

CS-UY 4563: Lecture 22

Principal Component Analysis, Semantic Embeddings

NYU Tandon School of Engineering, Prof. Christopher Musco

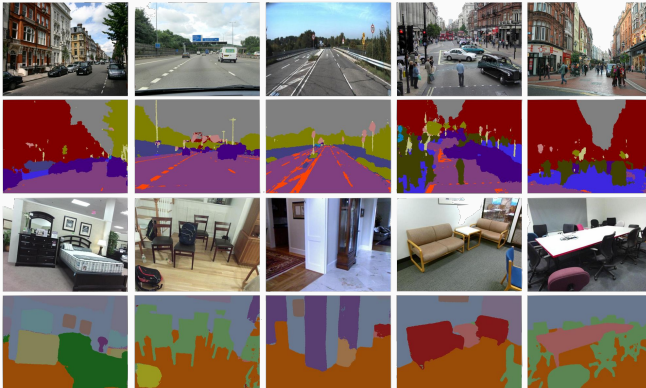
An autoencoder is a model $f: \mathbb{R}^d \rightarrow \mathbb{R}^d$. In other words, the output is the same dimension as the input:

- Image \rightarrow Image
- Video \rightarrow Video
- Audio clip \rightarrow Audio clip

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

IMAGE SEGMENTATION

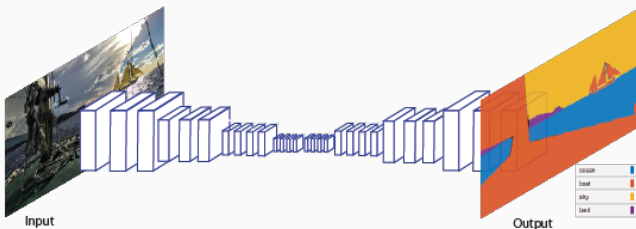
Goal: Learn mask which separates image pixels by what object (foreground or background) that they belong to.



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

END-TO-END IMAGE SEGMENTATION

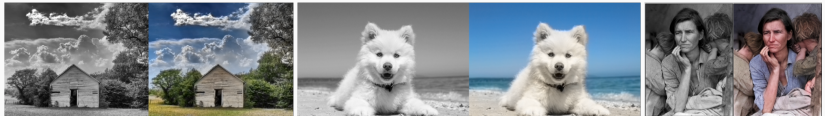
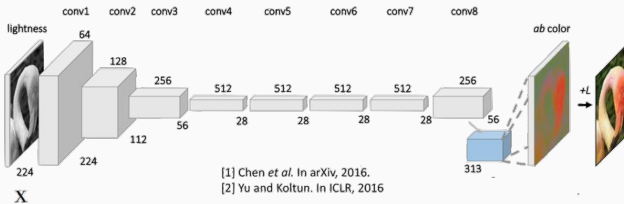
Model: Input is image \vec{x} , output is image \vec{m} that has the same size as \vec{x} , but each pixel value is a label for a segmented region.



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

END-TO-END IMAGE COLORIZATION

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



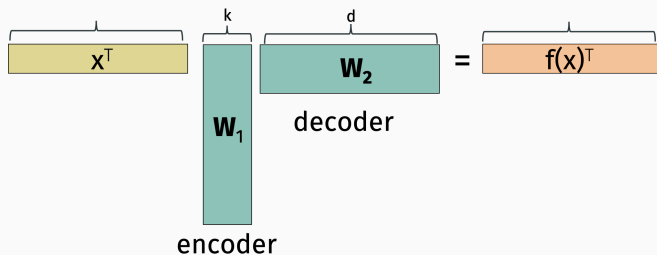
END-TO-END SUPER RESOLUTION

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



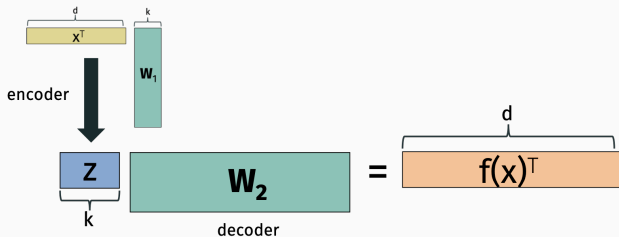
PRINCIPAL COMPONENT ANALYSIS

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



$$f(\vec{x})^T = \vec{x}^T W_1 W_2$$

PRINCIPAL COMPONENT ANALYSIS

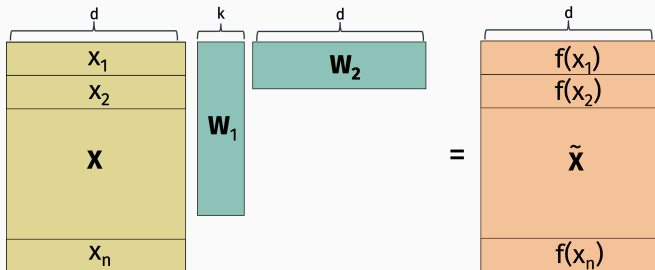


Encoder: $e(\vec{x}) = \vec{x}^T W_1$.

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

PRINCIPAL COMPONENT ANALYSIS

Given training data set $\vec{x}_1, \dots, \vec{x}_n$, 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
 $\|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2 = \|\mathbf{X} - \mathbf{X}\mathbf{W}_1\mathbf{W}_2\|_F^2$.

LOW-RANK APPROXIMATION

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**.

The diagram illustrates the equation $Z = XW_1$ as a low-rank approximation. On the left, a blue vertical rectangle represents matrix Z , with a bracket above it labeled k . It contains the labels z_1 , z_2 , $z = xw_1$, and z_n . In the middle, a green horizontal rectangle represents matrix W_1 , with a bracket above it labeled d and the label w_2 . To the right of W_1 is an equals sign. Further right is an orange vertical rectangle representing matrix \tilde{X} , with a bracket above it labeled d and the label \tilde{x} .

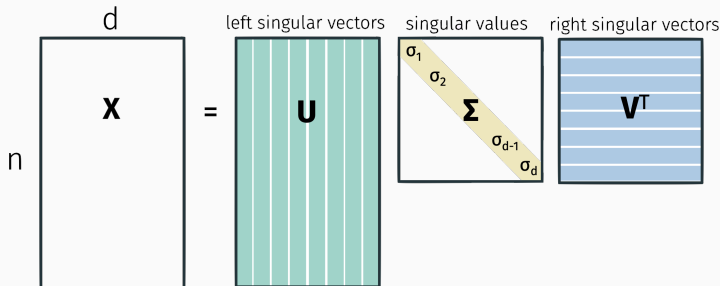
\tilde{X} is a **low-rank matrix**. It only has rank k for $k \ll d$.

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

In general, \mathbf{X} will have rank d .

SINGULAR VALUE DECOMPOSITION

Any matrix \mathbf{X} can be written:



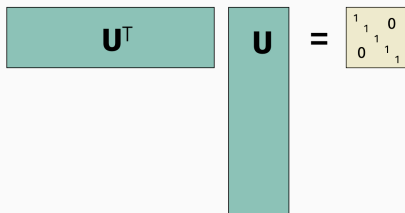
Where $\mathbf{U}^T \mathbf{U} = \mathbf{I}$, $\mathbf{V}^T \mathbf{V} = \mathbf{I}$, and $\sigma_1 \geq \sigma_2 \geq \dots \sigma_d \geq 0$. I.e. \mathbf{U} and \mathbf{V} are orthogonal matrices.

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

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

ORTHOGONAL MATRICES

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



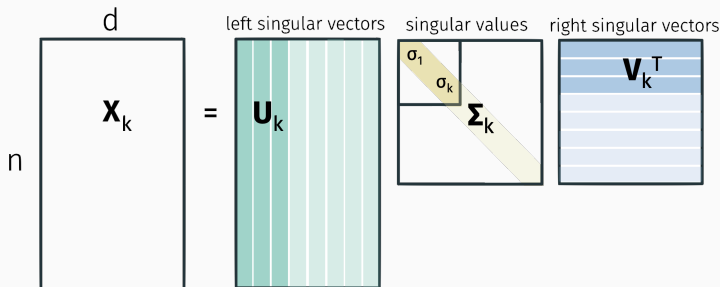
The diagram illustrates the product of two orthogonal matrices. On the left, a teal rectangular box labeled \mathbf{U}^T represents a row vector. To its right is another teal rectangular box labeled \mathbf{U} , representing a column vector. An equals sign follows, leading to a yellow square box containing a 3x3 identity matrix: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

$$\|\mathbf{u}_i\|_2^2 =$$

$$\mathbf{u}_i^T \mathbf{u}_j =$$

SINGULAR VALUE DECOMPOSITION

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



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

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

Claim: $X_k = U_k \Sigma_k V_k^T = X V_k V_k^T$.

So for a model with k hidden variables, we obtain an optimal autoencoder by setting $W_1 = V_k$, $W_2 = V_k^T$. $f(\vec{x}) = \vec{x} V_k V_k^T$.

Computing the SVD.

- Full SVD:

```
U,S,V = scipy.linalg.svd(X).
```

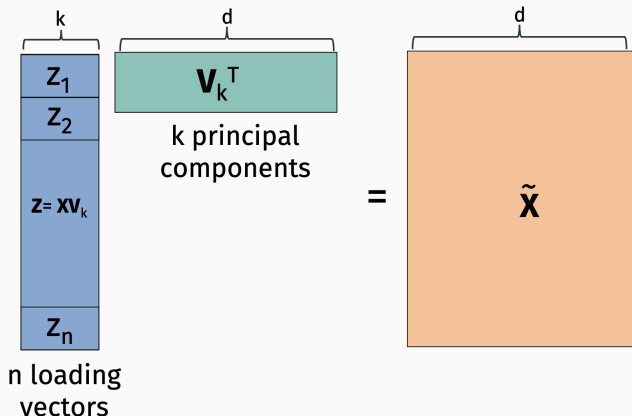
Runs in $O(nd^2)$ time.

- Just the top k components:

```
U,S,V = scipy.sparse.linalg.svds(X, k).
```

Runs in roughly $O(ndk)$ time.

PRINCIPAL COMPONENT ANALYSIS



Usually \tilde{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?

```
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
7 8 9 3 7 4 6 4 3 0
7 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

original data

rank 1 approx.

```
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
8 8 8 8 8 8 8 8 8 8
```

rank 2 approx.

```
0 0 1 0 0 1 9 9 8 8
0 0 9 0 1 0 9 8 8 8
7 0 8 8 8 0 8 8 8 8
8 7 8 0 8 0 8 1 0 1
1 7 7 9 8 9 7 8 0
8 8 8 0 0 0 0 1 0 9
8 8 8 7 8 7 0 8 7 0
8 0 0 7 7 7 8 7 8
8 8 8 8 8 8 7 0 0 1
8 8 7 7 7 8 7 1 0 9
```

rank 3 approx.

```
9 3 7 0 9 1 9 9 8 9
0 0 9 0 1 8 9 7 3 9
9 8 8 8 9 0 9 7 0 1
3 1 3 0 9 0 9 1 3 1
1 7 9 7 3 3 3 9 9
8 3 9 8 0 6 7 1 9
9 2 7 9 3 0 9 3 0
9 0 0 1 1 9 3 7 1 9
9 0 2 9 3 9 5 0 1
3 2 9 3 1 9 1 0 9
```

rank 4 approx.

```
9 3 7 0 9 1 9 9 8 9
0 0 9 0 1 8 9 7 3 9
9 8 8 8 9 0 9 7 0 1
3 1 3 0 9 0 9 1 3 1
1 7 9 7 3 3 3 9 9
8 8 8 0 0 0 0 1 9
9 8 9 3 9 7 0 9 3 0
9 0 0 1 1 9 3 7 1 9
9 0 8 9 9 9 9 0 0 1
3 2 9 3 1 9 1 0 9
```

rank 5 approx.

```
9 3 7 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
9 8 9 3 7 4 6 4 3 0
9 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

rank 6 approx.

```
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
9 8 9 3 7 4 6 4 3 0
9 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

rank 7 approx.

```
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
9 8 9 3 7 4 6 4 3 0
9 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

rank 8 approx.

```
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
9 8 9 3 7 4 6 4 3 0
9 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

rank 9 approx.

```
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
9 8 9 3 7 4 6 4 3 0
9 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

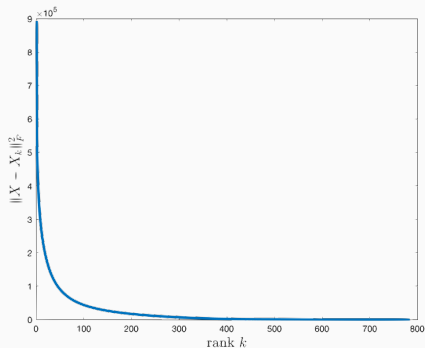
rank 50 approx.

```
7 2 1 0 4 1 4 9 5 9
0 6 9 0 1 5 9 7 8 4
9 6 6 5 4 0 7 4 0 1
3 1 3 4 7 2 7 1 2 1
1 7 4 2 3 5 1 2 4 4
6 3 5 5 6 0 4 1 9 5
9 8 9 3 7 4 6 4 3 0
9 0 2 7 1 7 3 2 7 7
9 6 2 7 8 4 7 3 6 1
3 6 8 3 1 4 1 7 6 9
```

LOW RANK APPROXIMATION

The error can be written as:

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



PRINCIPAL COMPONENTS

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

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

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

One possible basis:

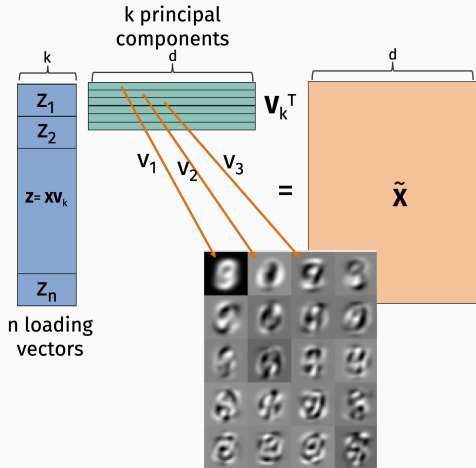


More compact basis:



PRINCIPAL COMPONENTS

MNIST principal components:

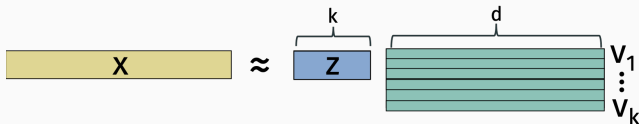


Often principal components are difficult to interpret.

LOADING VECTORS

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

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

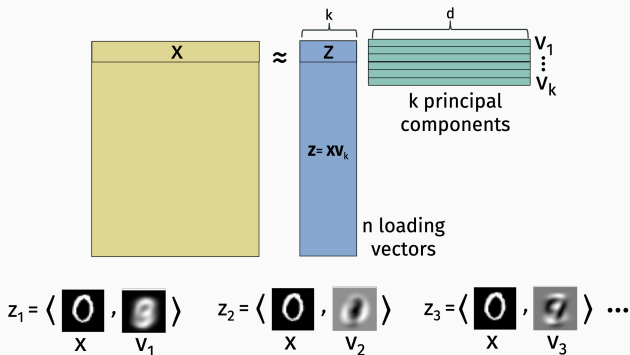


A visual equation showing the reconstruction of a handwritten digit 0 (labeled X) as a sum of weighted principal components. The digit 0 is approximately equal to the sum of z_1 times V_1 , z_2 times V_2 , z_3 times V_3 , z_4 times V_4 , and so on. The principal components V_1, V_2, V_3, V_4 are shown as grayscale images of the digit 0 with increasing levels of detail.

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

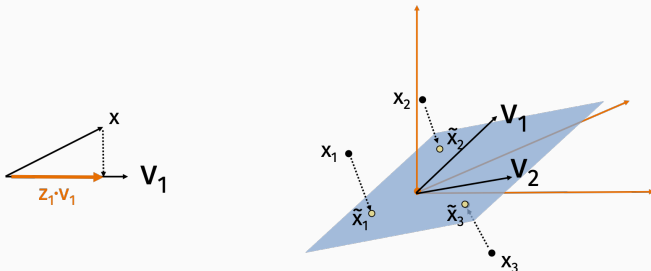
LOADING VECTORS: SIMILARITY VIEW

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 $\vec{x} \approx \tilde{x} = \langle \vec{x}, \vec{v}_1 \rangle \cdot \vec{v}_1 + \dots + \langle \vec{x}, \vec{v}_k \rangle \cdot \vec{v}_k$.

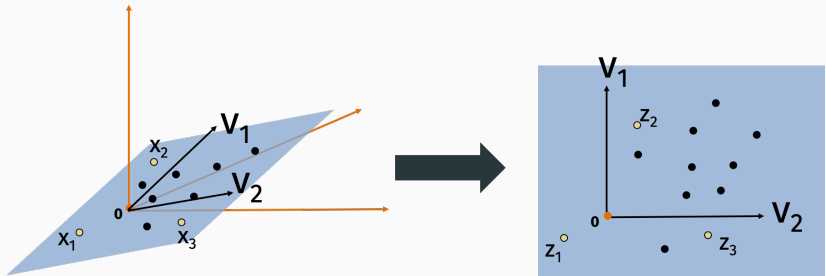


Projection onto first 2 principal components.

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

LOADING VECTORS: PROJECTION VIEW

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



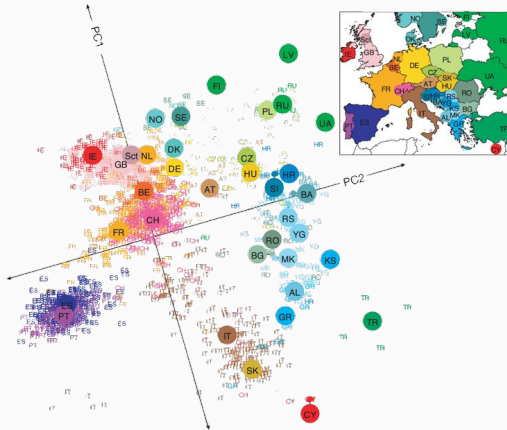
Projection onto first 2 principal components.

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.

SEMANTIC EMBEDDINGS: MOTIVATING PROBLEM

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.*

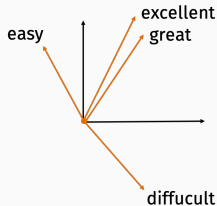
[, , , , , , ,]

SEMANTIC EMBEDDINGS

This approach only works well for very large data sets.

The algorithm is ignorant to 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.



LATENT SEMANTIC ANALYSIS

Corpus of Documents

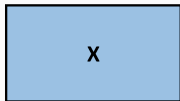


Term Document Matrix X

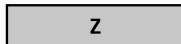
	car	house	...	dog	cat				
doc_1	0	0	1	0	0	1	0	0	
doc_2	0	0	0	1	0	1	0	0	
...	1	1	0	1	0	0	0	1	0
...	0	0	0	0	0	0	0	1	1
doc_n	1	0	0	0	0	0	0	1	1



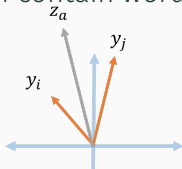
Low-Rank Approximation via
SVD



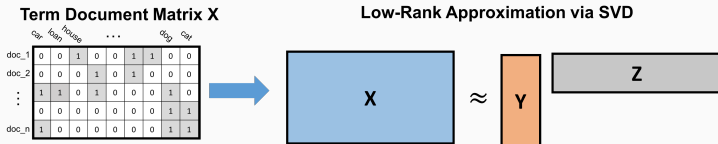
\approx



- $\langle \vec{y}_i, \vec{z}_a \rangle \approx 1$ 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



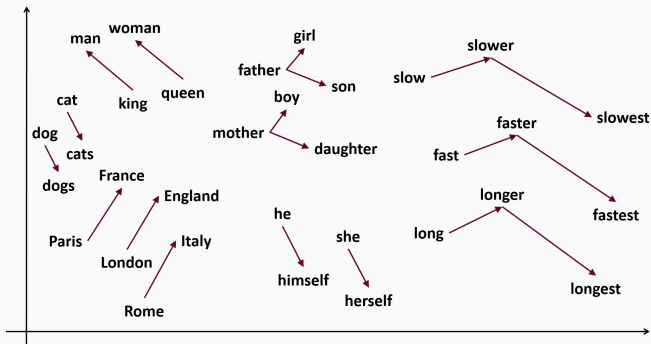
- The columns $\vec{z}_1, \vec{z}_2, \dots$ give representations of words, with \vec{z}_i and \vec{z}_j tending to have high dot product if $word_i$ and $word_j$ appear in many of the same documents.
- Z corresponds to the top k right singular vectors: the eigenvectors of XX^T . Intuitively, what is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

EXAMPLE: WORD EMBEDDING

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.*