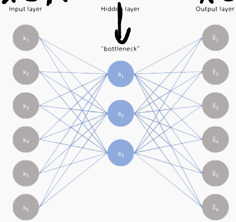
CS-GY 6923: Lecture 12
PCA, Semantic Embeddings, Image Generation

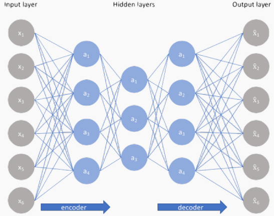
NYU Tandon School of Engineering, Akbar Rafiey
Slides by Prof. Christopher Musco

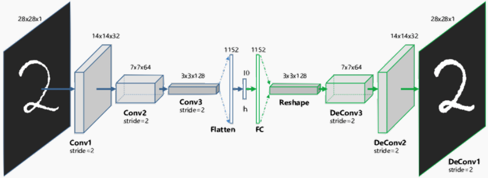
Autoencoder

Recap: Goal of autoencoder models is to map input data to a close approximation of the original that takes less space to represent.

$x \in \mathbb{R}^d$ $\quad k \quad$ $\tilde{x} \in \mathbb{R}^d$ $\quad \min_{\theta} \|x - \tilde{x}\|$

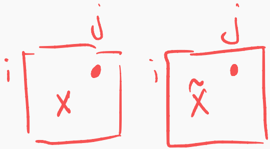






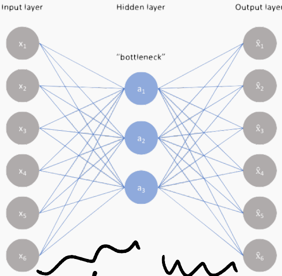
Principal Component Analysis

PCA is the "linear regression" of autoencoders:



$$\sum_{ij} (x_{ij} - \tilde{x}_{ij})^2$$

x



$\tilde{x}$

W_1 W_2

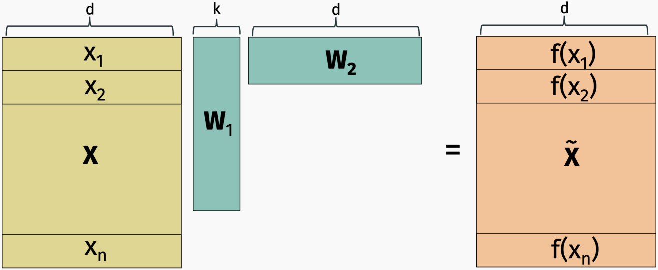
$$\min_{W_1, W_2} \|x - \tilde{x}\|$$

$$\min_{W_1, W_2} \|x - x W_1 W_2\|$$

X X

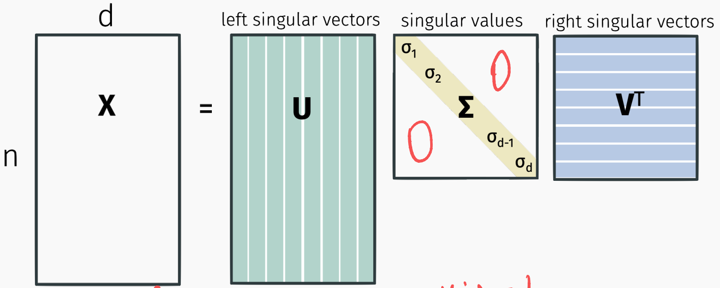
- Simplest possible model. One layer, no non-linearities.
  - $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{W}_1\mathbf{W}_2$ where $\mathbf{X} \in \mathbb{R}^{n\times d}, \mathbf{W}_1 \in \mathbb{R}^{d\times k}, \mathbf{W}_2 \in \mathbb{R}^{k\times d}$.
  - Want to minimize $\min_{\mathbf{W}_1,\mathbf{W}_2} \|\mathbf{X} - \mathbf{X}\mathbf{W}_1\mathbf{W}_2\|_F^2$.
- Equivalent to low-rank approximation. Can be efficiently and provable optimized using the SVD.

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



Singular Value Decomposition

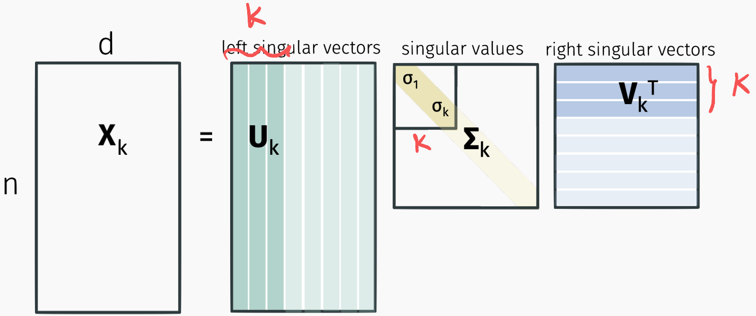
Any matrix **X** can be written:



$\langle u_i, u_j \rangle = 0$ if $i \neq j$; $\langle u_i, u_i \rangle = 1$

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

Partial Singular Value Decomposition

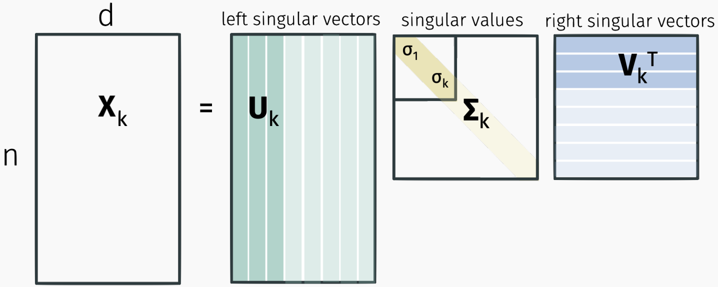


Can be computed in roughly $O(ndk)$ time.

$k \leq d$

Singular value decomposition

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



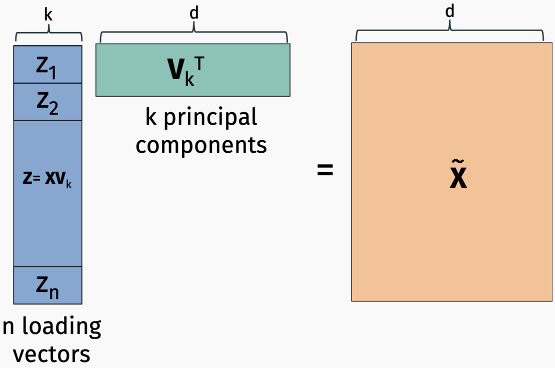
Eckart–Young–Mirsky Theorem: For any $k \leq d$,

$\mathbf{X}_k = \mathbf{U}_k \boldsymbol{\Sigma}_k \mathbf{V}_k^T$ is the optimal k rank approximation to $\mathbf{X}$:

$= X V_k V_k^T$

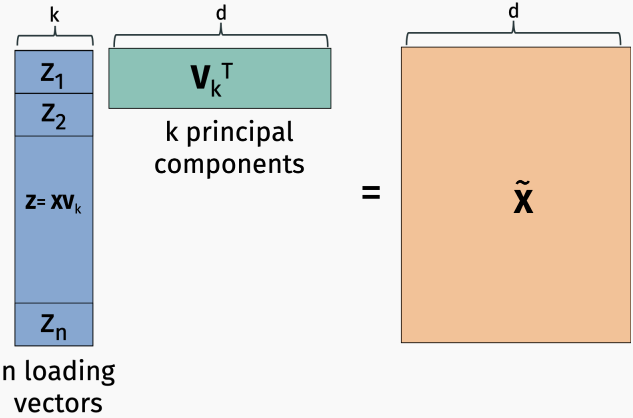
$$\mathbf{X}_k = \underset{\tilde{\mathbf{X}} \text{ with rank } \leq k}{\arg\min} \|\mathbf{X} - \tilde{\mathbf{X}}\|_F^2.$$

Principal Component Analysis



Eckart–Young–Mirsky Theorem: $\tilde{\mathbf{X}} = \mathbf{X}\mathbf{V}_k\mathbf{V}_k^T$ is the optimal low-rank approximation to $\mathbf{X}$. So $\mathbf{W}_1 = \mathbf{V}_k$ and $\mathbf{W}_2 = \mathbf{V}_k^T$ are optimal autoencoder parameters.

Principal Component Analysis (PCA)



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

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.

Connection to eigen-decomposition

Recall that for a matrix $\mathbf{M} \in \mathbb{R}^{p \times p}$, $\mathbf{q}$ is an eigenvector of $\mathbf{M}$ if $\lambda \mathbf{q} = \mathbf{M}\mathbf{q}$ for a scalar λ.

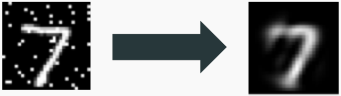
- $\mathbf{U}$'s columns (the left singular vectors) are the orthonormal eigenvectors of $\mathbf{X}\mathbf{X}^T$.
- $\mathbf{V}$'s columns (the right singular vectors) are the orthonormal eigenvectors of $\mathbf{X}^T\mathbf{X}$.
- $\sigma_i^2 = \lambda_i(\mathbf{X}\mathbf{X}^T) = \lambda_i(\mathbf{X}^T\mathbf{X})$

Exercise: Verify this directly. This means you can use any eigensolver for computing the SVD.

PCA applications

Like any autoencoder, PCA can be used for:

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

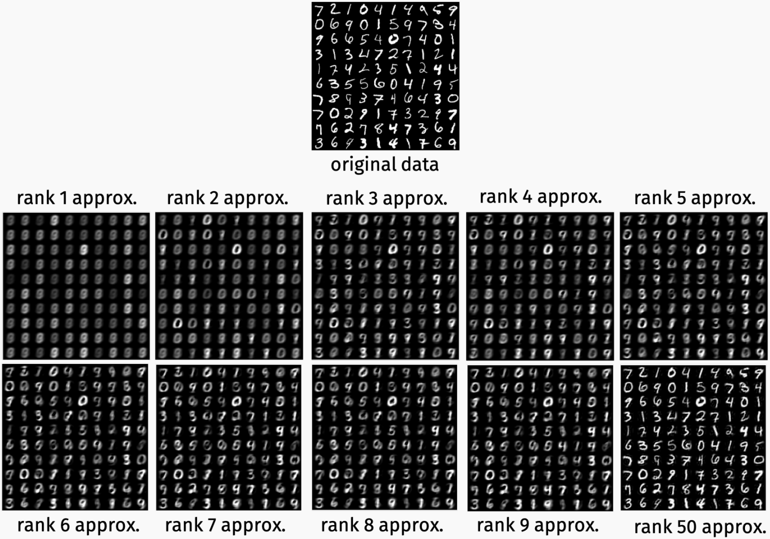


denoising

synthetic data generation

Low-rank approximation

The larger we set k, the better approximation we get.



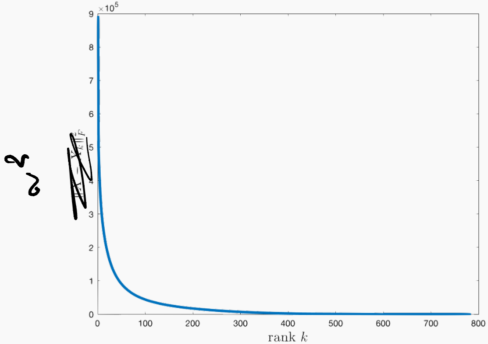
Low rank approximation

Error vs. k is dictated by $\mathbf{X}$'s singular values. The singular values are often called the **spectrum** of $\mathbf{X}$.

$$\|\mathbf{X} - \mathbf{X}_k\|_F^2 = \sum_{i=k+1}^{d} \sigma_i^2.$$

$$\begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_d \end{bmatrix} = \Sigma$$

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$$



Column redundancy

Colinearity of data features leads to an approximately low-rank data matrix.

$d-1$ $\quad$ d

	bedrooms	bathrooms	sq.ft.	floors	list price	sale price
home 1	2	2	1800	2	200,000	195,000
home 2	4	2.5	2700	1	300,000	310,000
⋮	⋮	⋮	⋮	⋮	⋮	⋮
home n	5	3.5	3600	3	450,000	450,000

removing $d-1$ column $\Longrightarrow$ lower rank data matrix

sale price $\approx 1.05 \cdot$ list price.

property tax $\approx .01 \cdot$ list price.

Column redundancy

Sometimes these relationships are simple, other times more complex. But as long as there exists linear relationships between features, we will have a lower rank matrix.

$$\textbf{yard size} \approx \textbf{lot size} - \frac{1}{2} \cdot \textbf{square footage}.$$

$$\begin{aligned}\textbf{cumulative GPA} \approx & \frac{1}{4} \cdot \textbf{year 1 GPA} + \frac{1}{4} \cdot \textbf{year 2 GPA} \\ & + \frac{1}{4} \cdot \textbf{year 3 GPA} + \frac{1}{4} \cdot \textbf{year 4 GPA}.\end{aligned}$$

Low-rank intuition

Two other examples of data with good low-rank approximations:

1. Genetic data:

	single nucleotide polymorphisms (SNPs) loci				
	144	312	436	800	943
individual 1	A	T	T	C	G
individual 2	T	G	G	C	C
...					
individual n	C	A	T	A	G

2. "Term-document" matrix with bag-of-words data:

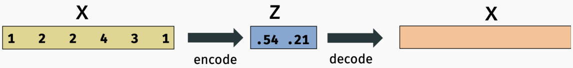
	car	loan	house	…				dog	cat
doc_1	0	0	1	0	0	1	1	0	0
doc_2	0	0	0	1	0	1	0	0	0
⋮	1	1	0	1	0	0	0	1	0
	0	0	0	0	0	0	0	1	1
doc_n	1	0	0	0	0	0	0	1	1

Examples of low-rank structure

SNPs matrices tend to be very low-rank.

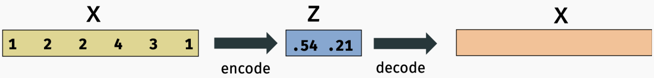
	single nucleotide polymorphisms (SNPs) loci				
	144	312	436	800	943
individual 1	A	T	T	C	G
individual 2	T	G	G	C	C
...					
individual n	C	A	T	A	G

Most of the information in $\mathbf{x}$ is explained by just a few **latent variable**.



Examples of low-rank structure

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



In data collected from European populations, latent variables capture information about geography.

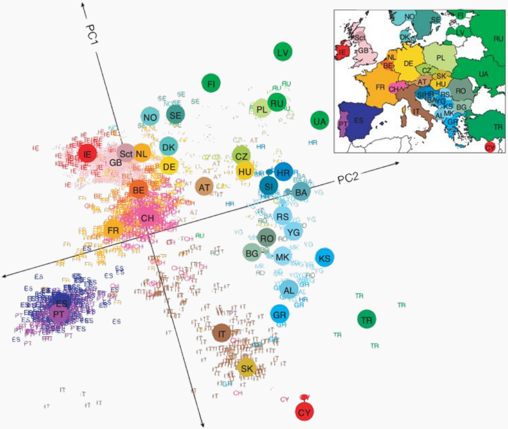
$\mathbf{z}[1] \approx$ relative north-south position of birth place

$\mathbf{z}[2] \approx$ relative east-west position of birth place

Individuals born in similar places tend to have similar genes.

PCA for data visualization

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



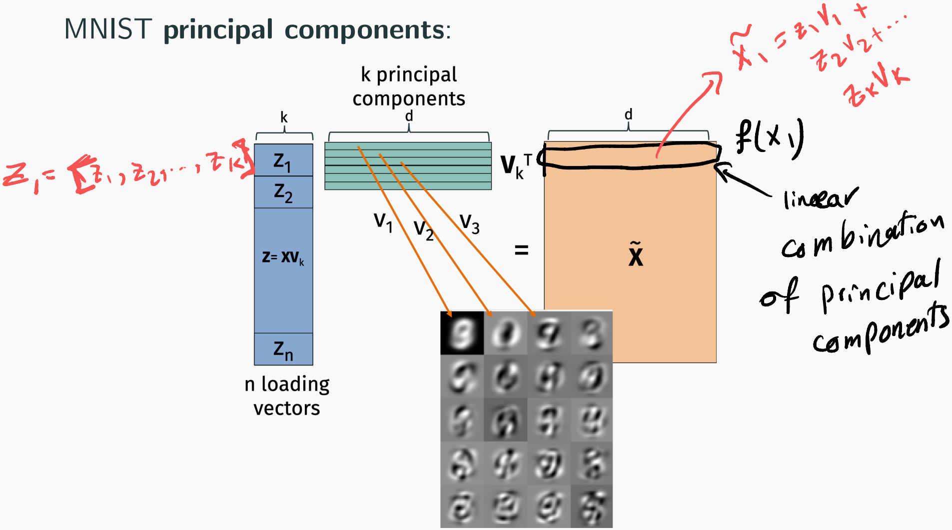
Genetic data can be nicely visualized using PCA! Plot each data example **x** using two loading variables in **z**.

Principal components

For more complex data, what do principal components and loading vectors look like?

Principal Components

MNIST **principal components**:

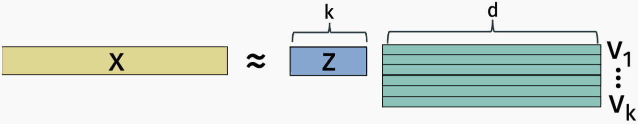


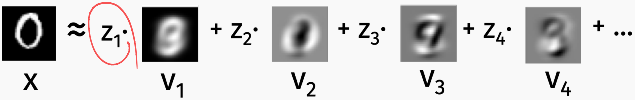
Principal components are a small set of vectors that can be recombined to approximate rows in $\tilde{\mathbf{X}}$.

Loading vectors

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

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



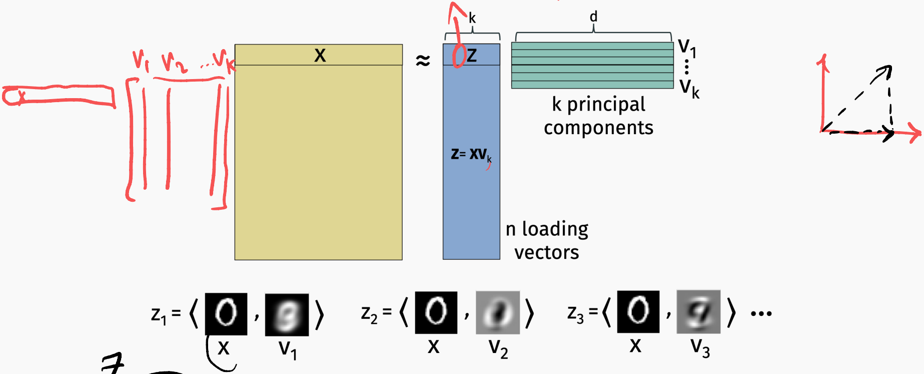


Provide a short "finger print" for any image $\mathbf{x}$ which can be used to reconstruct that image.

Loading vectors: similarity view

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

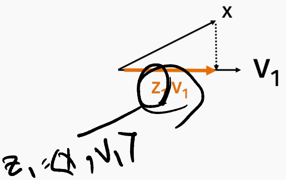
$z_i = \langle x, v_i \rangle$

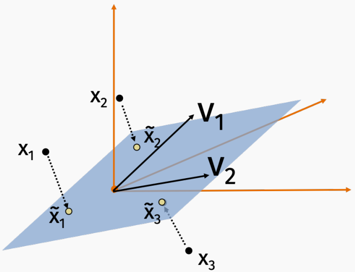


$$X \approx \underbrace{X V_k}_{Z} V_k^T$$

Loading vectors: projection view

So we approximate $\mathbf{x} \approx \tilde{\mathbf{x}} = \overset{z_1}{\langle \mathbf{x}, \mathbf{v}_1 \rangle} \cdot \mathbf{v}_1 + \ldots + \overset{z_k}{\langle \mathbf{x}, \mathbf{v}_k \rangle} \cdot \mathbf{v}_k$.



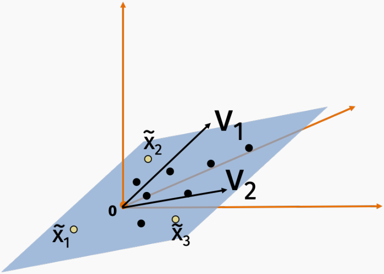


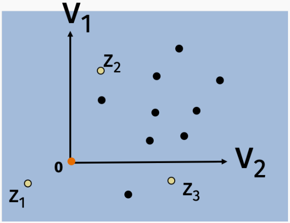
Since $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are orthonormal, this operation is a **projection** onto first k principal components.

I.e. we are projecting $\mathbf{x}$ onto the k-dimensional subspace spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_k$.

Loading vectors: projection view

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





Similarity preservation

Important takeaway for data visualization and more: Latent feature vectors preserve similarity and distance information in the original data.

Let $\mathbf{x}_1 \dots, \mathbf{x}_n \in \mathbb{R}^d$ be our original data vectors, $\mathbf{z}_1 \dots, \mathbf{z}_n \in \mathbb{R}^k$ be our loading vectors (encoding), and $\tilde{\mathbf{x}}_1 \dots, \tilde{\mathbf{x}}_n \in \mathbb{R}^d$ be our low-rank approximated data.

We have:

$\tilde{x}_i = \langle x_i, v_1 \rangle v_1 + \langle x_i, v_2 \rangle \cdot v_2 + \cdots$

$z_i = [\langle x_i, v_1 \rangle, \langle x_i, v_2 \rangle \cdots$

$x_i \approx \tilde{x}_i$

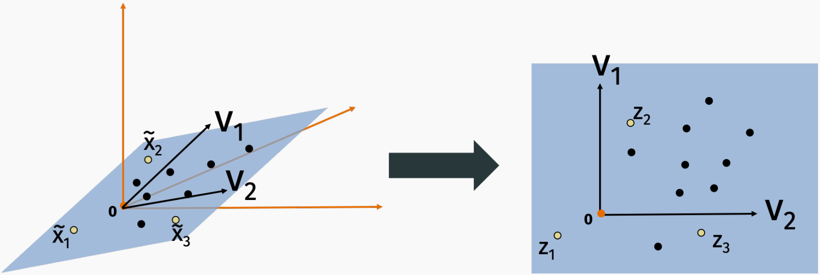
$$\|\tilde{\mathbf{x}}_i\|_2^2 = \|\mathbf{z}_i\|_2^2$$

$$\langle x_i, x_j \rangle \approx \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle = \langle \mathbf{z}_i, \mathbf{z}_j \rangle$$

$\|v_i\| = 1$

$$\|x_i - x_j\|_2^2 \approx \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2^2 = \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$$

PCA preserves geometry of input data



$\langle x_1, x_2 \rangle \approx \langle z_1, z_2 \rangle$

$$\|\mathbf{x}_i\|_2^2 \approx \|\mathbf{z}_i\|_2^2$$

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle \approx \langle \mathbf{z}_i, \mathbf{z}_j \rangle$$

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \approx \|\mathbf{z}_i - \mathbf{z}_j\|_2^2$$

Term document matrix

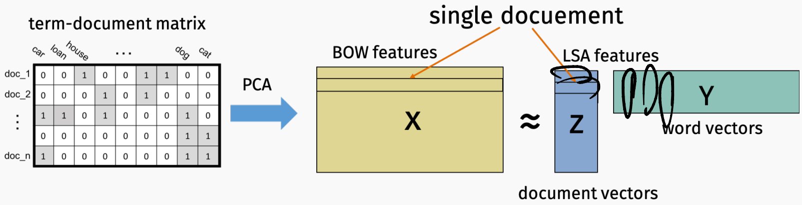
Word-document matrices tend to be low rank.

	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

Documents tend to fall into a relatively small number of different categories, which use similar sets of words:

- **Financial news:** *markets, analysts, dow, rates, stocks*
- **US Politics:** *president, senate, pass, slams, twitter, media*
- **StackOverflow posts:** *python, help, convert, javascript*

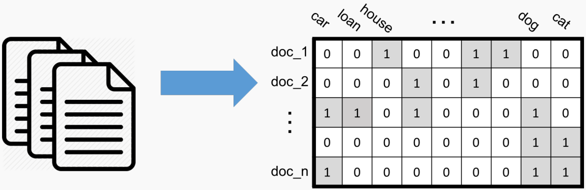
Latent semantic analysis = PCA applied to a word-document matrix (usually from a large corpus). One of the most fundamental techniques in **natural language processing** (NLP).

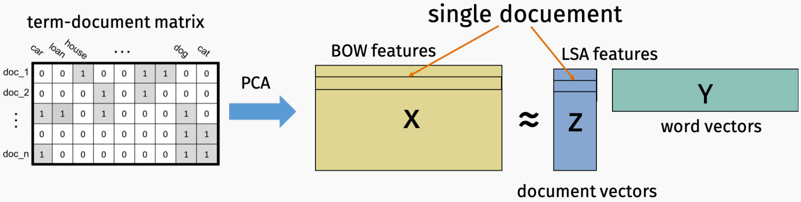


Each column of **z** corresponds to a latent "category" or "topic". Corresponding row in **Y** corresponds to the "frequency" with which different words appear in documents on that topic.

Latent Semantic Analysis (LSA)

Word-document matrix:





For documents with a lot of shared words, $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is a large positive number.

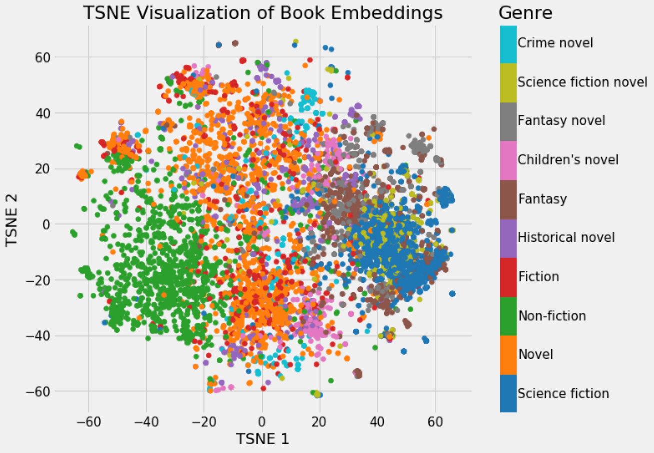
Latent semantic analysis

Similar documents have similar LSA document vectors. I.e. $\langle \mathbf{z}_i, \mathbf{z}_j \rangle$ is large.

- $\mathbf{z}_i$ provides a more compact "finger print" for documents than the long bag-of-words vectors. Useful for e.g search engines.
- Comparing document vectors is often more effective than comparing raw BOW features. Two documents can have $\langle \mathbf{z}_i, \mathbf{z}_j \rangle$ large even if they have no overlap in words. E.g. because both share a lot of words with words with another document k, or with a bunch of other documents.

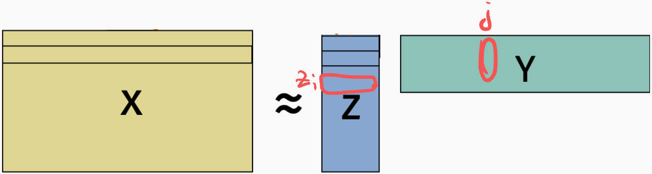
Document embeddings

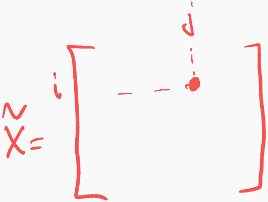
For similar documents, $\langle \mathbf{z}_i, \mathbf{z}_j \rangle$ should be large. I.e. $\mathbf{z}_i$ and $\mathbf{z}_j$ point in the same direction.

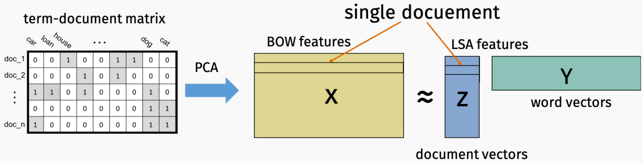


$= ZY$

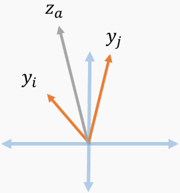
Simple but useful observation: The i, j entry of $\tilde{\mathbf{X}}$ equals $\langle \mathbf{z}_i, \mathbf{y}_j \rangle$.







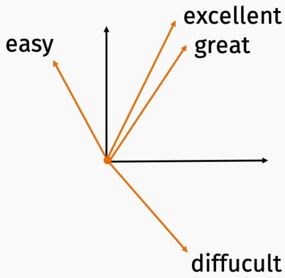
- $\langle \mathbf{y}_i, \mathbf{z}_a \rangle \approx 1$ when doc_a contains $word_i$.
- If $word_i$ and $word_j$ both appear in doc_a, then $\langle \mathbf{y}_i, \mathbf{z}_a \rangle \approx \langle \mathbf{y}_j, \mathbf{z}_a \rangle \approx 1$, so we expect $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ to be large.



If two words appear in the same document their, word vectors tend to point more in the same direction.

Word embeddings

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

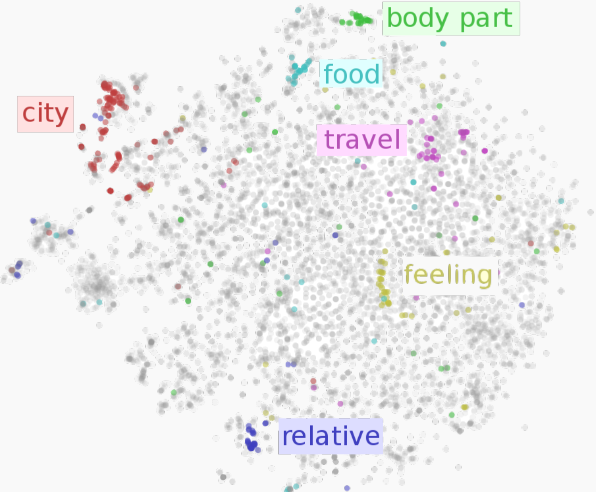


Extremely useful "side-effect" of LSA.

Capture e.g. the fact that "great" and "excellent" are near synonyms. Or that "difficult" and "easy" are antonyms.

Word embeddings

For similar words, $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ should be large. I.e. $\mathbf{y}_i$ and $\mathbf{y}_j$ point in the same direction.



Word embeddings: motivating problem

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.*

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

Bag-of-words features

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

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

$< r_1, r_2 > = 0$

$r_1 = [\,1\,,\,1\,,\,1\,,\,0\,,\,1\,,\,0\,,\,0\,,\,0\,]$

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

$r_2 = [\,0\,,\,0\,,\,0\,,\,1\,,\,0\,,\,1\,,\,1\,,\,0\,]$

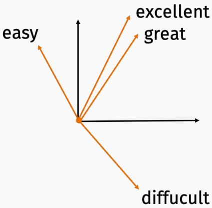
Review 3: *Not entirely sure this was worth \$20. Mom couldn't figure out how to use it and it's fairly difficult to turn for someone with arthritis.*

[, , , , , , ,]

Semantic embeddings

Bag-of-words approach typically only works for large data sets.

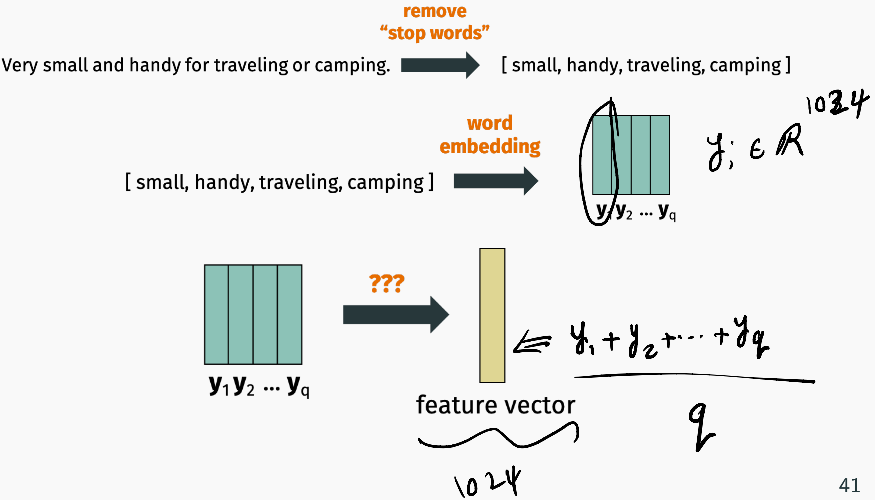
The features do not capture the fact that “great” and “excellent” are near synonyms. Or that “difficult” and “easy” are antonyms.



This can be addressed by first mapping words to semantically meaningful vectors. That mapping can be trained using a much large corpus of text than the data set you are working with (e.g. Wikipedia, Twitter, news data sets).

Using word embeddings

How to go from word embeddings to features for a whole sentence or chunk of text?



$y_i \in \mathbb{R}^{1024}$

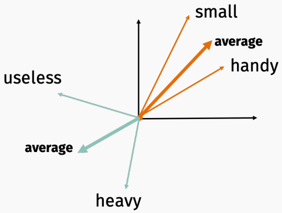
$\Leftarrow \dfrac{y_1 + y_2 + \cdots + y_q}{q}$

$\underbrace{\text{feature vector}}_{1024}$

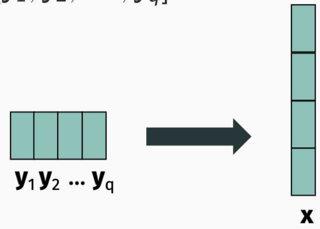
Using word embeddings

A few simple options:

Feature vector $\mathbf{x} = \frac{1}{q}\sum_{i=1}^{q} \mathbf{y}_q$.

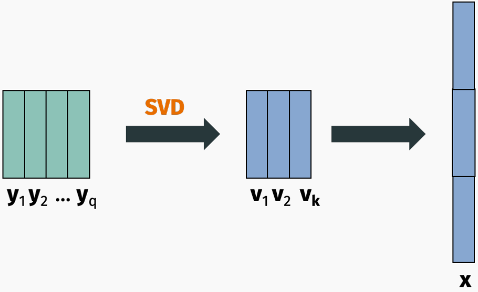


Feature vector $\mathbf{x} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_q]$.



Using word embeddings

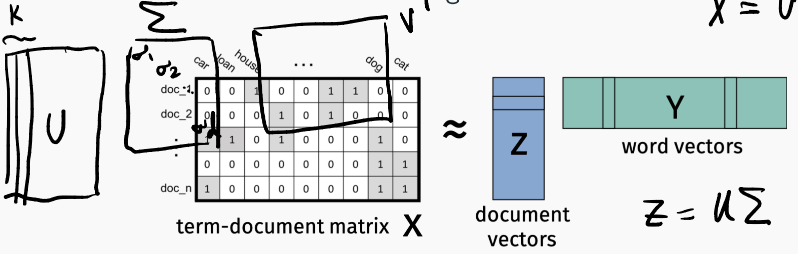
To avoid issues with inconsistent sentence length, word ordering, etc., can concatenate a fixed number of top principal components of the matrix of word vectors:



There are much more complicated approaches that account for word position in a sentence. Lots of pretrained libraries available (e.g. Facebook's `InferSent`).

Another view on word embeddings from LSA:

$X = U \Sigma V^T$



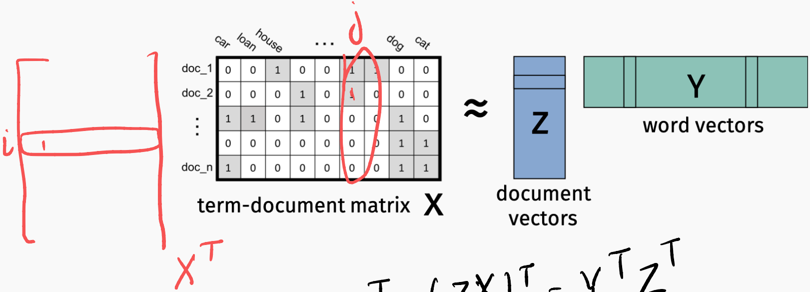
We chose $\mathbf{Z}$ to equal $\mathbf{X}\mathbf{V}_k = \mathbf{U}_k\mathbf{\Sigma}_k$ and $\mathbf{Y} = \mathbf{V}_k^T$.

Could have just as easily set $\mathbf{Z} = \mathbf{U}_k$ and $\mathbf{Y} = \mathbf{\Sigma}_k\mathbf{V}_k^T$, so $\mathbf{Z}$ has orthonormal columns.

$U U^T = I \Rightarrow U_k U_k^T = I \Rightarrow Z Z^T = I$

Word embeddings

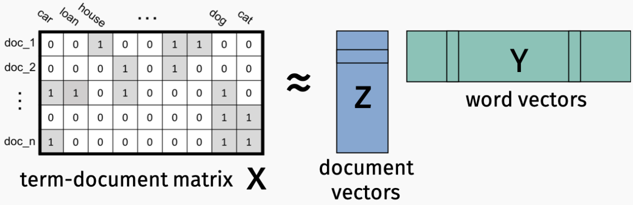
Another view on word embeddings from LSA:



- $\mathbf{X} \approx \mathbf{Z}\mathbf{Y}$ $\rightarrow$ $X^T = (ZY)^T = Y^T Z^T$
- $\mathbf{X}^T\mathbf{X} \approx \mathbf{Y}^T\underbrace{\mathbf{Z}^T\mathbf{Z}}_{I}\mathbf{Y} = \mathbf{Y}^T\mathbf{Y}$
- So for $word_i$ and $word_j$, $\langle \mathbf{y}_i, \mathbf{y}_j \rangle \approx [\mathbf{X}^T\mathbf{X}]_{i,j}$.

What does the i, j entry of $\mathbf{X}^T\mathbf{X}$ reprent?

Word embeddings



What does the i, j entry of $\mathbf{X}^T\mathbf{X}$ represent?

The number of documents where words i and j were both used.

Word embeddings

$\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ is larger if $word_i$ and $word_j$ appear in more documents together (high value in **word-word co-occurrence matrix**, $\mathbf{X}^T\mathbf{X}$). Similarity of word embeddings mirrors similarity of word context.

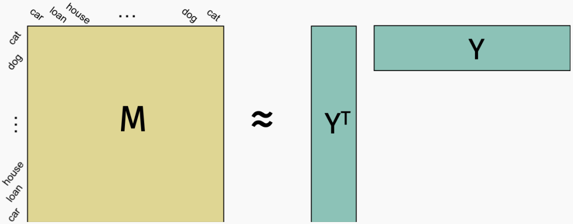
could be $X^T X$

General word embedding recipe:

1. Choose similarity metric $k(word_i, word_j)$ which can be computed for any pair of words.
2. Construct similarity matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ with $\mathbf{M}_{i,j} = k(word_i, word_j)$.
3. Find low rank approximation $\mathbf{M} \approx \mathbf{Y}^T\mathbf{Y}$ where $\mathbf{Y} \in \mathbb{R}^{k \times n}$.
4. Columns of $\mathbf{Y}$ are word embedding vectors.

We expect that $\langle \mathbf{y}_i, \mathbf{y}_j \rangle$ will be larger for more similar words.

Word embeddings



How do current state-of-the-art methods differ from LSA?

- Similarity based on co-occurrence in smaller chunks of words. E.g. in sentences or in any consecutive sequences of 3, 4, or 10 words.
- Usually transformed in non-linear way. E.g. $k(word_i, word_j) = \frac{p(i,j)}{p(i)p(j)}$ where $p(i,j)$ is the frequency both i, j appeared together, and $p(i)$, $p(j)$ is the frequency either one appeared.

Modern word embeddings

Computing word similarities for “window size” 4:

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

The girl walks to her dog to the park.
It can take a long time to park your car in NYC.
The dog park is always crowded on Saturdays.

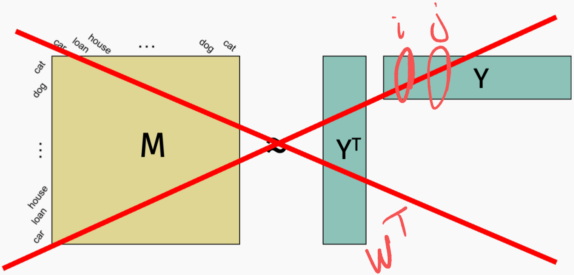
	dog	park	crowded	the
dog	0	~~2~~ 4	0	3
park	~~2~~ 4	0	1	2
crowded	0	1	0	0
the	3	2	0	0

Modern word embeddings

Current state of the art models: `GloVE`, `word2vec`.

- `word2vec` was originally presented as a shallow neural network model, but it is equivalent to matrix factorization method (Levy, Goldberg 2014).
- For `word2vec`, similarity metric is the "point-wise mutual information": $\log \frac{p(i,j)}{p(i)p(j)}$.

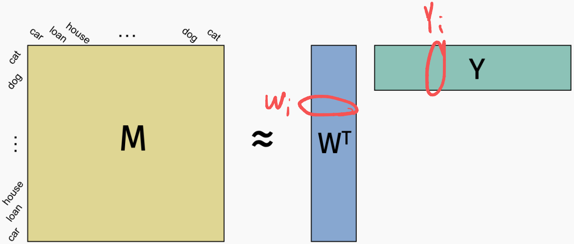
Caveat about factorization



SVD will not return a symmetric factorization in general. In fact, if **M** is not positive semidefinite[1] then the optimal low-rank approximation does not have this form.

[1] I.e., $k(word_i, word_j)$ is not a positive semidefinite kernel.

Caveat about factorization



- For each word i we get a left and right embedding vector $\mathbf{w}_i$ and $\mathbf{y}_i$. It's reasonable to just use one or the other.
- If $\langle \overset{w_i}{\cancel{\mathbf{y}_i}}, \mathbf{y}_j \rangle$ is large and positive, we expect that $\overset{w_i}{\cancel{\mathbf{y}_i}}$ and $\mathbf{y}_j$ have similar similarity scores with other words, so they typically are still related words.
- Another option is to use as your features for a word the concatenation $[\mathbf{w}_i, \mathbf{y}_i]$

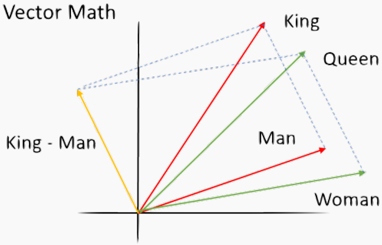
Easiest way to use word embeddings

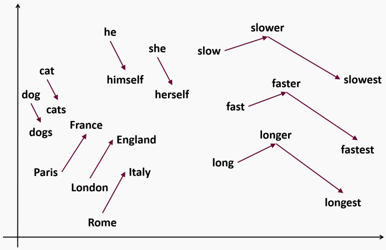
Lots of pre-trained word vectors are available online:

- **Original gloVe website:** `https://nlp.stanford.edu/projects/glove/`.
- **Compilation of many sources:** `https://github.com/3Top/word2vec-api`

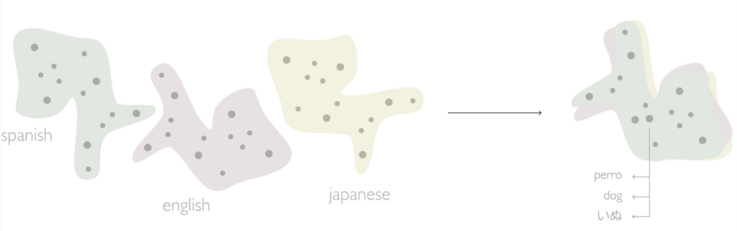
Word embeddings math

Lots of cool demos for what can be done with these embeddings. E.g. "vector math" to solve analogies.





Forward looking application: unsupervised translation



- Train word-embeddings for languages separately. Obtain lowish dimensional point clouds of words.
- Perform rotation/alignment to match up these point clouds.
- Equivalent words should land on top of each other.

No needs for labeled training data like translated pairs of sentences!

Forward looking application: unsupervised translation

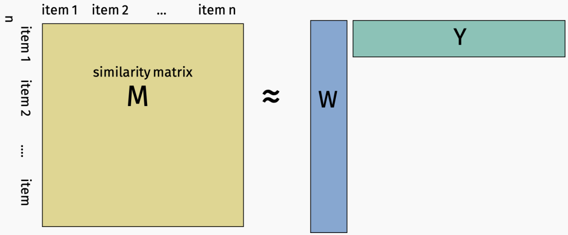
Why not monkey or whale language?

Earth Species Project (www.earthspecies.org), CETI Project (www.projectceti.org)

Semantic embeddings

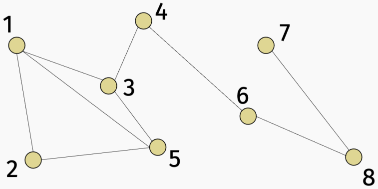
The same approach used for word embeddings can be used to obtain meaningful numerical features for any other data where there is a natural notion of similarity.



For example, the items could be nodes in a social network graph. Maybe be want to predict an individuals age, level of interest in a particular topic, political leaning, etc.

Node embeddings

Generate random walks (e.g. "sentences" of nodes) and measure similarity by node co-occurence frequency.



1, 3, 4, 4, 5, 2, 1, 2, 5
6, 8, 6, 4, 3, 1, 5, 3, 4
7, 8, 6, 8, 7, 8, 6, 8, 6
⋮
4, 6, 8, 6, 4, 3, 1, 2, 5

Node embeddings

Again typically normalized and apply a non-linearity (e.g. log) as in word embeddings.

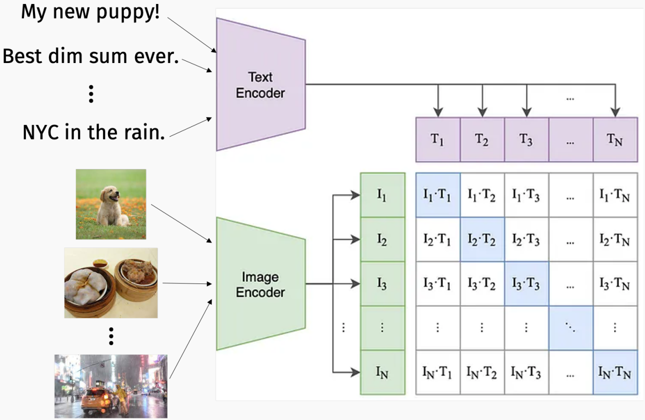
1, 3, 4, 4, 5, 2, 1, 2, 5
6, 8, 6, 4, 3, 1, 5, 3, 4
7, 8, 6, 8, 7, 8, 6, 8, 6
⋮
4, 6, 8, 6, 4, 3, 1, 2, 5

	node 1	node 2	...	node 8
node 1	0	2		1
node 2	2	0		0
...				
node 8	1	0		0

Popular implementations: `DeepWalk`, `Node2Vec`. Again initially derived as simple neural network models, but are equivalent to matrix-factorization (Qiu et al. 2018).

Bimodal embeddings

We can also create embeddings that represent different types of data. OpenAI's clip architecture:



Goal: Train embedding architectures so that $\langle \mathbf{T}_i, \mathbf{I}_j \rangle$ are similar if image and sentence are similar.

Clip training

What do we use as ground truth similarities during training? Sample a batch of sentence/image pairs and just use identity matrix.

			
My new puppy!	1	0	0
Best dim sum ever.	0	1	0
NYC in the rain.	0	0	1

This is called contrastive learning. Train unmatched text/image pairs to have nearly orthogonal embedding vectors.

Clip for zero-shot learning

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford [*1] Jong Wook Kim [*1] Chris Hallacy [1] Aditya Ramesh [1] Gabriel Goh [1] Sandhini Agarwal [1]
Girish Sastry [1] Amanda Askell [1] Pamela Mishkin [1] Jack Clark [1] Gretchen Krueger [1] Ilya Sutskever [1]