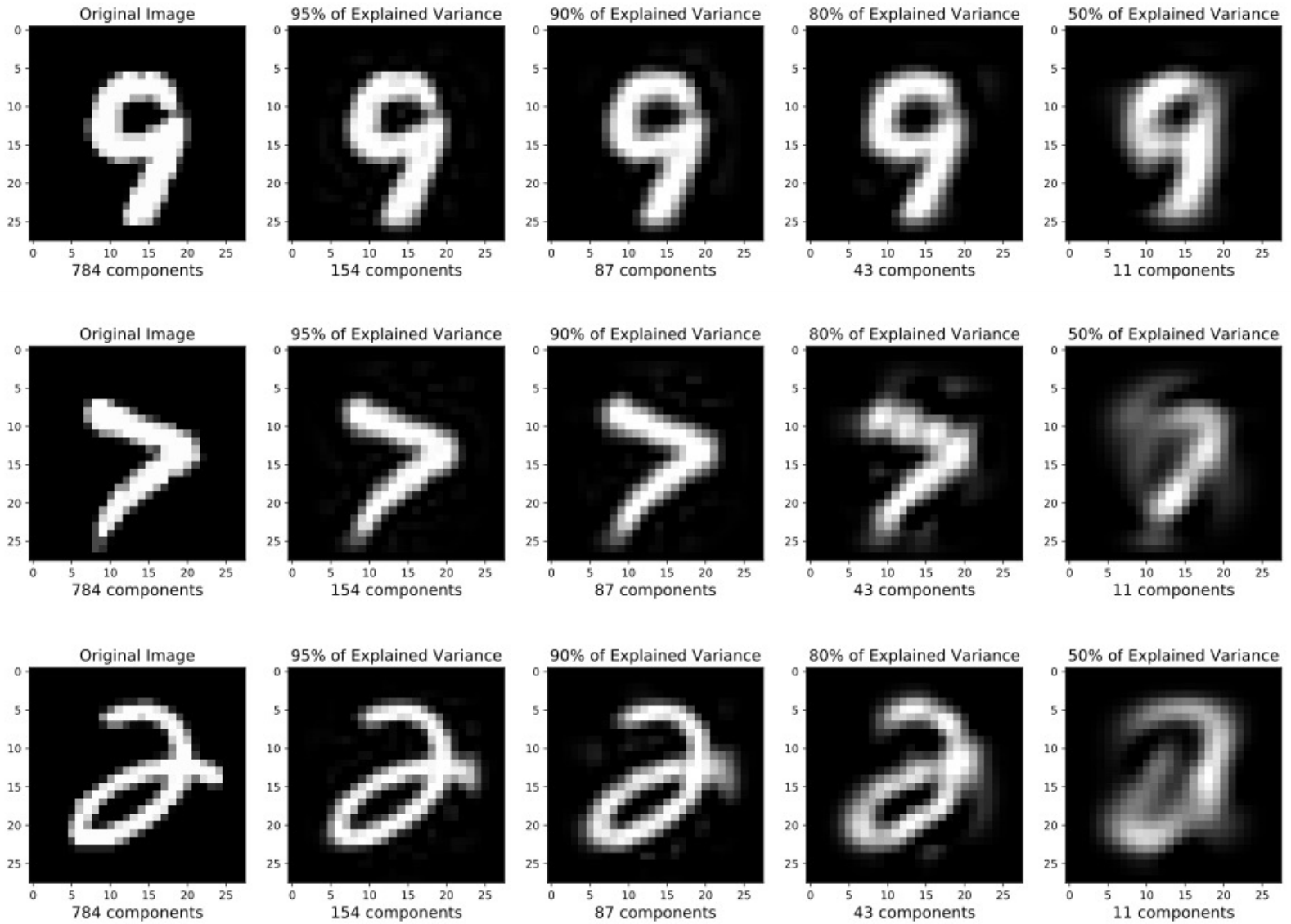
Choosing the number of PCs

- Define a percentage of explained variance for the i^{th} PC:

$$\lambda_i \Big/ \sum \lambda_j$$

- Select all PCs above some threshold of explained variance, e.g., 5%
- Keep selecting PCs until the total explained variance exceeds some threshold, e.g., 90%
- Evaluate on some downstream metric

PCA Example: MNIST Digits



Figures courtesy of Matt Gormley

PCA Example: MNIST Digits

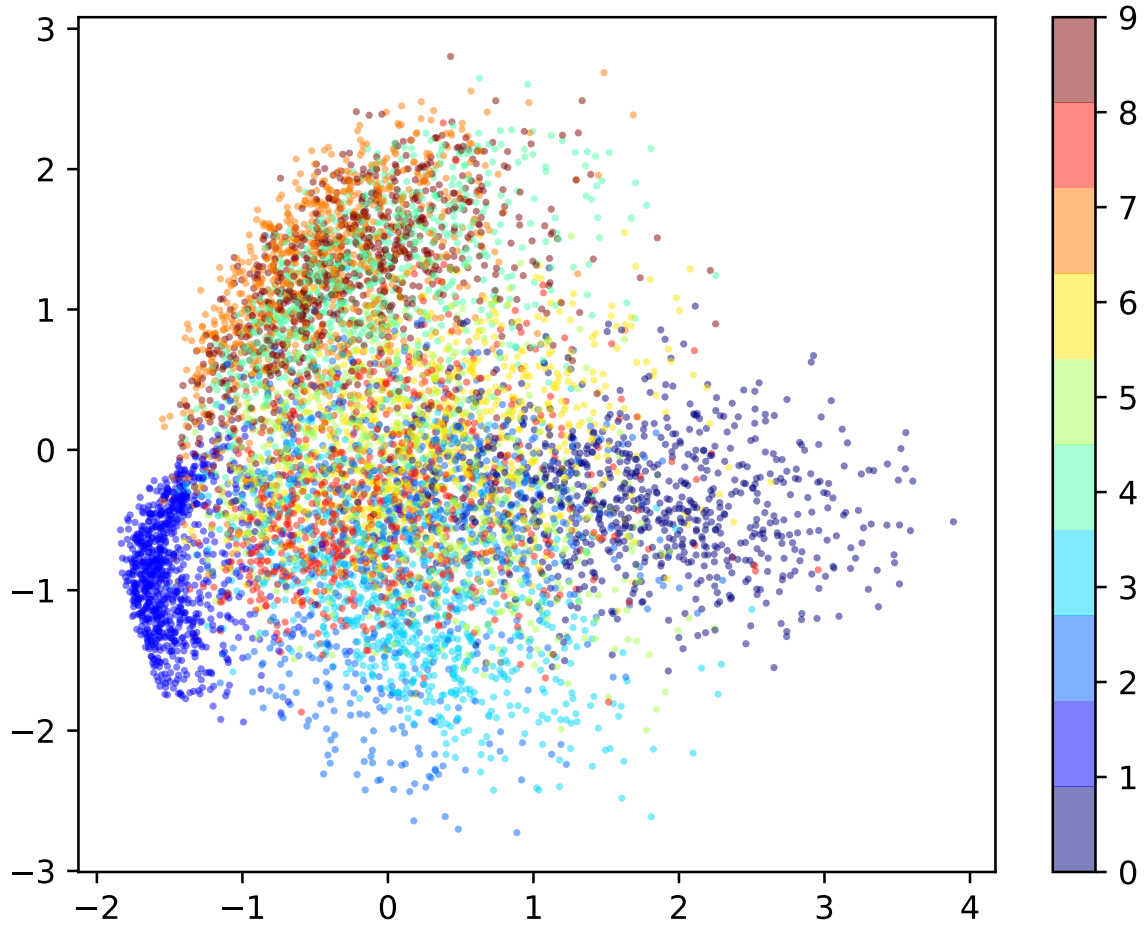


Figure courtesy of Matt Gormley

PCA Example: MNIST Digits

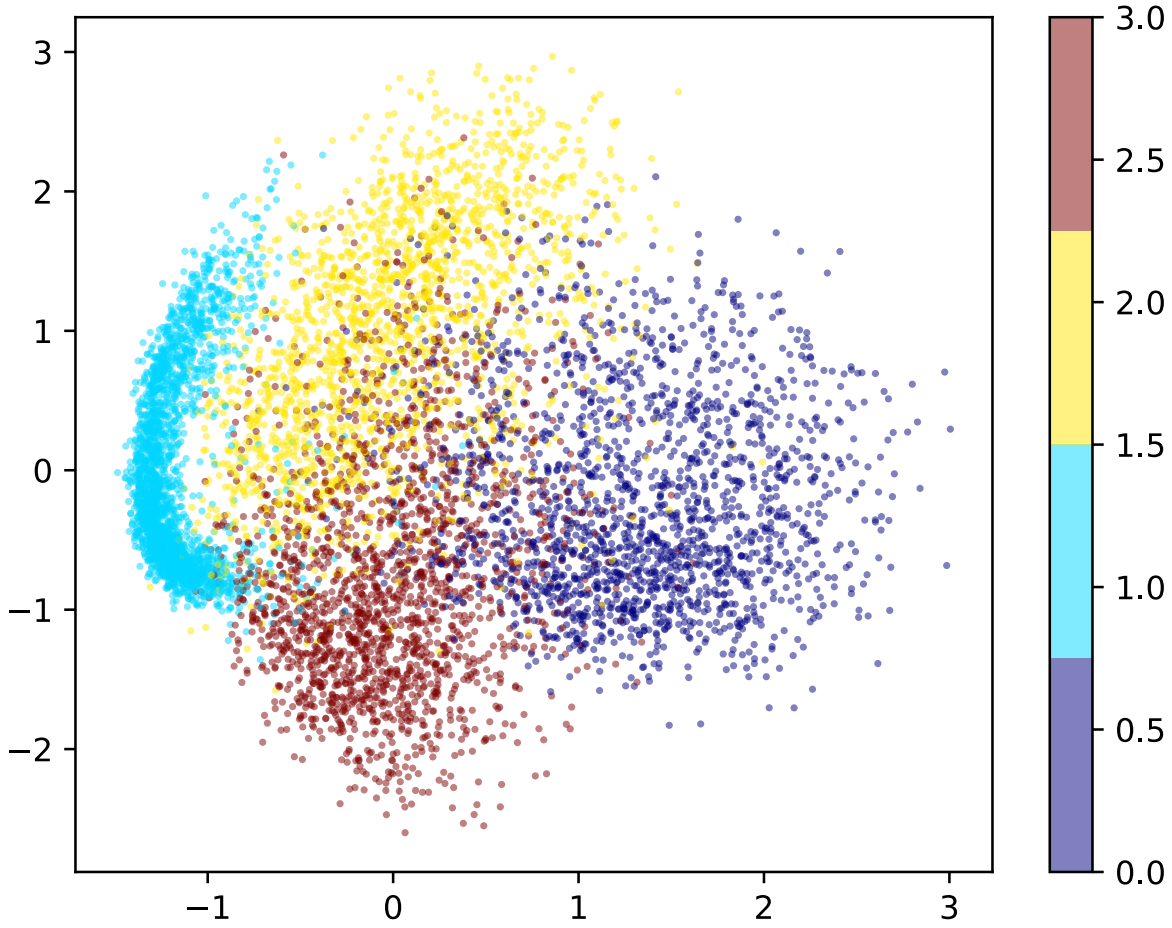
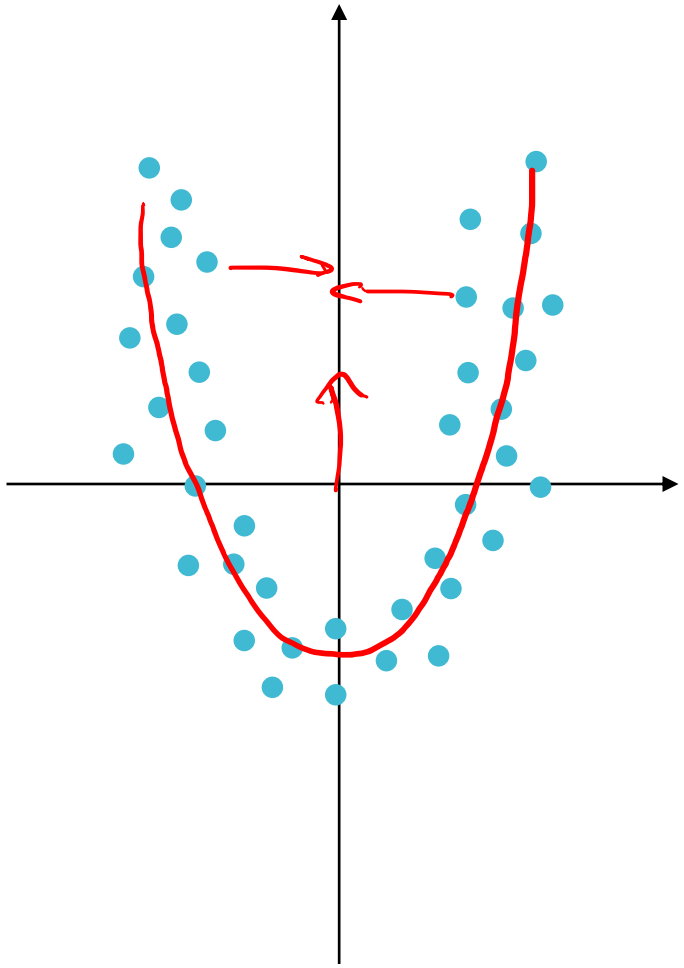


Figure courtesy of Matt Gormley

Shortcomings of PCA



- PCs always correspond to a linear projection
- PCs always orthogonal

Kernel PCA

- Claim: Principal components can be expressed as linear combinations of the (centered) data
- Proof:

$$(X^T X)\hat{v} = \lambda \hat{v} \Rightarrow \left(\sum_{n=1}^{N} \tilde{x}^{(n)} \tilde{x}^{(n)T}\right)\hat{v} = \lambda \hat{v}$$

$$\Rightarrow \frac{1}{\lambda} \sum_{n=1}^{N} a_n \tilde{x}^{(n)} = \hat{v} \qquad a_n$$

$$\Rightarrow \frac{1}{\lambda} X^T \hat{a} = \hat{v}$$

Kernel PCA

- Claim: Principal components can be expressed as linear combinations of the (centered) data
- Consequence:

$$\hat{v} = \frac{1}{\lambda} X^T \hat{a}$$

$$(X^T X)\hat{v} = \lambda \hat{v} \Rightarrow (X^T X)\left(\frac{1}{\lambda} X^T \hat{a}\right) = \lambda \frac{1}{\lambda} X^T \hat{a}$$

$$\Rightarrow \frac{1}{\lambda} X^T X X^T \hat{a} = X^T \hat{a}$$

$$\Rightarrow \frac{1}{\lambda} X X^T X X^T \hat{a} = X X^T \hat{a}$$

Recall: The Kernel Trick

- Approach: instead of computing $\Phi(\boldsymbol{x})$, find some function K_Φ s.t. $K_\Phi(\boldsymbol{x}, \boldsymbol{x}') = \Phi(\boldsymbol{x})^T \Phi(\boldsymbol{x}') \; \forall \; \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$
  - $K_\Phi(\boldsymbol{x}, \boldsymbol{x}')$ should be cheaper to compute than $\Phi(\boldsymbol{x})$
- Given some data set $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}\right)\right\}_{i=1}^{N}$, define the Gram matrix of a kernel K as

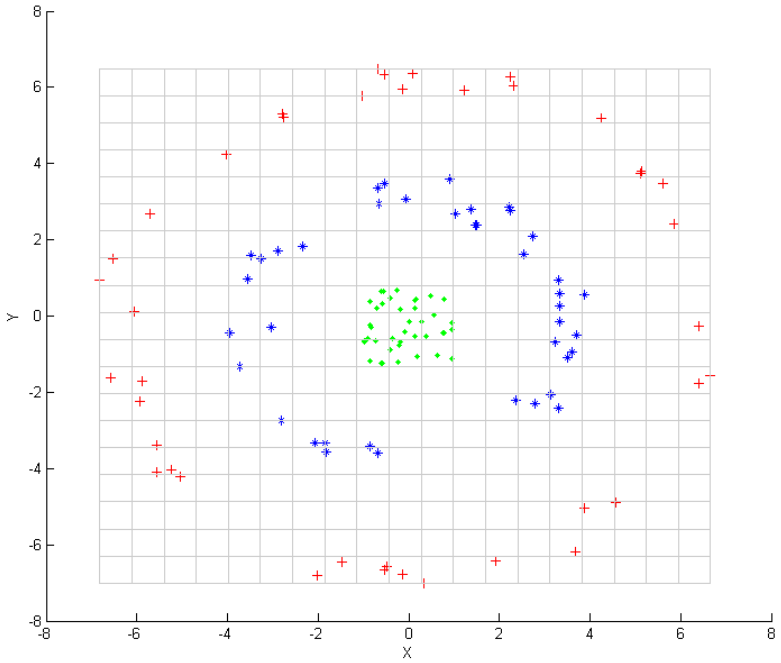
$$\mathrm{K} = \begin{bmatrix} K\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}\right) & K\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) & \cdots & K\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(N)}\right) \\ K\left(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}\right) & K\left(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(2)}\right) & \cdots & K\left(\boldsymbol{x}^{(2)}, \boldsymbol{x}^{(N)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ K\left(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(1)}\right) & K\left(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(2)}\right) & \cdots & K\left(\boldsymbol{x}^{(N)}, \boldsymbol{x}^{(N)}\right) \end{bmatrix}$$

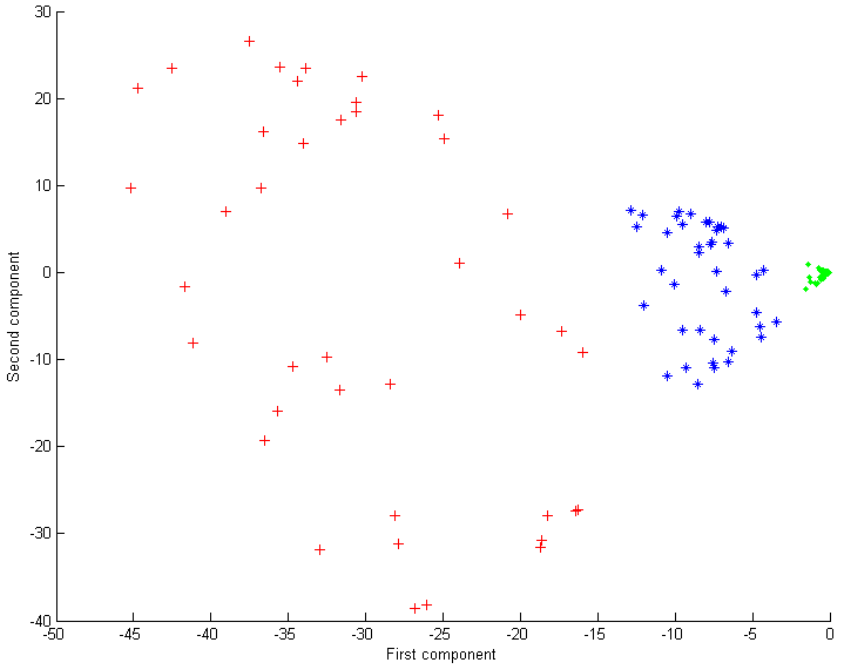
Kernel PCA

$$\mathrm{K}\mathrm{K}\hat{\boldsymbol{a}} = \lambda\mathrm{K}\hat{\boldsymbol{a}} \rightarrow \mathrm{K}\hat{\boldsymbol{a}} = \lambda\hat{\boldsymbol{a}}$$

- Principal components are the eigenvectors of the Gram matrix K if the data is centered in the transformed space
- We can center the transformed data *without explicitly computing the transformations* (see Bishop 12.3 for complete details):

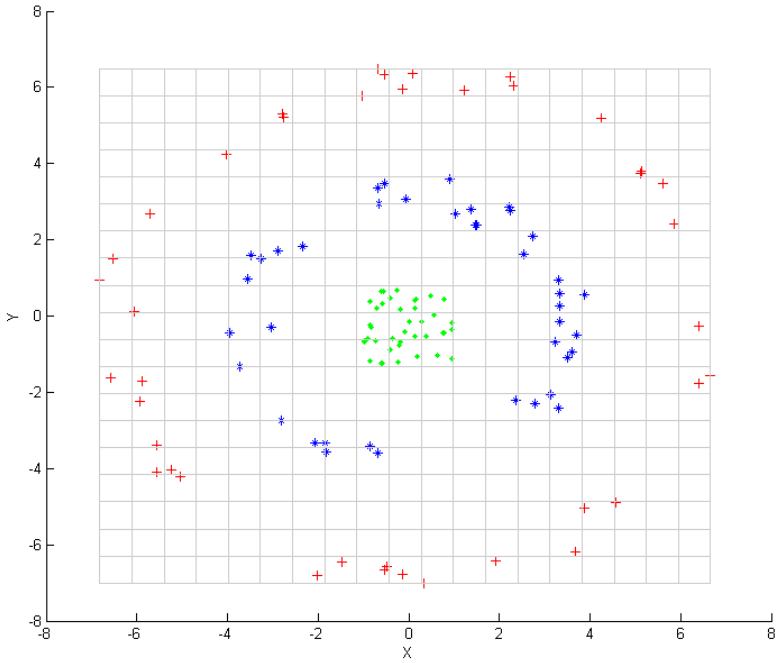
$$\widetilde{\mathrm{K}} = \mathrm{K} - \frac{1}{N}I\mathrm{K} - \frac{1}{N}\mathrm{K}I + \frac{1}{N^2}I\mathrm{K}I$$

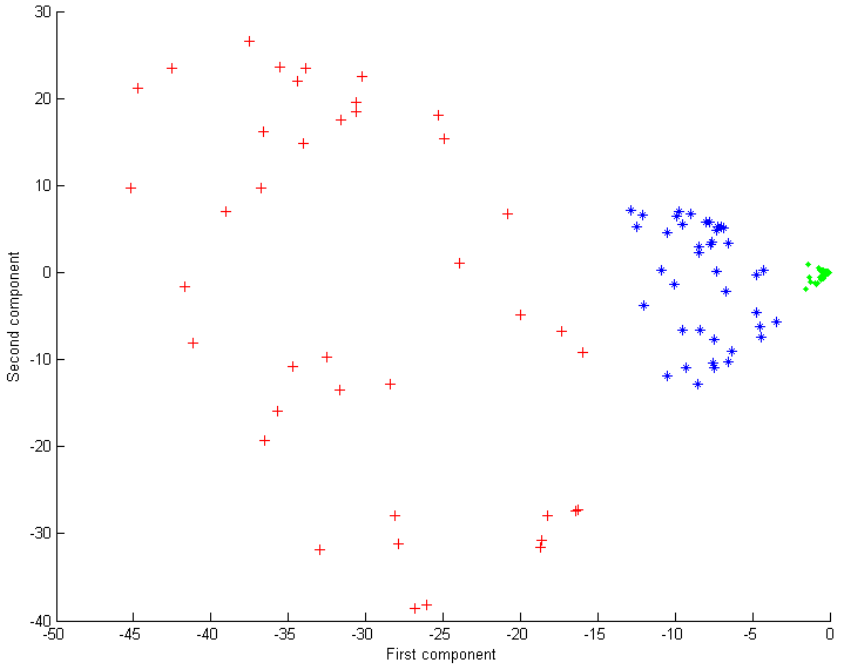




Kernel PCA: Example

Source: https://en.wikipedia.org/wiki/Kernel_principal_component_analysis





PCs are still orthogonal…

Source: https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

Independent Component Analysis (ICA)

- Assume our data is a linear transformation of arbitrary (not necessarily orthogonal) components ("signals")

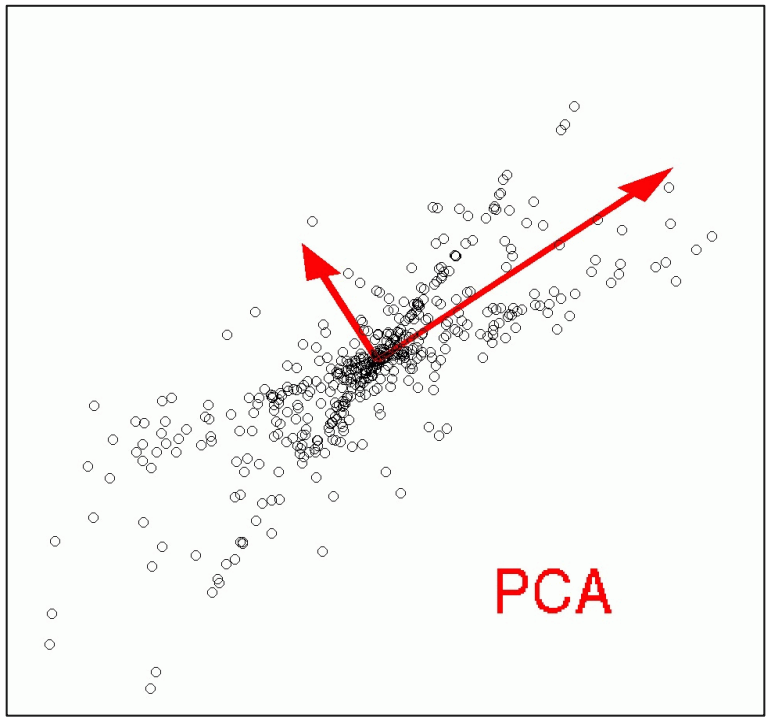
$$\boldsymbol{x}^{(i)} = A\boldsymbol{s}^{(i)}$$

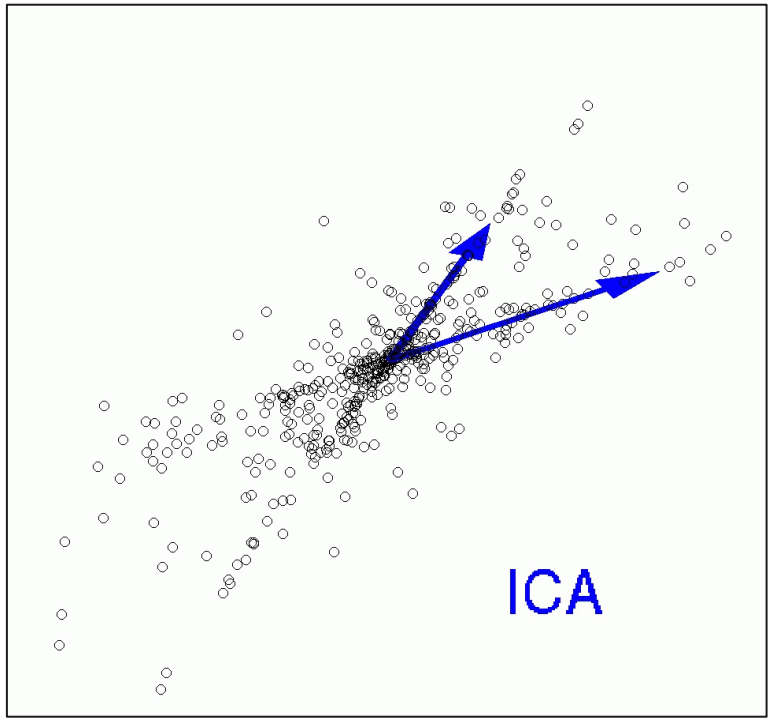
  - Typically assume $\boldsymbol{s}^{(i)}$ is the same size as $\boldsymbol{x}^{(i)}$

- Goal: find components that are as statistically independent as possible:

$$p\left(s_1^{(i)}, \dots, s_D^{(i)}\right) \approx p\left(s_1^{(i)}\right) \dots p\left(s_D^{(i)}\right)$$

- Common approach: minimize the mutual information between $s_1^{(i)}, \dots, s_D^{(i)}$

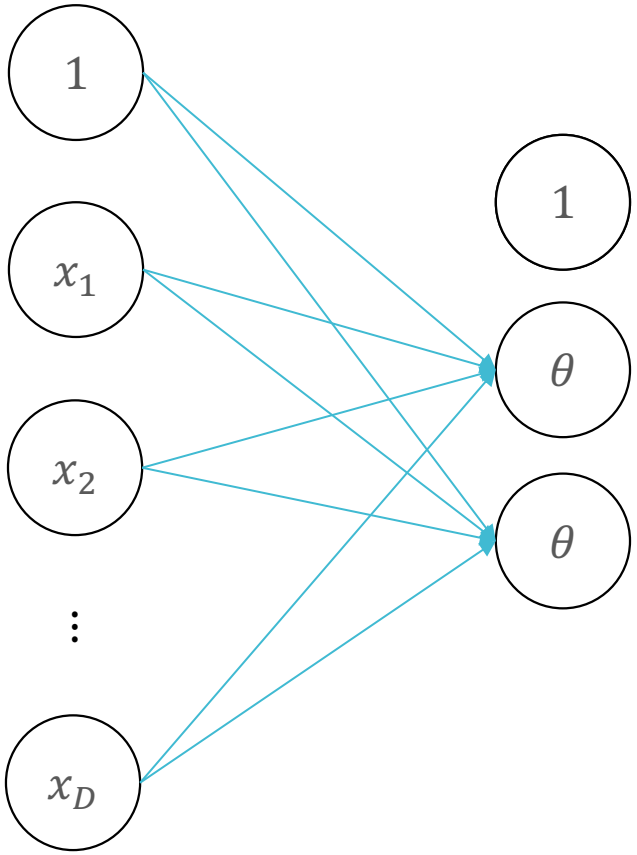




PCA vs. ICA

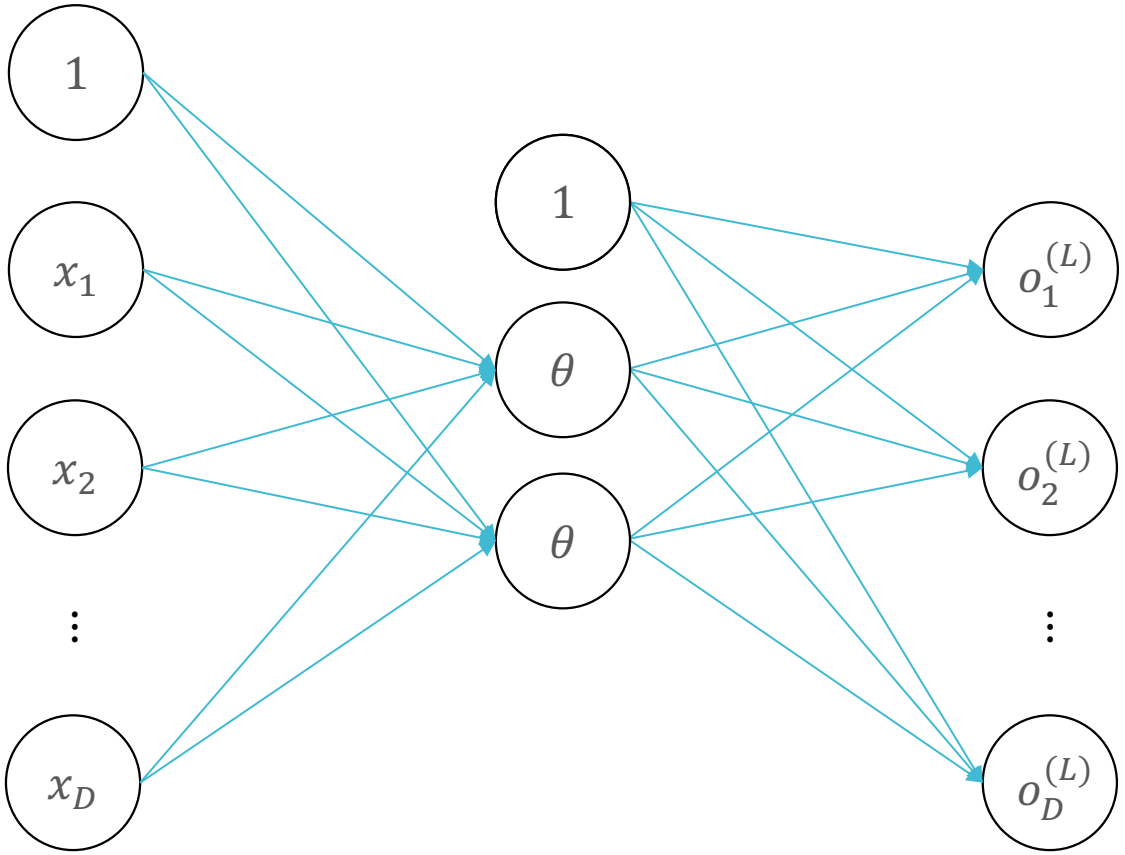
Figure courtesy of Barnabás Póczos

Autoencoders



Insight: neural networks implicitly learn low-dimensional representations of inputs in hidden layers

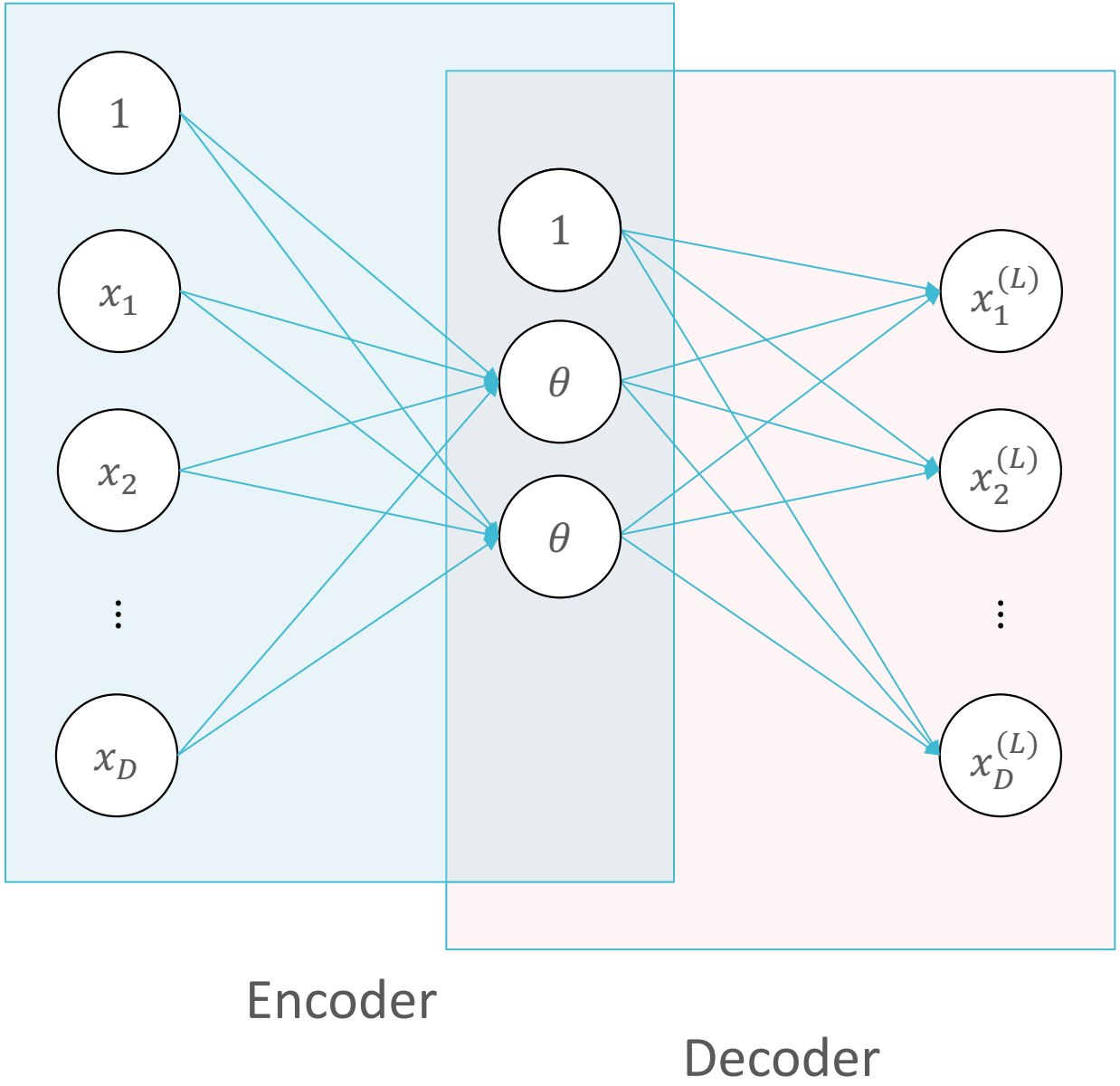
Autoencoders



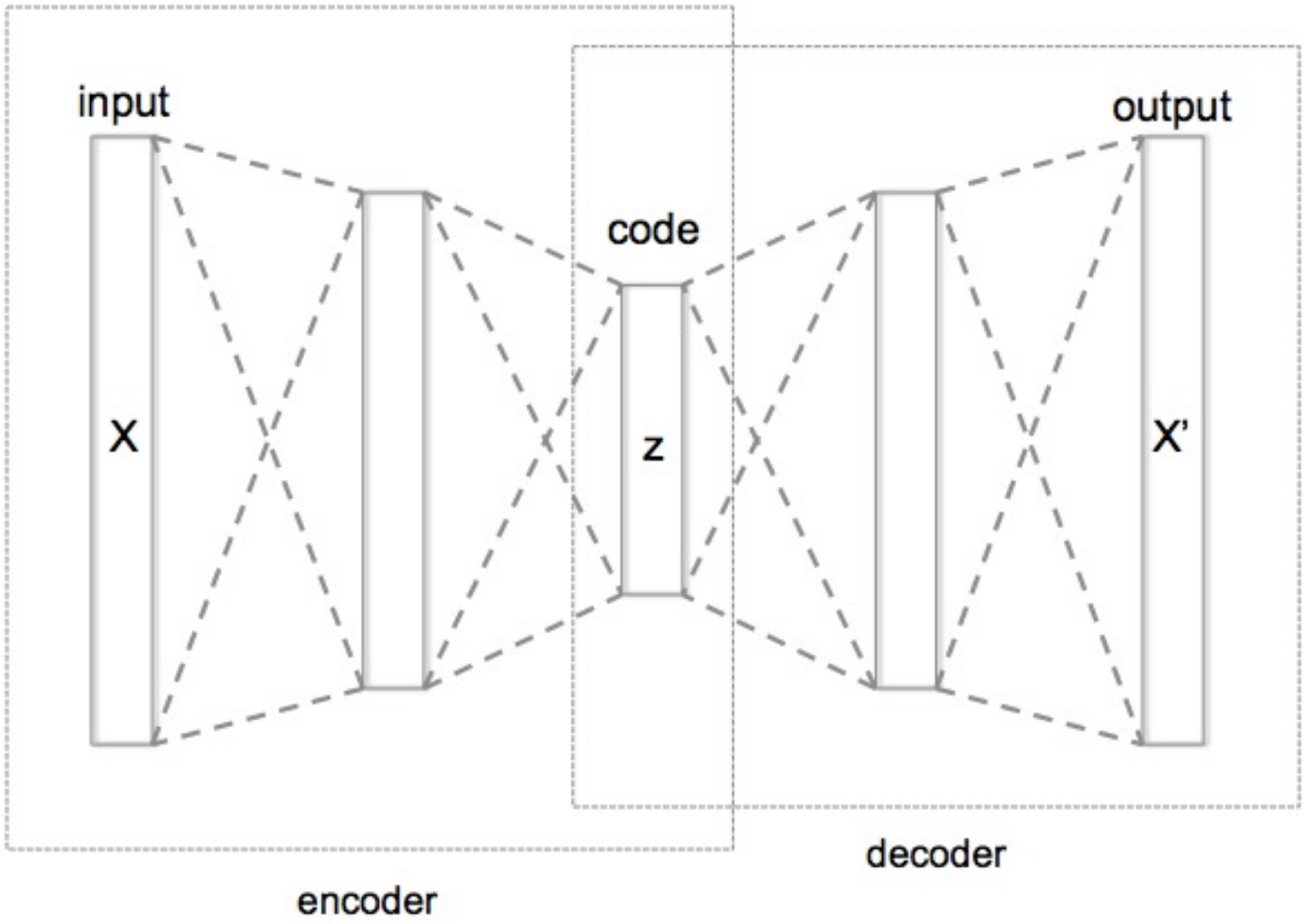
- Learn the weights by minimizing the reconstruction loss:

$$e(\boldsymbol{x}) = \left\| \boldsymbol{x} - \boldsymbol{o}^{(L)} \right\|_2^2$$

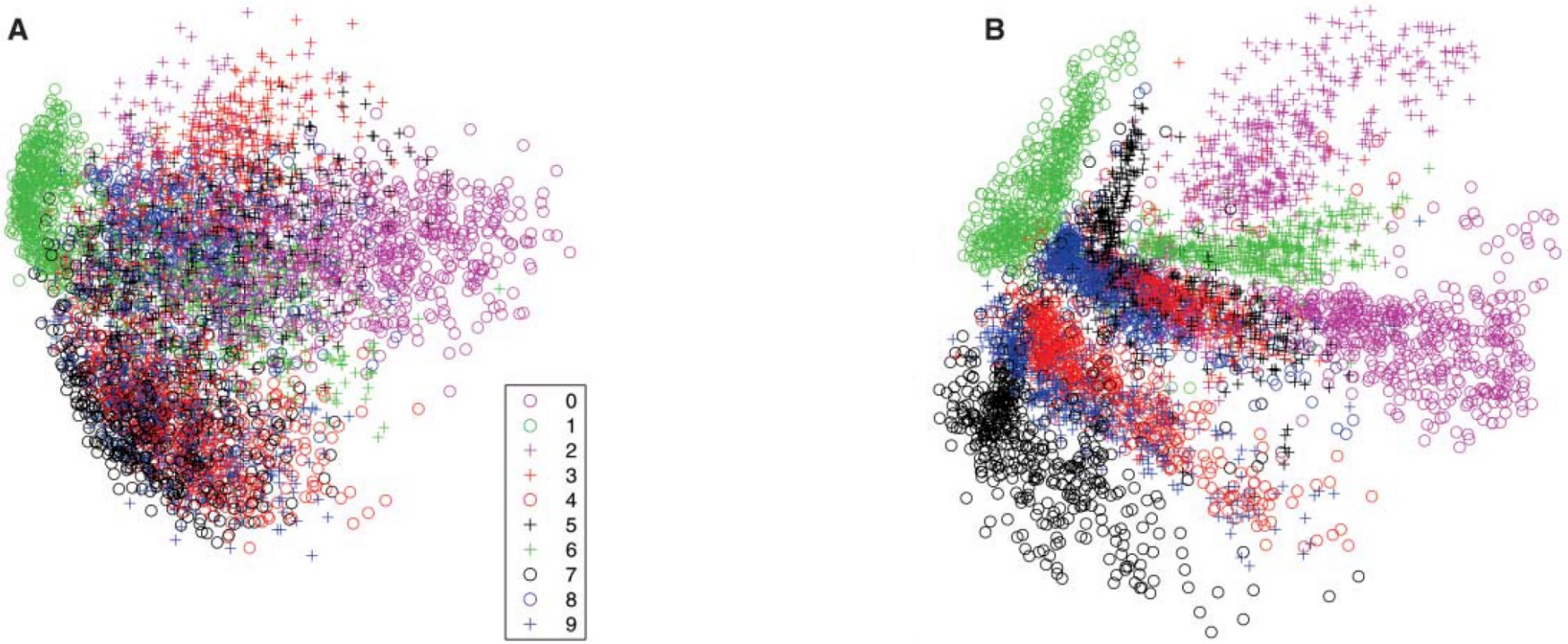
Autoencoders



Deep Autoencoders



Source: https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png



PCA (A) vs. Autoencoders (B) (Hinton and Salakhutdinov, 2014)

Source: https://www.science.org/doi/10.1126/science.1127647

Key Takeaways

- K-means partitions the dataset into K groups using block coordinate descent
  - The K-means objective function is non-convex
  - K-means++ can help avoid poor initializations
- PCA finds an orthonormal basis where the first principal component maximizes the variance $\Leftrightarrow$ minimizes the reconstruction error
  - Can be kernelized
- ICA finds statistically independent, not orthogonal components
- Autoencoders use neural networks to automatically learn a latent representation that minimizes the reconstruction error