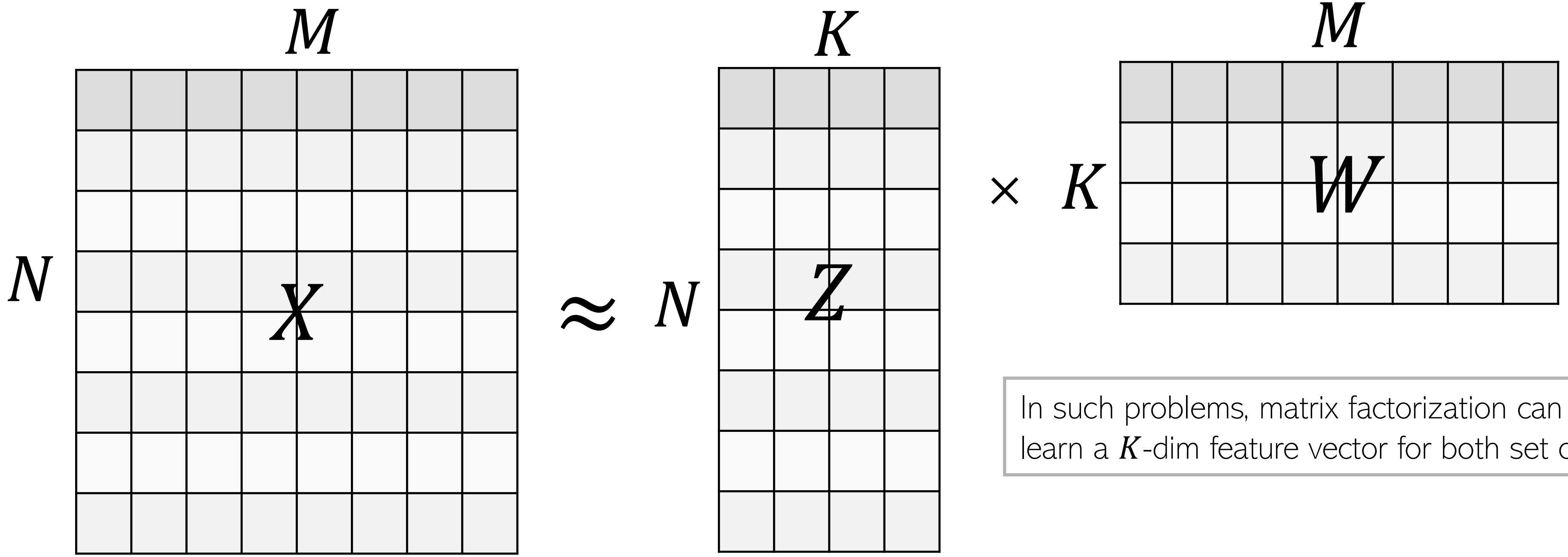
Dimensionality Reduction (wrap-up)

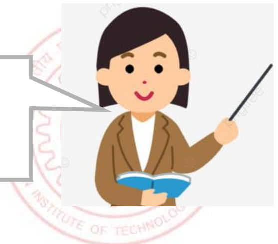
CS771: Introduction to Machine Learning

Matrix Factorization is a very useful method!

- In many problems, we are given co-occurrence data in form of an $N \times M$ matrix X
- Data consists of relationship b/w two sets of entities containing N and M entities
- Each entry X_{ij} denotes how many times the pair (i, j) co-occurs, e.g.,
  - Number of times a document i (total N docs) contains word j of a vocabulary (total M words)
  - Rating user i gave to item (or movie) j on a shopping (or movie streaming) website

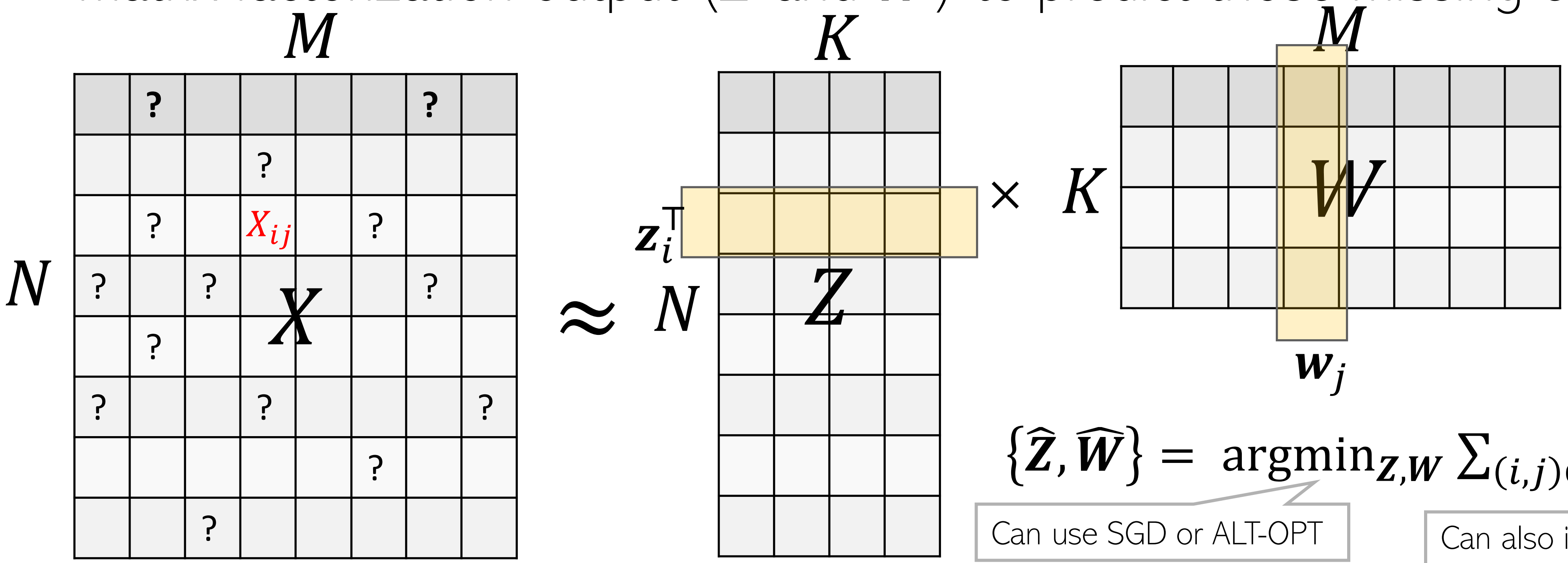


In such problems, matrix factorization can be used to learn a K-dim feature vector for both set of entities



Matrix Completion via Matrix Factorization

- If some entries of $\boldsymbol{X}$ are missing, we can still do matrix factorization and use the matrix factorization output ($\boldsymbol{Z}$ and $\boldsymbol{W}$) to predict those missing entries



Ω denotes the set of entries of X that are observed in training data

$$\{\hat{\boldsymbol{Z}}, \hat{\boldsymbol{W}}\} = \text{argmin}_{\boldsymbol{Z},\boldsymbol{W}} \sum_{(i,j)\in\Omega} (X_{ij} - \boldsymbol{z}_i^\top \boldsymbol{w}_j)$$

Can use SGD or ALT-OPT

Can also include regularization terms on these

- Once $\boldsymbol{Z}$ and $\boldsymbol{W}$ are learned, to predict a missing entry at location (i', j') as

$$X_{i'j'} \approx \boldsymbol{z}_{i'}^\top \boldsymbol{w}_{j'}$$

If K is small (as compared to N and M) then we call it low-rank matrix completion

Dim. Reduction by Preserving Pairwise Distances

- PCA/SVD etc assume we are given points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ as vectors (e.g., in D dim)
- Often the data is given in form of distances d_{ij} between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ $(i, j = 1,2, \ldots, N)$
- Would like to project data such that pairwise distances between points are preserved

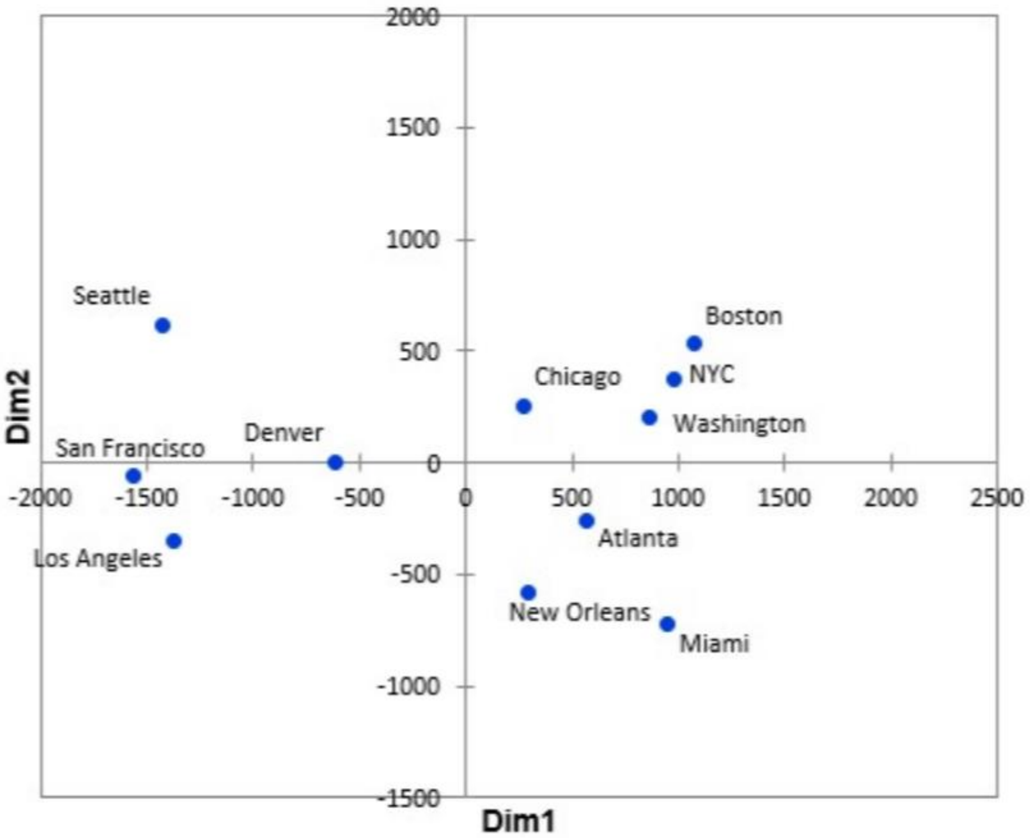
$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \mathcal{L}(\mathbf{Z}) = \arg\min_{\mathbf{Z}} \sum_{i,j=1}^{N} (d_{ij} - ||\mathbf{z}_i - \mathbf{z}_j||)^2$$

$\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ denote low-dim embeddings/projections of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively

- Basically, if d_{ij} is large (resp. small), would like $\|\boldsymbol{z}_i - \boldsymbol{z}_j\|$ to be large (resp. small)
- Multi-dimensional Scaling (MDS) is one such algorithm
- Note: If d_{ij} is the Euclidean distance, MDS is equivalent to PCA

MDS: An Example

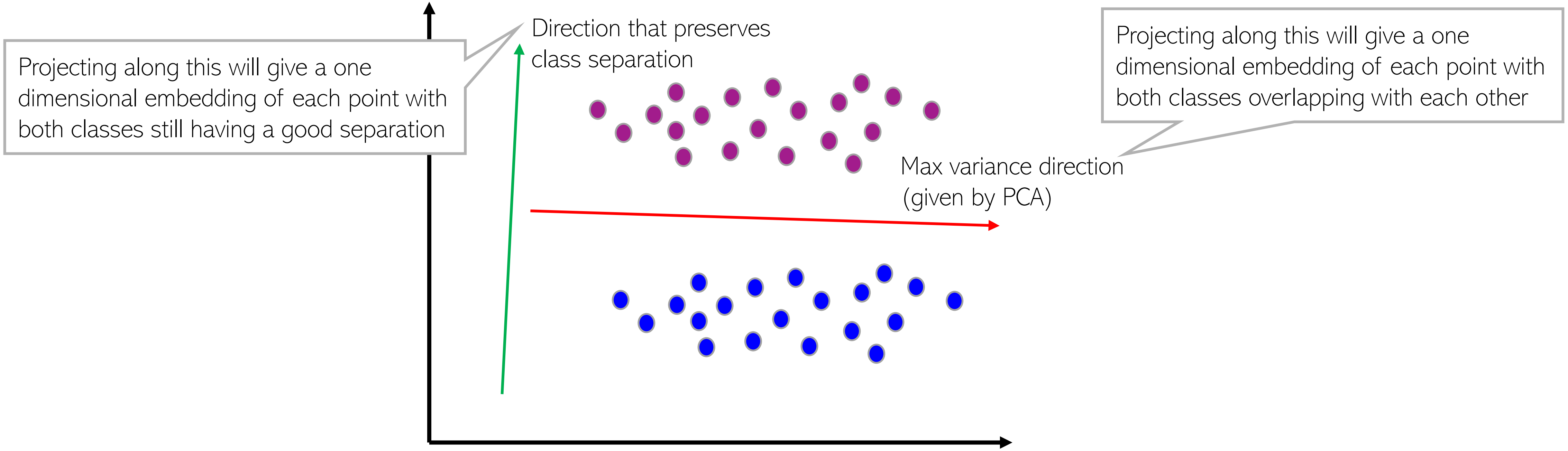
- Result of applying MDS (with $K = 2$) on pairwise distances between some US cities



- Here MDS produces 2D embedding of each city such that geographically close cities are also close in 2D embedding space

Supervised Dimensionality Reduction

- Maximum variance directions may not be aligned with class separation directions (focusing only on variance/reconstruction error of the inputs x_n, is not always ideal)

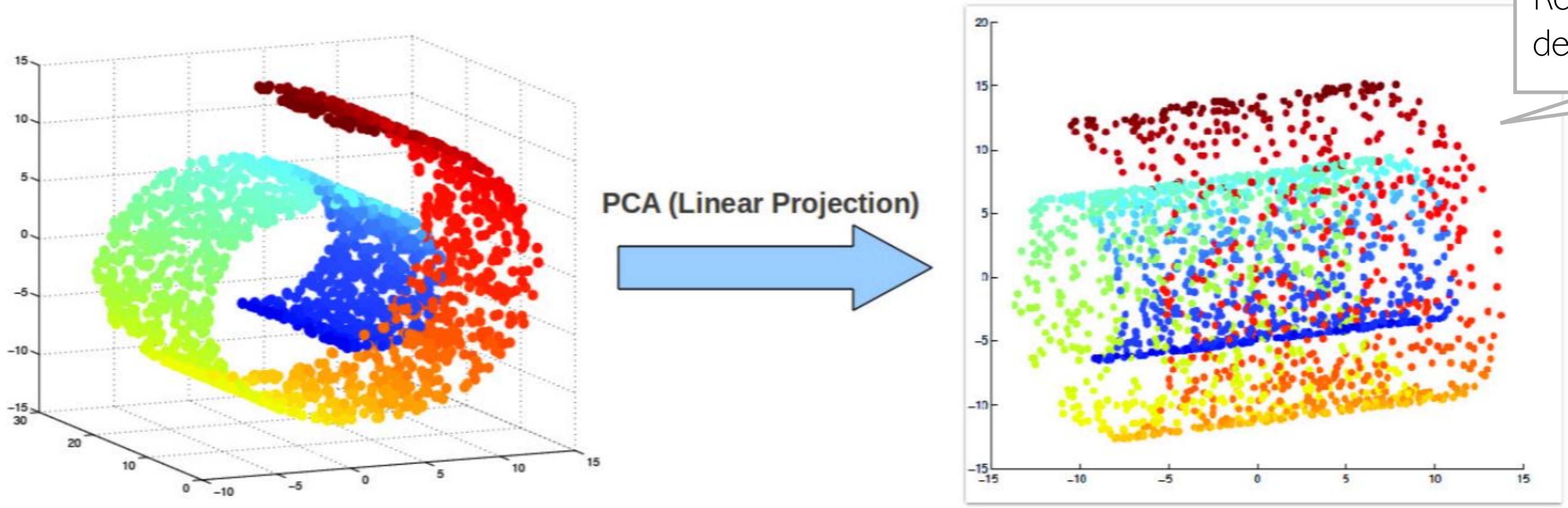


- Be careful when using methods like PCA for supervised learning problems
- A better option would be to find projection directions such that after projection
  - Points within the same class are close (low intra-class variance)
  - Points from different classes are well separated (the class means are far apart)

Nonlinear Dimensionality Reduction

Beyond Linear Projections

- Consider the swiss-roll dataset (points lying close to a manifold)

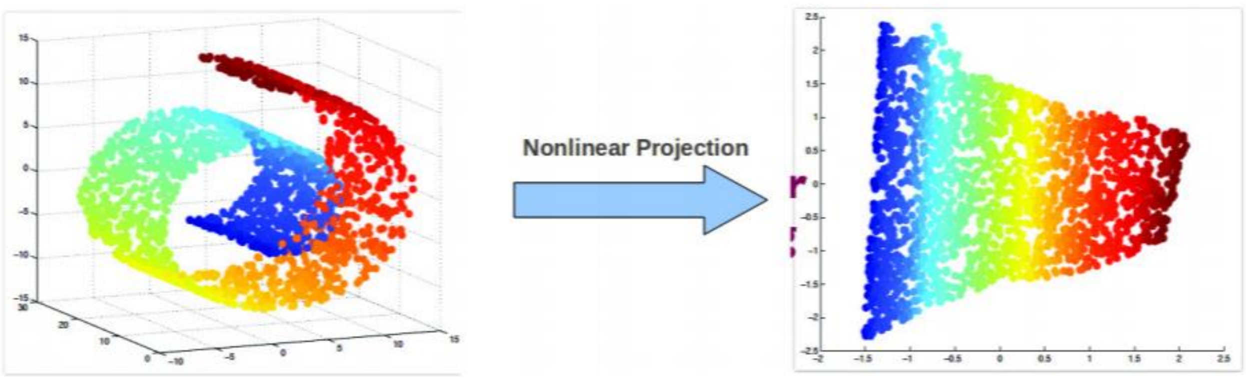


Relative positions of points destroyed after the projection

- Linear projection methods (e.g., PCA) can't capture intrinsic nonlinearities
  - Maximum variance directions may not be the most interesting ones

Nonlinear Dimensionality Reduction

- We want to a learn nonlinear low-dim projection



Relative positions of points preserved after the projection

- Some ways of doing this
  - Nonlinearize a linear dimensionality reduction method. E.g.:
    - Cluster data and apply linear PCA within each cluster (mixture of PCA)
    - Kernel PCA (nonlinear PCA)
  - Using manifold based methods that intrinsically preserve nonlinear geometry, e.g.,
    - Locally Linear Embedding (LLE), Isomap
    - Maximum Variance Unfolding
    - Laplacian Eigenmap, and others such as SNE/tSNE, etc.
- .. or use unsupervised deep learning techniques (later)

Will look at KPCA, LLE, SNE/tSNE

Kernel PCA

- Recall PCA: Given N observations $\boldsymbol{x}_n \in \mathbb{R}^D$, $n = 1,2,\dots,N$,

$D \times D$ cov matrix assuming centered data

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top \qquad \mathbf{S}\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i \ \forall i = 1,\dots,D$$

D eigenvectors of $\mathbf{S}$

- Assume a kernel k with associated M dimensional nonlinear map ϕ

$M \times M$ cov matrix assuming centered data in the kernel-induced feature space

$$\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N} \phi(\mathbf{x}_n)\phi(\mathbf{x}_n)^\top \qquad \mathbf{C}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \ \forall i = 1,\dots,M$$

M eigenvectors of $\mathbf{C}$

- Would like to do it without computing $\mathbf{C}$ and the mappings $\phi(\boldsymbol{x}_n)'s$ since M can be very large (even infinite, e.g., when using an RBF kernel)

- Boils down to doing eigendecomposition of the $N \times N$ kernel matrix $\mathbf{K}$ (PRML 12.3)
  - Can verify that each $\boldsymbol{v}_i$ above can be written as a lin-comb of the inputs: $\boldsymbol{v}_i = \sum_{n=1}^{N} a_{in}\phi(\boldsymbol{x}_n)$
  - Can show that finding $\boldsymbol{a}_i = [a_{i1}, a_{i2}, \dots, a_{iN}]$ reduces to solving an eigendecomposition of $\mathbf{K}$
  - Note: Due to req. of centering, we work with a centered kernel matrix $\tilde{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N\mathbf{K} - \mathbf{K}\mathbf{1}_N + \mathbf{1}_N\mathbf{K}\mathbf{1}_N$

$N \times N$ matrix of all 1s

Locally Linear Embedding

Several non-lin dim-red algos use this idea

Essentially, neighbourhood preservation, but only local

- Basic idea: If two points are local neighbors in the original space then they should be local neighbors in the projected space too
- Given N observations $\boldsymbol{x}_n \in \mathbb{R}^D$, $n = 1,2,\dots,N$, LLE is formulated as

Solve this to learn weights W_{ij} such that each point $\boldsymbol{x}_i$ can be written as a weighted linear combination of its local neighbors in the original feature space

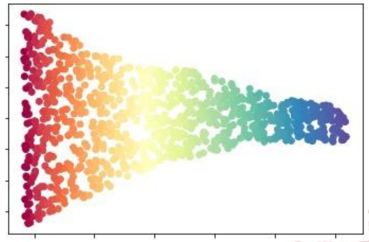
$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \sum_{i=1}^{N} ||\mathbf{x}_i - \sum_{j \in \mathcal{N}(i)} W_{ij}\mathbf{x}_j||^2$$

$\mathcal{N}(i)$ denotes the local neighbors (a predefined number, say K, of them) of point $\boldsymbol{x}_i$

- For each point $\boldsymbol{x}_n \in \mathbb{R}^D$, LLE learns $\boldsymbol{z}_n \in \mathbb{R}^K$, $n = 1,2,\dots,N$ such that the same neighborhood structure exists in low-dim space too

Requires solving an eigenvalue problem

$$\hat{\mathbf{Z}} = \arg\min_{\mathbf{Z}} \sum_{i=1}^{N} ||\mathbf{z}_i - \sum_{j \in \mathcal{N}(i)} W_{ij}\mathbf{z}_j||^2$$



- Basically, if point $\boldsymbol{x}_i$ can be reconstructed from its neighbors in the original space, the same weights W_{ij} should be able to reconstruct $\boldsymbol{z}_i$ in the new space too

S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. Science 290 (2000)

SNE and t-SNE

Thus very useful if we want to visualize some high-dim data in two or three dims

- Also nonlin. dim-red methods, especially suited for projecting to 2D or 3D
- SNE stands for Stochastic Neighbor Embedding (Hinton and Roweis, 2002)
- Uses the idea of preserving probabilistically defined neighborhoods
- SNE, for each point $\boldsymbol{x}_i$, defines the probability of a point $\boldsymbol{x}_j$ being its neighbor as

Neighbor probability in the original space

$$p_{j|i} = \frac{\exp(-||\mathbf{x}_i - \mathbf{x}_j||^2/2\sigma^2)}{\sum_{k\neq i}\exp(-||\mathbf{x}_i - \mathbf{x}_k||^2/2\sigma^2)}$$

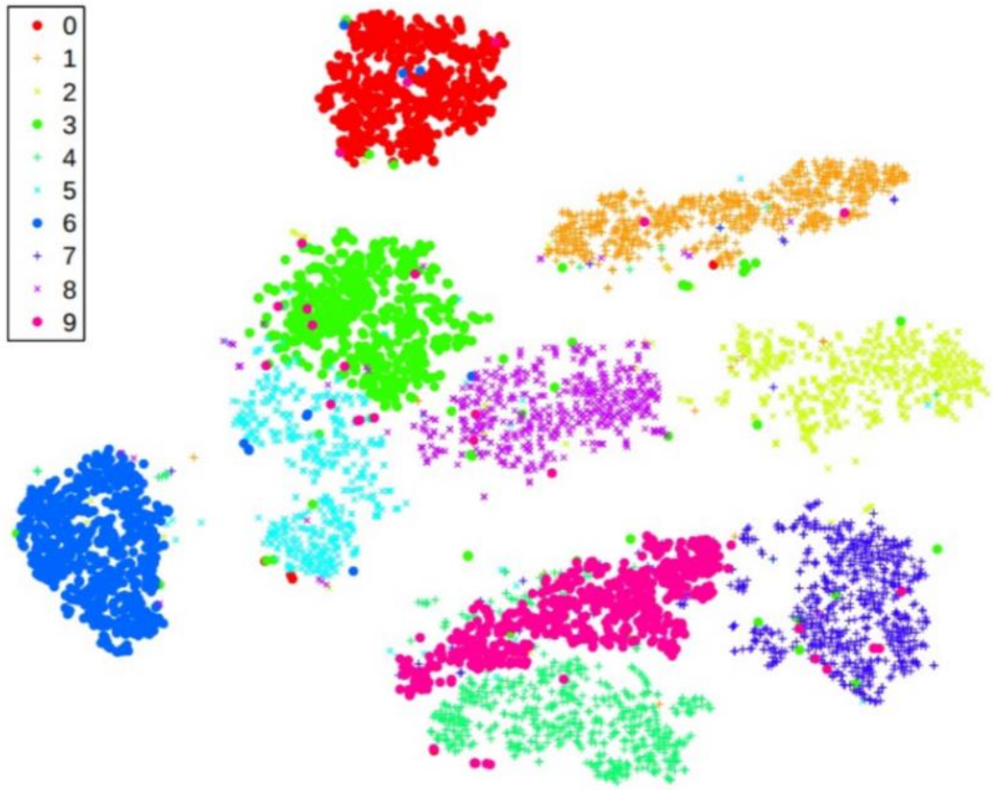
Neighbor probability in the projected/embedding space

$$q_{j|i} = \frac{\exp(-||\mathbf{z}_i - \mathbf{z}_j||^2/2\sigma^2)}{\sum_{k\neq i}\exp(-||\mathbf{z}_i - \mathbf{z}_k||^2/2\sigma^2)}$$

- SNE ensures that neighbourhood distributions in both spaces are as close as possible
  - This is ensured by minimizing their total mismatch (KL divergence) $\mathcal{L} = \sum_{i=1}^{N}\sum_{j=1}^{N} p_{j|i}\log\frac{p_{j|i}}{q_{j|i}}$
- t-SNE (van der Maaten and Hinton, 2008) offers a couple of improvements to SNE
  - Learns $\boldsymbol{z}_i$'s by minimizing symmetric KL divergence
  - Uses Student-t distribution instead of Gaussian for defining $q_{j|i}$

SNE and t-SNE

- Especially useful for visualizing data by projecting into 2D or 3D



Result of visualizing MNIST digits data in 2D (Figure from van der Maaten and Hinton, 2008)

Word Embeddings: Dim-Reduction for Words

Or sentences, paragraphs, documents, etc which are basically a set of words

- Feature representation/embeddings of words are very useful in many applications
- Naively we can a one-hot vector of size V for each word (where V is the vocab size)

1 only at the n-th location

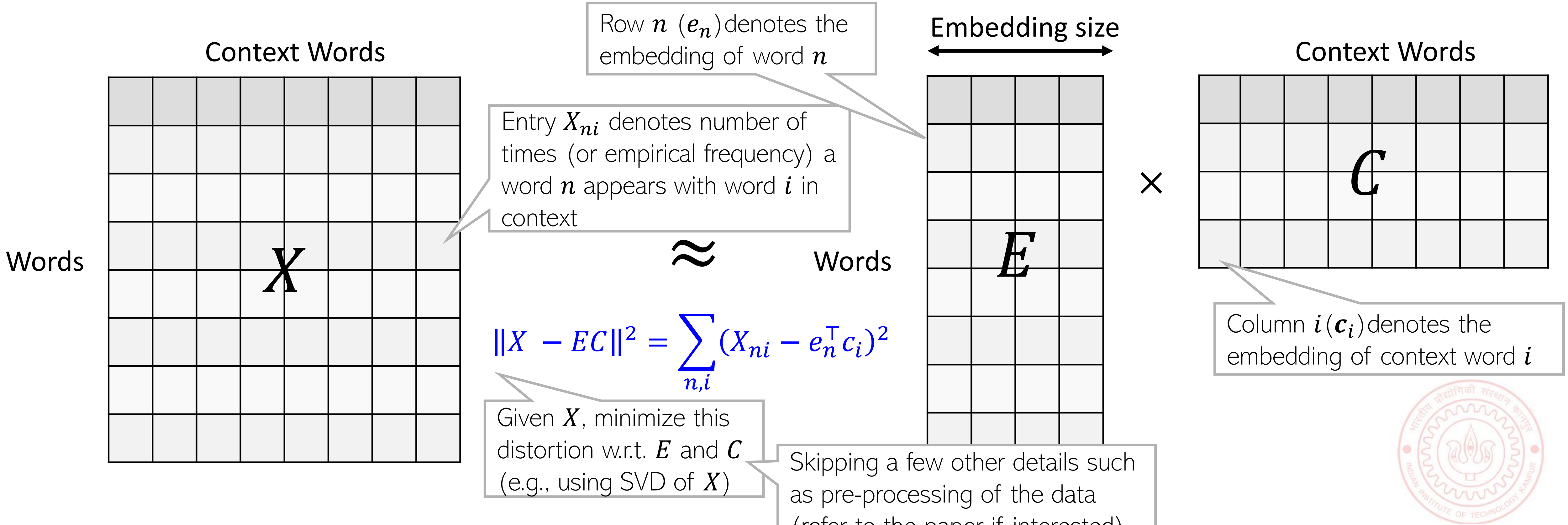
Word n of the vocabulary

0	0	0	0	0	0	....	0	1	....	0	0	0	0

- One-hot representation of a word has two main issues
  - Very high dimensionality (V) for each word
  - One-hot vector does not capture word semantics (any pair of words will have zero similarity)
- Desirable: Learning low-dim word embeddings that capture the meaning/semantics
- We want embedding of each word n to be low-dimensional vector $\boldsymbol{e}_n \in \mathbb{R}^K$
  - If two words n and n' are semantically similar (dissimilar), we want $\boldsymbol{e}_n$ and $\boldsymbol{e}_{n'}$ to be close (far)
- Many methods to learn word embeddings (e.g., Glove and Word2Vec)

GloVe

- GloVe (Global Vectors for Word Representation) is a linear word embedding method
- Based on matrix factorization of a word-word co-occurrence matrix
- Co-occurrence is w.r.t. some "context" (e.g., 2 words before and after a word)



GloVe: Global Vectors for Word Representation (Pennington et al, 2014)

Word2Vec

- A deep neural network based nonlinear word embedding method
- Usually learned using one of the following two objectives
  - Skip-gram
  - Continuous bag of words (CBOW)
- Skip-gram: Probability of a context word $\boldsymbol{i}$ occurring around a word $\boldsymbol{n}$

Conditional probability which can be estimated from training data

$$p(i|n) = \frac{\exp(\boldsymbol{c}_i^\top \boldsymbol{e}_n)}{\sum_i \exp(\boldsymbol{c}_i^\top \boldsymbol{e}_n)}$$

Embeddings are learned by optimizing a neural network based loss function which makes the difference b/w LHS and RHS small

- CBOW: Probability of word $\boldsymbol{n}$ occurring given a context window, e.g., $\boldsymbol{k}$ previous and $\boldsymbol{k}$ next words

Conditional probability which can be estimated from training data

Sum/average of the embeddings of words in the context window for word $\boldsymbol{n}$

$$p(n|n-k:n+k) = \frac{\exp(\boldsymbol{e}_n^\top \boldsymbol{c}_n)}{\sum_n \exp(\boldsymbol{e}_n^\top \boldsymbol{c}_n)}$$

Embeddings are learned by optimizing a neural network based loss function which makes the difference b/w LHS and RHS small

Efficient Estimation of Word Representations in Vector Space (Mikolov et al, 2013)

Dimensionality Reduction: Out-of-sample Embedding

- Some dim-red methods can only compute the embedding of the training data
- Given N training samples $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ they will give their embedding $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N\}$
- However, given a new point $\boldsymbol{x}_*$ (not in the training samples), they can't produce its embedding $\boldsymbol{z}_*$ easily
  - Thus no easy way of getting "out-of-sample" embedding
- Some of the nonlinear dim-red methods like LLE, SNE, KPCA, etc have this limitation
  - Reason: They don't learn an explicit encoder and directly optimize for $\{\boldsymbol{z}_n\}_{n=1}^N$ given $\{\boldsymbol{x}_n\}_{n=1}^N$
  - To get "out-of-sample" embeddings, these methods require some modifications*
- But many other methods do explicitly learn a mapping $\boldsymbol{z} = f(\boldsymbol{x})$ in form of an "encoder" f that can give $\boldsymbol{z}_*$ for any new $\boldsymbol{x}_*$ as well (such methods are more useful)
  - For PCA, the $D \times K$ projection matrix $\mathbf{W}_K$ is this encoder function and $\boldsymbol{z}_* = \mathbf{W}_K^\top \boldsymbol{x}_*$
  - Neural network based autoencoders can also do this (will see them later)

*Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering (Bengio et al, 2003)