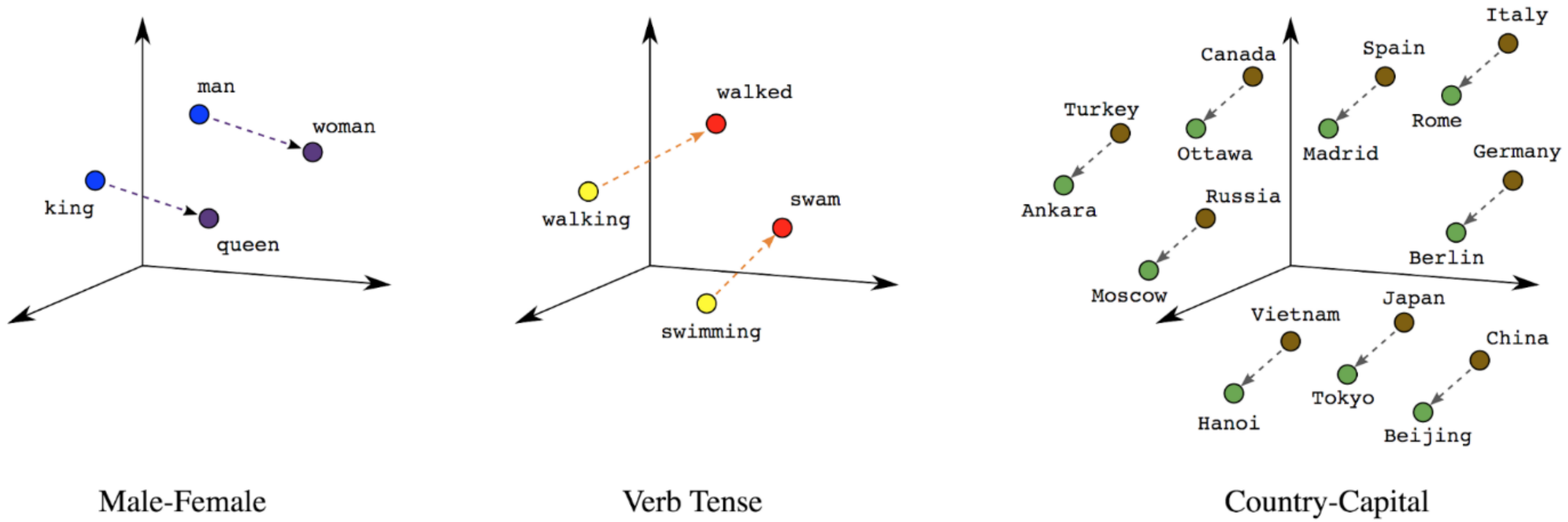
University of Ljubljana, Faculty of Computer and Information Science

Dense embeddings



Prof Dr Marko Robnik-Šikonja

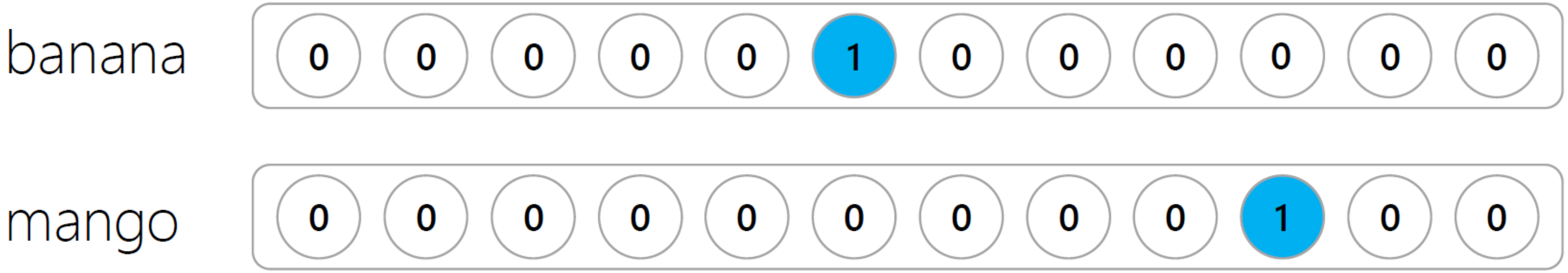
Natural language processing, Edition 2023

Contents

- Dense embeddings
- LSA embedding
- (Neural dense embeddings are covered later)

Why dense textual embeddings?

- Best machine learning models for text (SVM, deep neural networks) require numerical input.
- Simple representations like 1-hot-encoding and bag-of-words do not preserve semantic similarity.
- We need dense vector represenation for text elements.

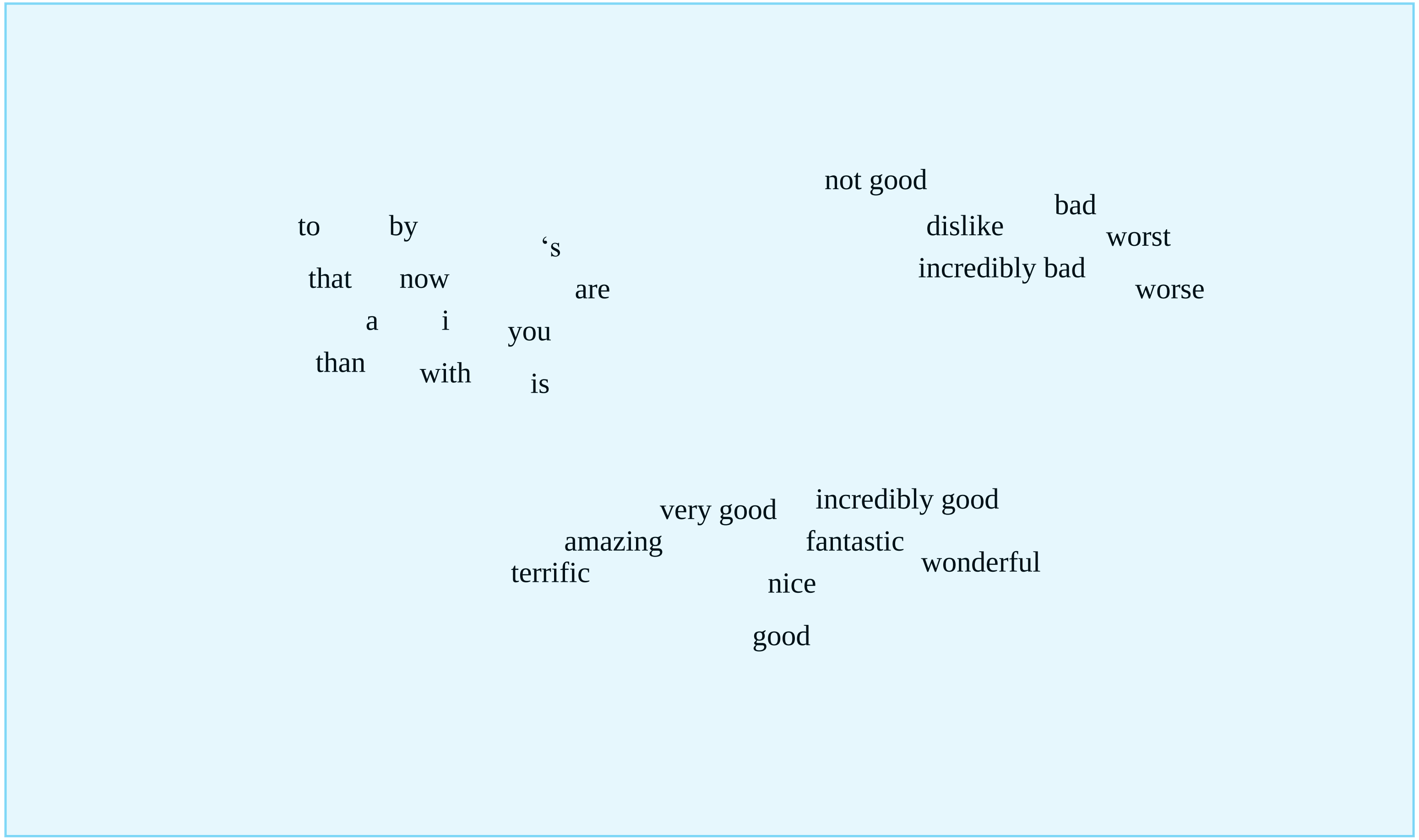


Dense vector embeddings

- advantages compared to sparse embeddings:
  - less dimensions, less space
  - easier input for ML methods
  - potential generalization and noise reduction
  - potentially captures synonymy, e.g., road and highway are different dimensions in BOW
- the most popular approaches
  - matrix based transformations to reduce dimensionality (SVD or LSA)
  - we will cover the following ones later:
    - Brown clustering
    - neural embeddings (word2vec, Glove)
    - contextual neural embeddings (ELMo, BERT)

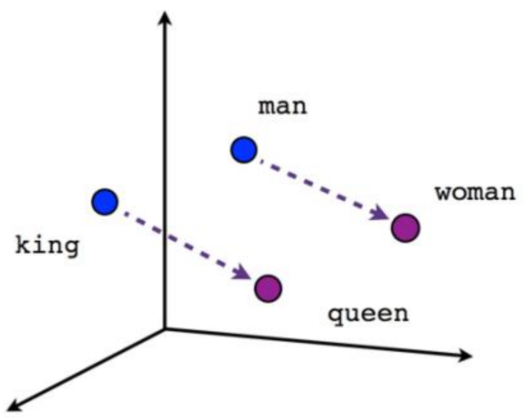
Meaning focused on similarity

- Each word = a vector
- Similar words are "nearby in space"

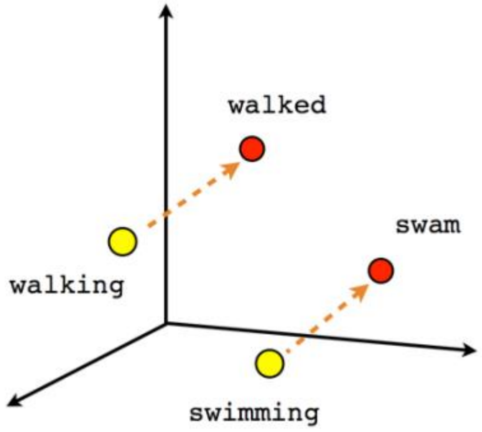


Dense Word Embeddings

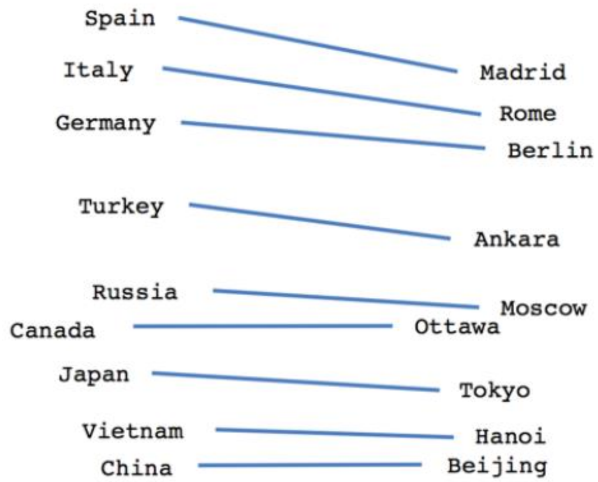
- Word embeddings store semantic and syntactic information
- Word embeddings are currently the standard way to go with natural language processing



Male-Female



Verb tense



Country-Capital

Idea of LSA – Latent Semantic Analysis

- decomposition of word-context matrix with SVD
- approximation with the most important dimensions

Word-word matrix (or "term-context matrix")

- Two **words** are similar in meaning if their context vectors are similar.

sugar, a sliced lemon, a tablespoonful of	**apricot**	jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

SVD for matrices

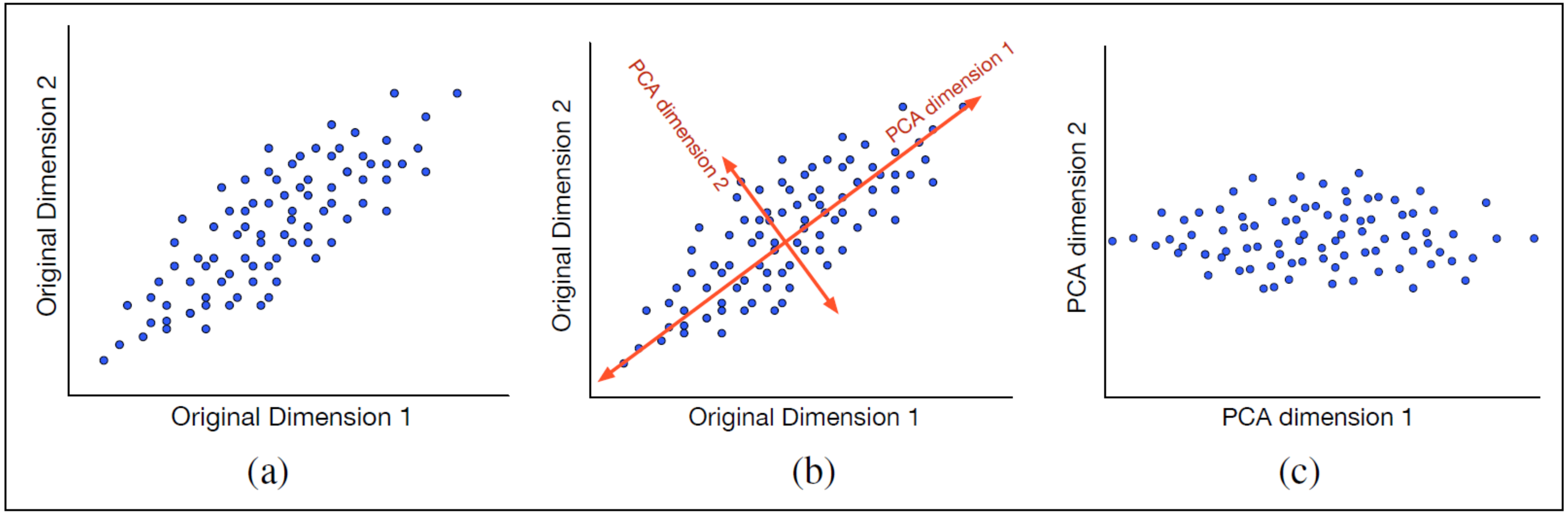
- SVD (singular value decomposition) for arbitrary matrices, generalizes decomposition of eigenvalues

$$M = U\Sigma V^T$$

- approximation of N-dimensional space with lower dimensional space (similarly to PCA)
- in ML used for feature extraction
- a rotation in the direction of the largest variance

Principal components analysis

- principal components analysis, PCA
- we iteratively find the orthogonal axes of the largest variance
- we use the new dimensions to approximate the original space



Latent semantic analysis

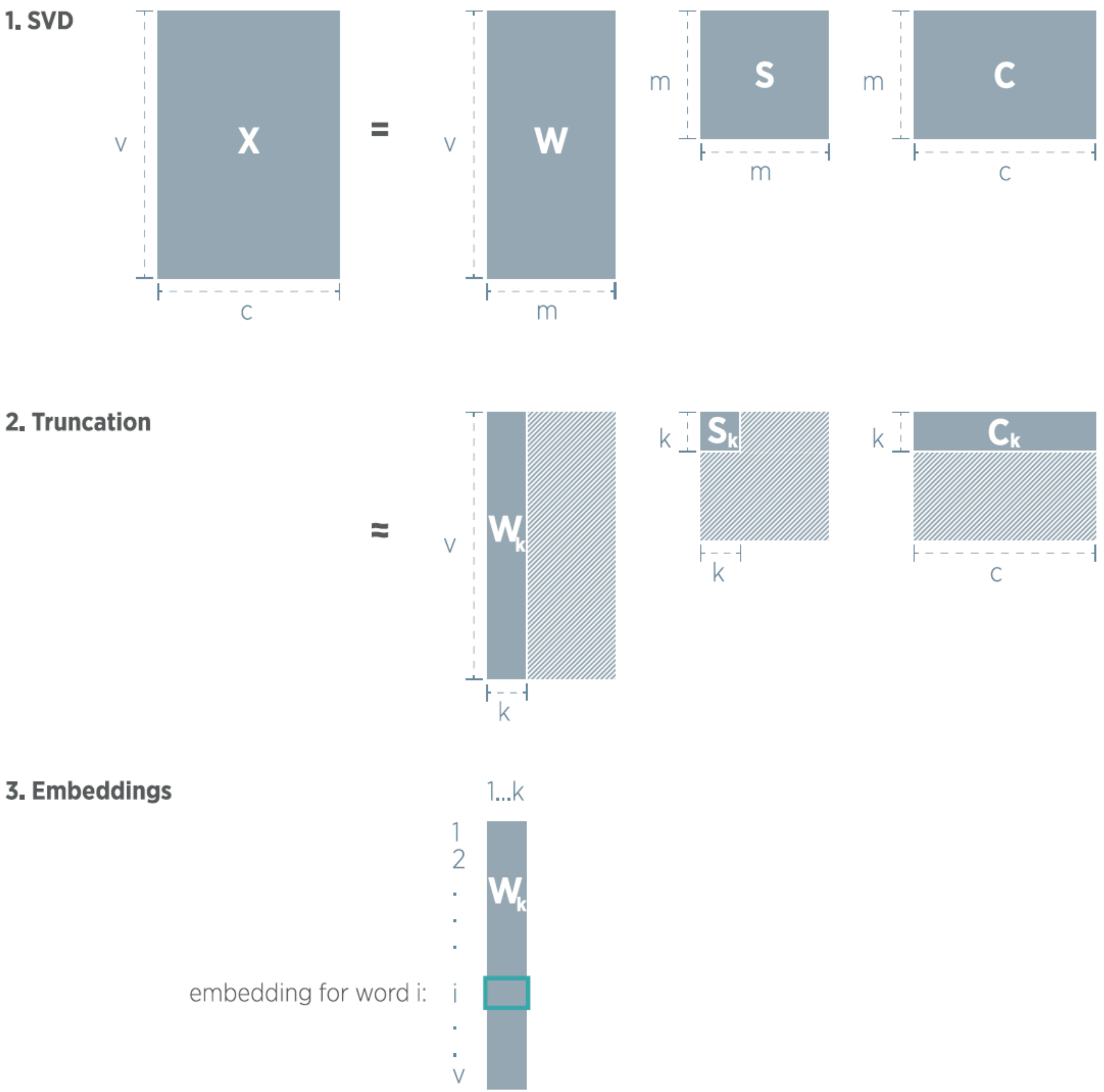
- latent semantic analysis (LSA), also latent semantic indexing (LSI)
- uses SVD on the term-document matrix X of dimension |V| x c, where V is a vocabulary and c the number of documents (contexts)
- $X = W\Sigma C^T$, where
  - W is a matrix of dimension |V| x m;
    rows represent words and columns are dimensions in new latent m-dimensional space
  - Σ is diagonal matrix of dimension m x m with singular values on diagonal
  - C^T is a matrix of dimension m x c, where columns are documents/context in a new m dimensional latent space
- we approximate m original dimensions with the most important k dimensions
- matrix W_k of dimension |V| x k represents embedding of words in lower k - dimensional space

Diagram of LSA

$$\underset{|V|\times c}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V|\times m}{\begin{bmatrix} \\ \\ W \\ \\ \\ \end{bmatrix}} \underset{m\times m}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_m \end{bmatrix}} \underset{m\times c}{\begin{bmatrix} \\ \\ C \\ \\ \\ \end{bmatrix}}$$

$$\underset{|V|\times c}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V|\times k}{\begin{bmatrix} \\ \\ W_k \\ \\ \\ \end{bmatrix}} \underset{k\times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k\times c}{\begin{bmatrix} \quad C \quad \end{bmatrix}}$$

SVD for embeddings



LSA parameters

- usually k=300 or k=500
- weighting with local and global weights
- local weight of each word *i* is log of its frequency in document *j:*
  1+ log f(i, j)
- global weight of each word is a variant of entropy, where ndocs is the number of documents

$$1 + \frac{\sum_j p(i,j) \log p(i,j)}{\log \mathrm{ndocs}}$$

Dense embeddings

Dense. Dim = 200 (for example)

```
In [67]: print(vec['banana'])
         plt.plot(vec['banana'])

[-0.065091, 0.037847, -0.040299, -0.022862, 0.046481, 0.204306, 0.132157, 0.000275, -0.069716, 0.014626, 0.038425, 0.053029, -
0.024947, -0.013991, 0.010317, 0.012735, -0.094237, 0.007101, -0.007268, -0.091869, 0.097138, -0.002357, -0.065102, -0.089856,
 -0.013727, -0.074923, 0.007938, -0.066188, 0.064525, -0.0436, -0.001177, -0.140017, -0.003096, -0.086315, -0.0763, -0.071214,
 -0.051458, 0.123467, 0.031151, 0.068839, -0.039029, 4e-06, -0.127185, -0.049415, -0.007708, 0.035502, 0.009538, -0.075545, 0.0
69583, 0.062794, -0.021556, 0.031155, 0.087352, 0.117663, 0.034883, 0.104613, 0.004534, 0.037999, -0.058016, -0.110679, -0.0353
5, -0.012488, -0.0924, 0.126315, 0.080949, -0.040334, 0.047046, -0.182169, -0.1268, 0.082376, 0.082963, 0.110073, -0.031732, 0.
022219, -0.054332, 0.015394, -0.019853, -0.04169, -0.106969, -0.134253, 0.093094, 0.094716, 0.002643, 0.017417, 0.00309, -0.014
145, 0.078464, 0.041464, 0.026328, 0.12988, -0.02715, 0.027002, -0.014312, -0.017305, -0.066002, 0.002747, 0.033995, 0.053829,
 0.040628, 0.127369, 0.040216, 0.045803, -0.003395, -0.024843, 0.052411, -0.039267, 0.043378, 0.110868, 0.067947, -0.050505, 0.
019753, -0.094825, 0.094058, 0.057547, 0.045447, -0.016258, -0.102323, 0.080506, -0.219969, -0.053595, -0.069609, -0.120579, -
0.048799, -0.019837, -0.109987, -0.002571, 0.031825, -0.124037, -0.024646, -0.102276, 0.038512, 0.035166, 0.031713, 0.008979,
 0.114415, 0.0421, -0.034152, 0.014497, -0.04199, -0.018534, -0.065822, -0.020059, 0.019861, -0.159393, -0.03374, 0.083666, -0.
025234, -0.058921, -0.014924, 0.035292, 0.050979, 0.031609, 0.0322, 0.015638, 0.146793, -0.062475, 0.042192, 0.157084, 0.00237
1, -0.035507, 0.08275, 0.173776, 0.007175, 0.016044, 0.025942, 0.137863, 0.094541, -0.013125, 0.065621, 0.040823, -0.010574, 0.
007796, -0.085031, -0.003617, 0.102267, 0.018047, 0.037613, -0.056187, 0.036693, 0.053867, 0.094616, 0.015941, -0.041536, 0.005
796, -0.03694, -0.063241, -0.067796, -0.026023, 0.069142, -0.008786, 0.042428, -0.017718, 0.03318, -0.052277, 0.114012, 0.08154
2, 0.063282, -0.012149, -0.134274, -0.118431]

Out[67]: [<matplotlib.lines.Line2D at 0x12a60774e48>]
```