CS-UY 4563: Lecture 22
Principal Component Analysis, Semantic Embeddings

NYU Tandon School of Engineering, Prof. Christopher Musco

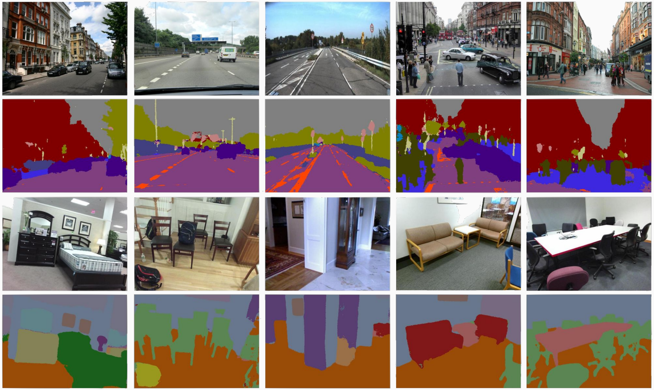
SIDE NOTE ON AUTOENCODER ARCHITECTURES

An autoencoder is a model $f : \mathbb{R}^d \to \mathbb{R}^d$. In other words, the output is the same dimension as the input:

- Image $\to$ Image
- Video $\to$ Video
- Audio clip $\to$ Audio clip

This structure is also useful for some supervised machine learning problems.

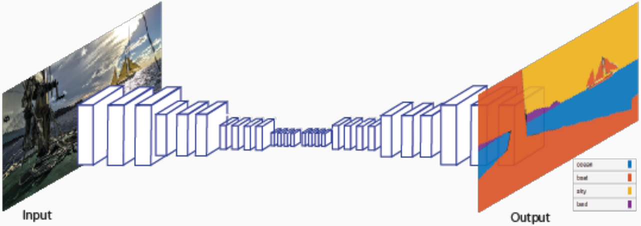
Goal: Learn mask which separates image pixels by what object (foreground or background) that they belong to.



First step in multi-objects classification and scene understanding. Harder than classifying single objects.

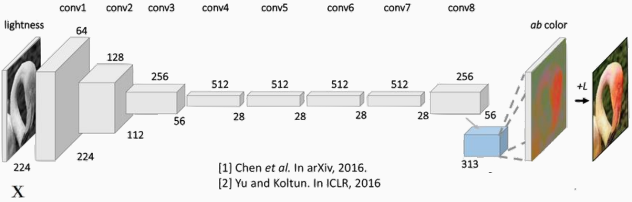
END-TO-END IMAGE SEGMENTATION

Model: Input is image $\vec{x}$, output is image $\vec{m}$ that has the same size as $\vec{x}$, but each pixel value is a label for a segmented region.



Now our training process is actually <u>supervised</u>, but uses the same structure as an autoencoder.

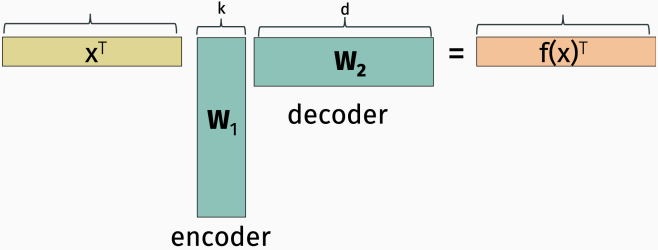
Model: Input is black and white image $\vec{x}$, output is colorized image $\vec{m}$.



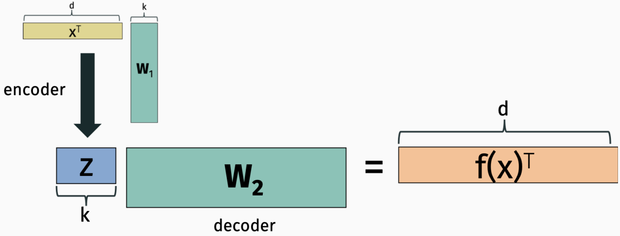
END-TO-END SUPER RESOLUTION

Model: Input is pixelated or blurred image $\vec{x}$, output is full-resolution image $\vec{m}$.

Simple linear autoencoder: Given input $\vec{x} \in \mathbb{R}^d$,



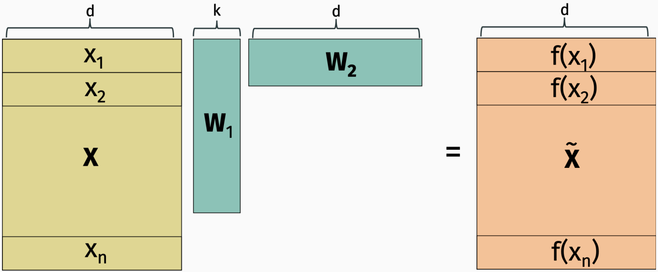
$$f(\vec{x})^T = \vec{x}^T \mathrm{W}_1 \mathrm{W}_2$$



Encoder: $e(\vec{x}) = \vec{x}^T W_1$.

Decoder: $d(\vec{z}) = \vec{z} W_2$

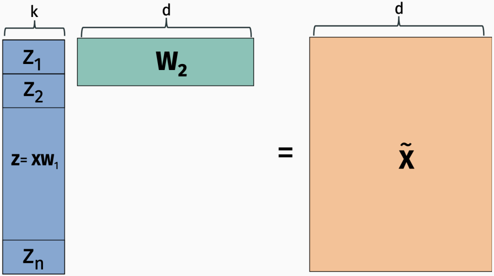
Given training data set $\vec{x}_1, \ldots, \vec{x}_n$, let $\mathbf{X}$ denote our data matrix. Let $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{W}_1\mathbf{W}_2$.



Goal: Find $\mathbf{W}_1, \mathbf{W}_2$ to minimize the Frobenius norm loss $\|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2 = \|\mathbf{X} - \mathbf{X}\mathbf{W}_1\mathbf{W}_2\|_F^2$.

Recall:

- The columns of a matrix with column rank k can all be written as linear combinations of just k columns.
- The rows of a matrix with row rank k can all be written as linear combinations of k rows.
- Column rank = row rank = **rank**.

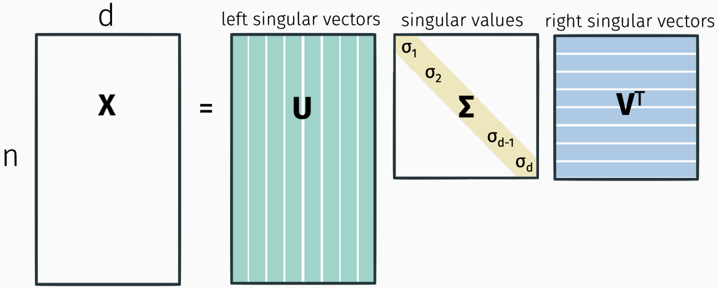


$\tilde{X}$ is a low-rank matrix. It only has rank k for $k \ll d$.

Principal component analysis amounts to finding a rank k matrix $\tilde{\mathrm{X}}$ which approximates the data matrix X as closely as possible.

In general, X will have rank d.

Any matrix **X** can be written:



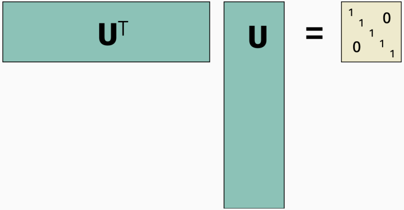
Where $\mathbf{U}^T\mathbf{U} = \mathbf{I}$, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, and $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_d \geq 0$. I.e. **U** and **V** are orthogonal matrices.

This is called the **singular value decomposition.**

Can be computed in $O(nd^2)$ time (faster with approximation algos).

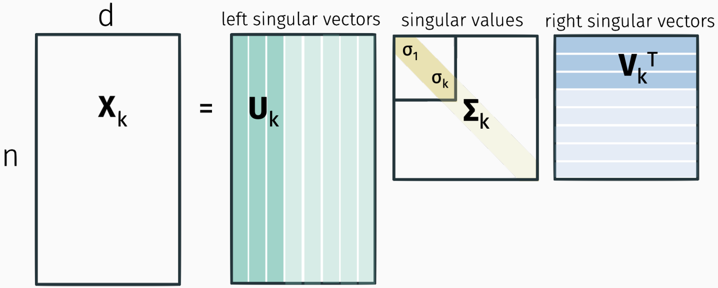
ORTHOGONAL MATRICES

Let $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^n$ denote the columns of $\mathbf{U}$. I.e. the left singular vectors of $\mathbf{X}$.



$\|u_i\|_2^2 =$ $\qquad\qquad$ $\mathbf{u}_i^T \mathbf{u}_j =$

Can read off optimal low-rank approximations from the SVD:



Eckart–Young–Mirsky Theorem: For any $k \leq d$, $\mathbf{X}_k = \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k^T$ is the optimal k rank approximation to $\mathbf{X}$:

$$\mathbf{X}_k = \underset{\tilde{\mathbf{X}} \text{ with rank } \leq k}{\arg\min} \|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2.$$

Claim: $\mathbf{X}_k = \mathbf{U}_k \boldsymbol{\Sigma}_k \mathbf{V}_k^T = \mathbf{X}\mathbf{V}_k\mathbf{V}_k^T$.

So for a model with k hidden variables, we obtain an optimal autoencoder by setting $\mathbf{W}_1 = \mathbf{V}_k$, $\mathbf{W}_2 = \mathbf{V}_k^T$. $f(\vec{x}) = \vec{x}\mathbf{V}_k\mathbf{V}_k^T$.

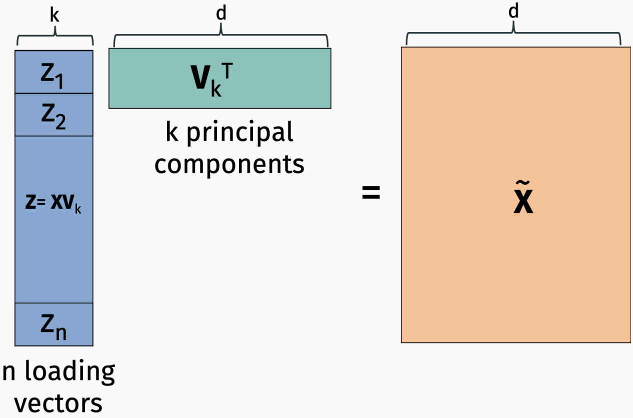
Computing the SVD.

- Full SVD:
  `U,S,V = scipy.linalg.svd(X)`.

  Runs in $O(nd^2)$ time.

- Just the top k components:
  `U,S,V = scipy.sparse.linalg.svds(X, k).`

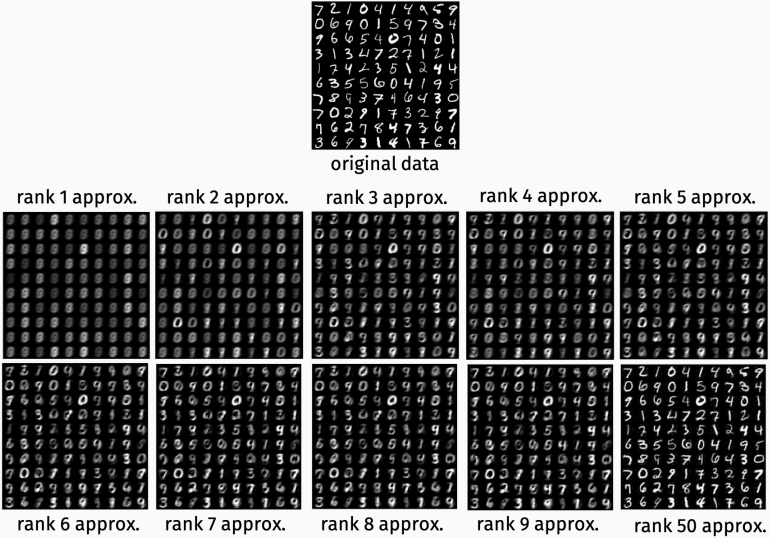
  Runs in roughly $O(ndk)$ time.



Usually $\vec{X}$'s columns (features) are mean centered and normalized to variance 1 before computing principal components.

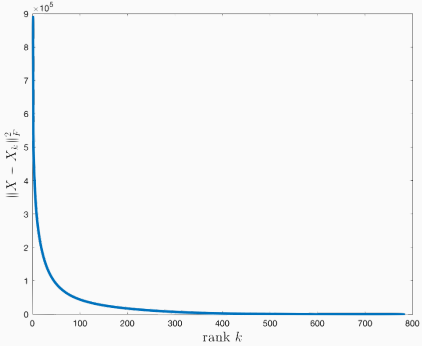
LOW RANK APPROXIMATION

What does recovered data $\tilde{X} = X_k$ look like?



The error can be written as:

$$\|X - X_k\|_F^2 = \sum_{i=k}^{d} \sigma_i^2.$$



What do the **principal components** looks like?

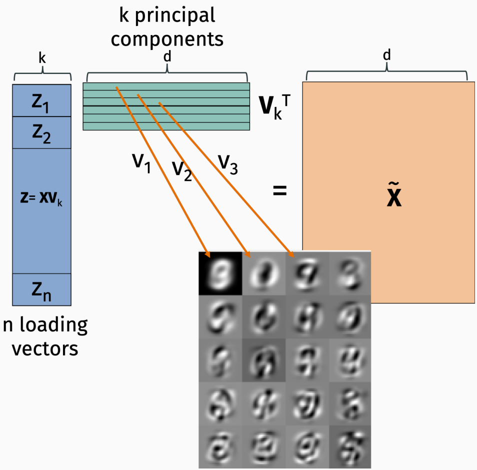
Want a small set of vector $\vec{v}_1, \ldots, \vec{v}_k$ so that most data examples $\vec{x}$ can be written as a linear combination of these basis vectors:

$$\vec{x} \approx c_1\vec{v}_1 + c_2\vec{v}_2 + \ldots + c_k\vec{v}_k$$

One possible basis:

More compact basis:

MNIST **principal components**:

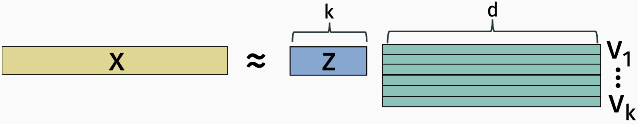


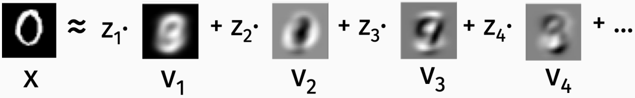
Often principal components are difficult to interpret.

LOADING VECTORS

What do the **loading vectors** looks like?

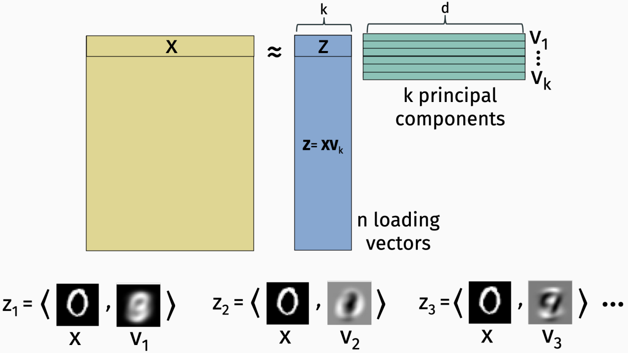
The loading vector $\vec{z}$ for an example $\vec{x}$ contains coefficients which recombine the top k principal components $\vec{v}_1, \ldots, \vec{v}_k$ to approximately reconstruct $\vec{x}$.





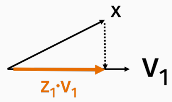
Provide a short "finger print" for any image $\vec{x}$ which can be used to reconstruct that image.

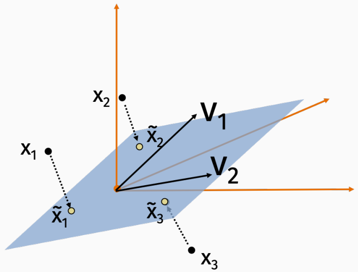
For any $\vec{x}$ with loading vector $\vec{z}$, z_i is the inner product similarity between $\vec{x}$ and the i^{th} principal component $\vec{v}_i$.



LOADING VECTORS: PROJECTION VIEW

So we approximate $\vec{x} \approx \tilde{x} = \langle \vec{x}, \vec{v}_1 \rangle \cdot v_1 + \ldots + \langle \vec{x}, \vec{v}_k \rangle \cdot \vec{v}_k$.

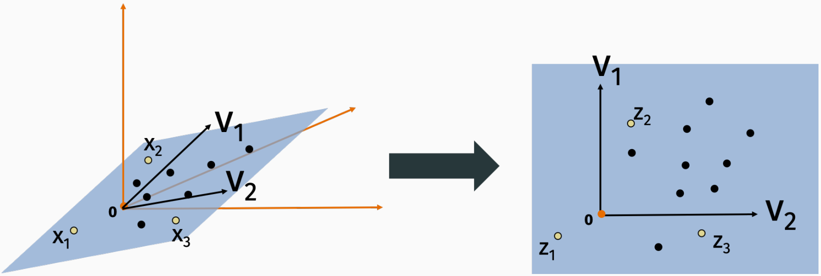




Projection onto first 2 principal components.

Equivalent to projecting $\vec{x}$ onto the k-dimensional subspace spanned by $\vec{v}_1, \ldots, \vec{v}_k$.

For an example $\vec{x}_i$, the loading vector $\vec{z}_i$ contains the coordinates in the projection space:



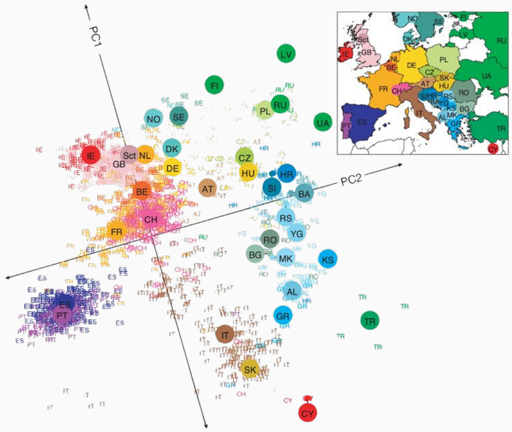
Projection onto first 2 principal components.

Like any autoencoder, PCA can be used for:

- Feature extraction
- Denoising and rectification
- Data generation
- Compression
- Visualization

PCA FOR DATA VISUALIZATION

"Genes Mirror Geography Within Europe" – Nature, 2008.



Each data vector $\mathbf{x}_i$ contains genetic information for one person in Europe. Set $k = 2$ and plot $(XV)_i$ for each i on a 2-d plane. Color points by what country they are from.

Consider data sets which consist of text:

Review 1: *So far this thing is great. It takes up way less space and does a great job opening cans.*

Review 2: *Well designed, compact, and easy to use. I'll never use another can opener.*

Review 3: *Not entirely sure this was worth 20. Mom couldn't figure out how to use it and it's fairly difficult to turn for someone with arthritis.*

Goal is to classify reviews as "positive" or "negative".

Step 1: Need to convert reviews to numerical data.

One approach: Bag-of-words features.

SEMANTIC EMBEDDINGS: MOTIVATING PROBLEM

Vocabulary: Small, handy, excellent, great, quality, compact, easy, difficult.

Review 1: *Very small and handy for traveling or camping. Excellent quality, operation, and appearance..*

[, , , , , , ,]

Review 2: *So far this thing is great. Well designed, compact, and easy to use. I'll never use another can opener.*

[, , , , , , ,]

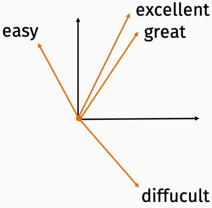
Review 3: *Not entirely sure this was worth* 20. *Mom couldn't figure out how to use it and it's fairly difficult to turn for someone with arthritis.*

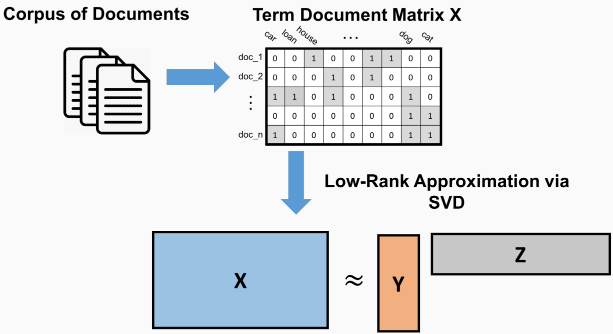
[, , , , , , ,]

This approach only works well for very large data sets.

The algorithm is ignorant to a the fact that “great” and “excellent” are near synonyms. Or that “difficult” and “easy” are antonyms.

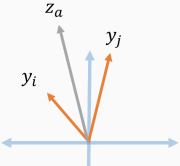
Goal: Map words to numerical vectors in a semantically meaningful way. Similar words map to similar vectors. Dissimilar words to dissimilar vectors.



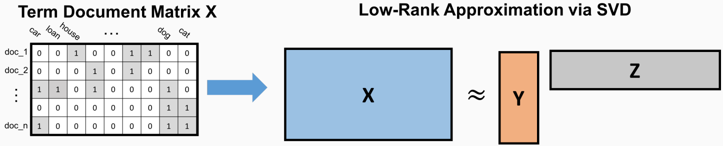


	car	loan	house		…			dog	cat
doc_1	0	0	1	0	0	1	1	0	0
doc_2	0	0	0	1	0	1	0	0	0
⋮	1	1	0	1	0	0	0	1	0
	0	0	0	0	0	0	0	1	1
doc_n	1	0	0	0	0	0	0	1	1

- $\langle \vec{y}_i, \vec{z}_a \rangle \approx 1$ when doc_i contains $word_a$.
- If doc_i and doc_j both contain $word_a$, $\langle \vec{y}_i, \vec{z}_a \rangle \approx \langle \vec{y}_j, \vec{z}_a \rangle = 1$.



EXAMPLE: LATENT SEMANTIC ANALYSIS

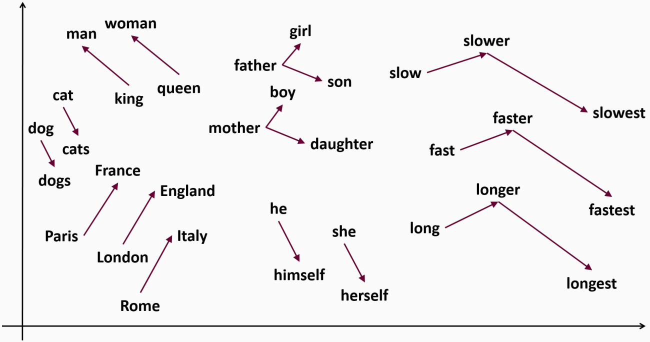


- The columns $\vec{z}_1, \vec{z}_2, \ldots$ give representations of words, with $\vec{z}_i$ and $\vec{z}_j$ tending to have high dot product if $word_i$ and $word_j$ appear in many of the same documents.
- $\mathbf{Z}$ corresponds to the top k right singular vectors: the eigenvectors of $\mathbf{X}\mathbf{X}^T$. Intuitively, what is $\mathbf{X}\mathbf{X}^T$?
- $(\mathbf{X}\mathbf{X}^T)_{i,j} =$

Not obvious how to convert a word into a feature vector that captures the meaning of that word. Approach suggested by LSA: build a $d \times d$ symmetric "similarity matrix" $\mathbf{M}$ between words, and factorize: $\mathbf{M} \approx \mathbf{F}\mathbf{F}^T$ for rank k $\mathbf{F}$.

- **Similarity measures:** How often do $word_i, word_j$ appear in the same sentence, in the same window of w words, in similar positions of documents in different languages?
- Replacing $\mathbf{X}\mathbf{X}^T$ with these different metrics (sometimes appropriately transformed) leads to popular word embedding algorithms: `word2vec`, `GloVe`, etc.

EXAMPLE: WORD EMBEDDING



`word2vec` was originally described as a neural-network method, but Levy and Goldberg show that it is simply low-rank approximation of a specific similarity matrix. *Neural word embedding as implicit matrix factorization.*