Clustering (contd), and Dimensionality Reduction

CS771: Introduction to Machine Learning

Soft Clustering

- If clusters overlap, doing a "soft" clustering is more desirable
- Instead of hard assignment to a cluster, report cluster membership <u>probabilities</u>

Hard Clustering (assuming K=5)

0	**1**	**0**	**0**	**0**

$\boldsymbol{z}_n$ is a one-hot vector (only one z_{nk} is 1)

Soft Clustering (assuming K=5)

0.04	**0.65**	**0.15**	**0.10**	**0.06**

$\boldsymbol{z}_n$ is a probability vector ($\sum_{k=1}^{K} z_{nk} = 1$)

- Several methods exist for soft clustering such as latent variable models like Gaussian mixture models (will see later), and heuristics such as "soft" K-means

Hard K-means: each iteration

$$\boldsymbol{z}_n = \text{argmin}_k \, ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \quad \forall n$$

Only inputs with $z_{nk} = 1$ contribute to $\boldsymbol{\mu}_k$

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n:z_n=k} \boldsymbol{x}_n = \frac{\sum_{n=1}^{N} z_{nk}\boldsymbol{x}_n}{\sum_{n=1}^{N} z_{nk}} \quad \forall k$$

A heuristic to convert distances into the probability of $\boldsymbol{x}_n$ belonging to the k^{th} cluster

Soft/Fuzzy K-means: each iteration

$$z_{nk} = \frac{\exp(-||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2)}{\sum_{\ell=1}^{K} \exp(-||\boldsymbol{x}_n - \boldsymbol{\mu}_\ell||^2)} \quad \forall n, k$$

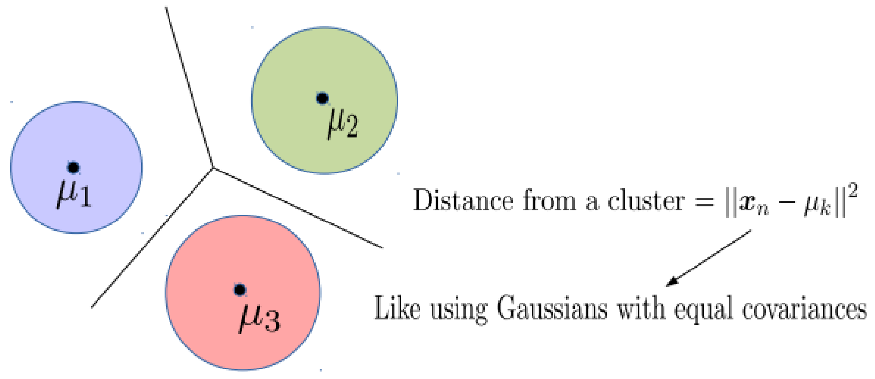
All N inputs contribute to $\boldsymbol{\mu}_k$ since $z_{nk} > 0 \;\; \forall \, k$

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} z_{nk}\boldsymbol{x}_n}{\sum_{n=1}^{N} z_{nk}} \quad \forall k$$

K-means: Some Other Limitations

- K-mean assumes that the decision boundary between any two clusters is linear
- Reason: The K-means loss function implies assumes equal-sized, spherical clusters

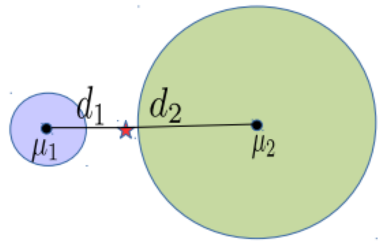
Reason: Use of Euclidean distances

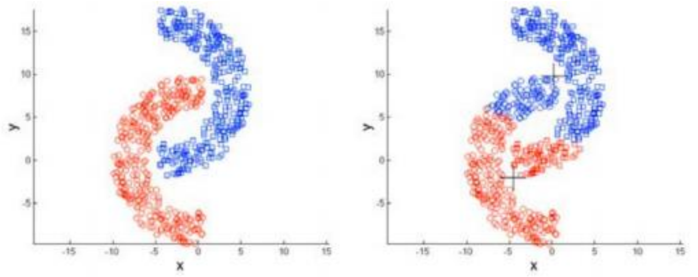


μ_1
μ_2
μ_3
Distance from a cluster $= ||\boldsymbol{x}_n - \mu_k||^2$
Like using Gaussians with equal covariances

Some of these issues can be addressed using probabilistic models for clustering (like mixture models) or using kernels

- May do badly if clusters are not roughly equi-sized and convex-shaped





Kernel K-means

Helps learn non-spherical clusters and nonlinear cluster boundaries

- Basic idea: Replace the Eucl. distances in K-means by the kernelized versions

Kernelized distance between input $\boldsymbol{x}_n$ and mean of cluster k

$$||\phi(\boldsymbol{x}_n) - \phi(\boldsymbol{\mu}_k)||^2 = ||\phi(\boldsymbol{x}_n)||^2 + ||\phi(\boldsymbol{\mu}_k)||^2 - 2\phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{\mu}_k)$$

- Here $k(.,.)$ denotes the kernel function and ϕ is its (implicit) feature map
- Note: $\phi(\mu_k)$ is the mean of ϕ mappings of the data points assigned to cluster k

Not the same as the ϕ mapping of the mean of the data points assigned to cluster k

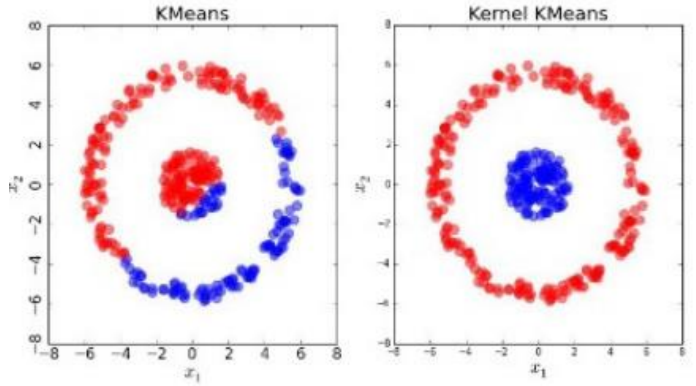
$$\phi(\boldsymbol{\mu}_k) = \frac{1}{|\mathcal{C}_k|} \sum_{n:z_n=k} \phi(\boldsymbol{x}_n)$$

Can also used landmarks or kernel random features idea to get new features and run standard k-means on those

$$||\phi(\boldsymbol{\mu}_k)||^2 = \phi(\boldsymbol{\mu}_k)^\top \phi(\boldsymbol{\mu}_k)$$

$$= \frac{1}{|\mathcal{C}_k|^2} \sum_{n:z_n=k} \sum_{n:z_m=k} k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$

$$\phi(\boldsymbol{x}_n)^\top \phi(\boldsymbol{\mu}_k) = \frac{1}{|\mathcal{C}_k|} \sum_{m:z_m=k} k(\boldsymbol{x}_n, \boldsymbol{x}_m)$$



Note: Apart from kernels, it is also possible to use other distance functions in K-means. Bregman Divergence* is such a family of distances (Euclidean and Mahalanobis are special cases)

*Clustering with Bregman Divergences (Banerjee et al, 2005)

Overlapping Clustering

Kind of unsupervised version of multi-label classification (just like standard clustering is like unsupervised multi-class classification)

Example: Clustering people based on the interests they may have (a person may have multiple interests; thus may belong to more than one cluster simultaneously)

- Have seen hard clustering and soft clustering
- In hard clustering, z_n is a one-hot vector
- In soft clustering, z_n is a vector of probabilities
- Overlapping Clustering: A point can simultaneously belong to multiple clusters
  - This is different from soft-clustering
  - z_n would be a binary vector, rather than a one hot or probability vector, e.g.,

$$z_n = [1\ 0\ 0\ 1\ 0]$$

K=5 clusters with point x_n belonging (in whole, not in terms of probabilities) to clusters 1 and 4

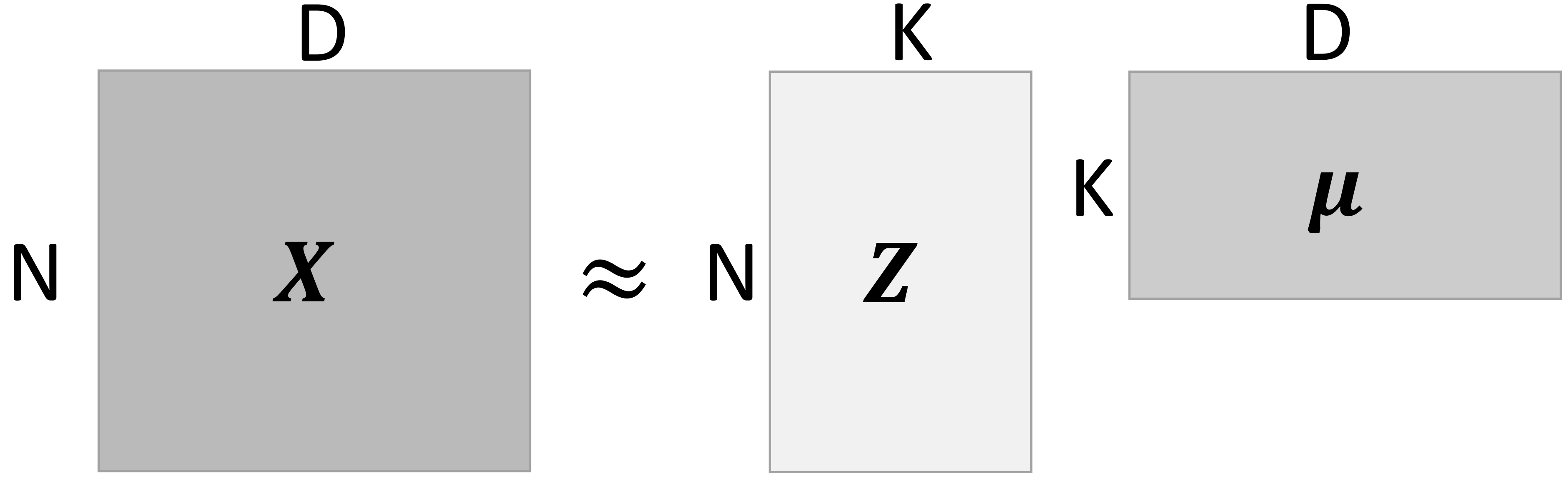
- In general, more difficult than hard/soft clustering (for N data points and K clusters, the size of the space of possible solutions is not K^N but 2^{NK} - exp in both N and K)
- K-means has extensions* for doing overlapping clustering. There also exist latent variable models for doing overlapping clustering

*An extended version of the k-means method for overlapping clustering (Cleuziou, 2008); Non-exhaustive, Overlapping k-means (Whang et al, 2015)

Clustering as Matrix Factorization

Also an unsupervised learning problem

- Clustering (hard, soft, overlapping) can also be posed as matrix factorization



- Minimize the distortion $||\boldsymbol{X} - \boldsymbol{Z\mu}||_F^2$ subject to suitable constraints on $\boldsymbol{Z}$, e.g.,
  - $\boldsymbol{z}_n$ is a one-hot vector
  - Entries of $\boldsymbol{z}_n$ are non-negative and sum to 1
  - $\boldsymbol{z}_n$ is a binary vector

Hierarchical Clustering

Similarity between two clusters (or two set of points) is needed in HC algos (e.g., this can be average pairwise similarity between the inputs in the two clusters)

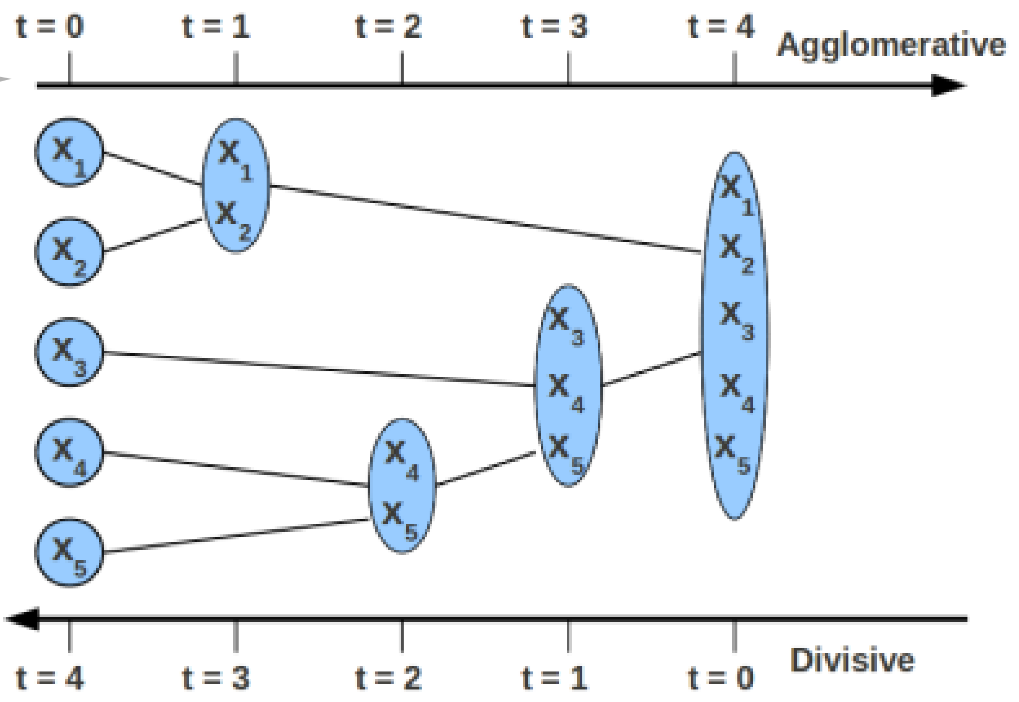
- Can be done in two ways: Agglomerative or Divisive

Agglomerative: Start with each point being in a singleton cluster

At each step, greedily merge two most "similar" sub-clusters

Stop when there is a single cluster containing all the points

Learns a dendrogram-like structure with inputs at the leaf nodes. Can then choose how many clusters we want



Keep recursing until the desired number of clusters found

At each step, break a cluster into (at least) two smaller homogeneous sub-clusters

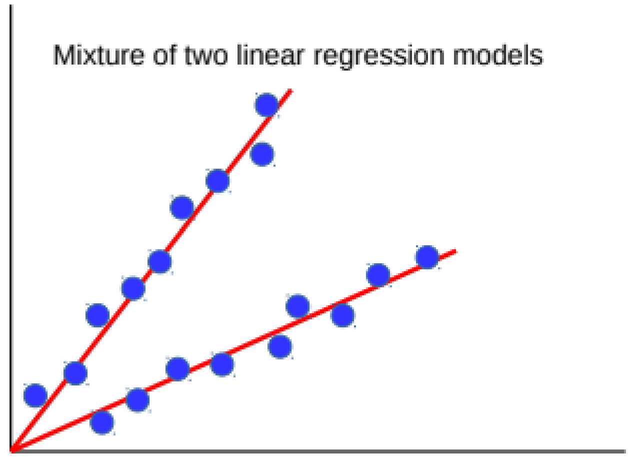
Divisive: Start with all points being in a single cluster

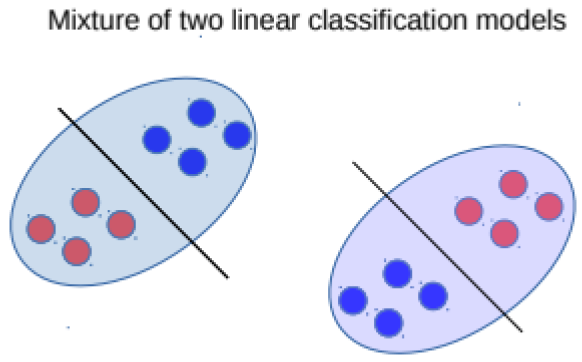
Tricky because no labels (unlike Decision Trees)

- Agglomerative is more popular and simpler than divisive (the latter usually needs complicated heuristics to decide cluster splitting).
- Neither uses any loss function

Clustering can help supervised learning, too

- Often "difficult" sup. learning problems can be seen as mixture of simpler models
- Example: Nonlinear regression or nonlinear classification as mixture of linear models





- Don't know which point should be modeled by which linear model ⇒ Clustering
- Can therefore solve such problems as follows
  - Initialize each linear model somehow (maybe randomly)
  - Cluster the data by assigning each point to its "closest" linear model (one that gives lower error)
  - (Re-)Learn a linear model for each cluster's data. Go to step 2 if not converged.

Such an approach is also an example of divide and conquer and is also known as "mixture of experts" (will see it more formally when we discuss latent variable models)

Evaluating Clustering Algorithms

- Clustering algos are in general harder to evaluate since we rarely know the ground truth clustering (since clustering is unsupervised)

- If ground truth labels not available, use output of clustering for some other task
  - For example, use cluster assignment z_n (hard or soft) as a new feature representation
  - Performance on some task using this new rep. is a measure of goodness of clustering

- If ground truth labels are available, can compare them with clustering based labels
  - Not straightforward to compute accuracy since the label identities may not be the same, e.g.,

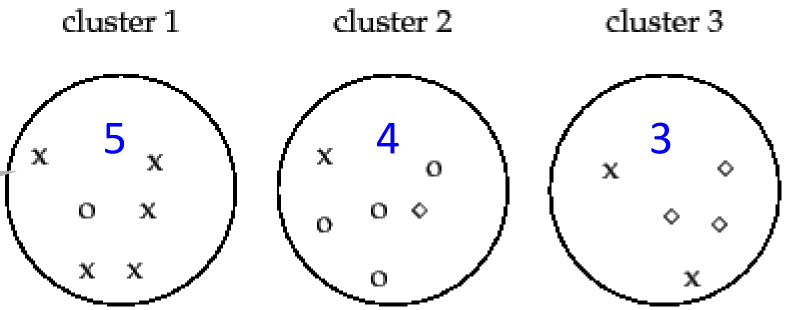
    Ground truth = [1 1 1 0 0 0]   Clustering = [0 0 0 1 1 1]

    (Perfect clustering but zero "accuracy" if we just do a direct match)
  - There are various metrics that take into account the above fact
    - Purity, Rand Index, F-score, Normalized Mutual Information, distortion or loss on test data etc

Evaluating Clustering Algorithms

- Purity: Looks at how many points in each cluster belong to the majority class in that cluster

3 classes (x, o, △, assuming known ground truth labels)



Sum and divide by total number of points

Purity = (5+4+3)/17 ≈ 0.71

Close to 0 for bad clustering, 1 for perfect clustering

Also a bad metric if number of clusters is very large – each cluster will be kind of pure anyway

- Rand Index (RI): Can also look at what fractions of pairs of points with same (resp. different) label are assigned to same (resp. different) cluster

True Positive: No. of pairs with same true label and same cluster

True Negative: No. of pairs with diff true label and diff clusters

$$\mathrm{RI} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}}$$

False Positive: No. of pairs with diff true label and same cluster

False Negative: No. of pairs with same true label and diff cluster

F_β score is also popular

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \qquad R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \qquad F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

Precision

Recall

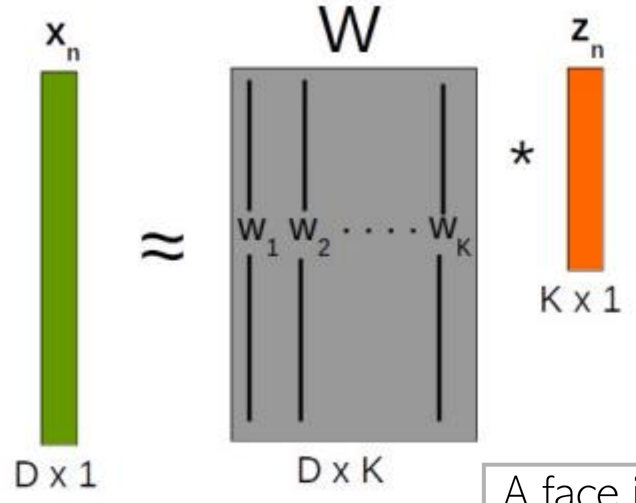
Dimensionality Reduction: A motivative example

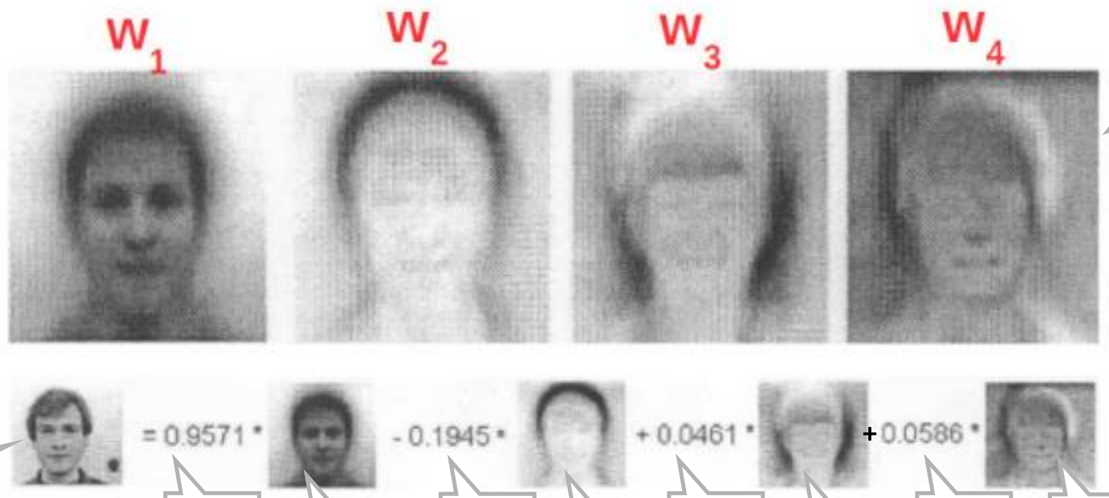
- Consider a linear model of the form

$$\boldsymbol{x}_n \approx \tilde{\boldsymbol{x}}_n = \boldsymbol{W}\boldsymbol{z}_n = \sum_{k=1}^{K} z_{nk}\boldsymbol{w}_k$$

$\boldsymbol{w}_k$ is the k-th column of $\boldsymbol{W}$

- Above means that each $\boldsymbol{x}_n$ is appox a linear comb of K vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \dots, \boldsymbol{w}_K$





$K = 4$ "basis" face images

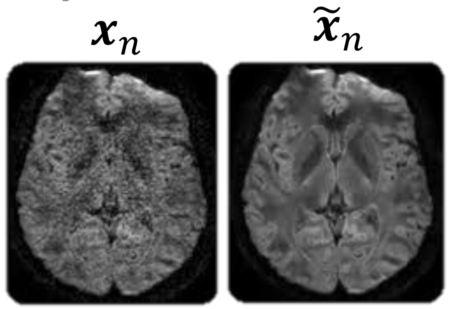
Each "basis" image is like a "template" that captures the common properties of face images in the dataset

A face image $\boldsymbol{x}_n \in \mathbb{R}^D$

z_{n1} $\boldsymbol{w}_1$ z_{n2} $\boldsymbol{w}_2$ z_{n3} $\boldsymbol{w}_3$ z_{n4} $\boldsymbol{w}_4$

- In this example, $\boldsymbol{z}_n \in \mathbb{R}^K$ $(K = 4)$ is a low-dim feature rep. for each image $\boldsymbol{x}_n \in \mathbb{R}^D$

Dimensionality Reduction: More formally..



- Goal: Reduce the dimensionality of each input $\boldsymbol{x}_n \in \mathbb{R}^D$

$\boldsymbol{z}_n \in \mathbb{R}^K$ $(K \ll D)$ is a compressed version of $\boldsymbol{x}_n$

$$\boldsymbol{z}_n = f(\boldsymbol{x}_n)$$

- Also want to be able to (approximately) reconstruct $\boldsymbol{x}_n$ from $\boldsymbol{z}_n$

Often $\tilde{\boldsymbol{x}}_n$ is a "cleaned" version of $\boldsymbol{x}_n$ (the loss in information is often the noise/redundant information in $\boldsymbol{x}_n$)

$$\tilde{\boldsymbol{x}}_n = g(\boldsymbol{z}_n) = g(f(\boldsymbol{x}_n)) \approx \boldsymbol{x}_n$$

- Sometimes f is called "encoder" and g is called "decoder". Can be linear/nonlinear

- These functions are learned by minimizing the distortion/reconstruction error of inputs

$$\mathcal{L} = \sum_{n=1}^{N} ||\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n||^2 = \sum_{n=1}^{N} ||\boldsymbol{x}_n - g(f(\boldsymbol{x}_n))||^2$$

Image source: Manjon et al (2013): Diffusion Weighted Image Denoising using overcomplete Local PCA