Learning Representations of Text using Neural Networks

Tomáš Mikolov
Joint work with Ilya Sutskever, Kai Chen, Greg Corrado, Jeff Dean, Quoc Le, Thomas Strohmann

Google Research

NIPS Deep Learning Workshop 2013

How do we represent the meaning of a word?

Definition: **Meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

How to represent meaning in a computer?

Common answer: Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets (good):

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good

Problems with this discrete representation

- Great as resource but missing nuances, e.g. **synonyms**:
  adept, expert, good, practiced, proficient, skillful?

- Missing new words (impossible to keep up to date):
  wicked, badass, nifty, crack, ace, wizard, genius, ninjia
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity →

Problems with this discrete representation

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: hotel, conference, walk

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

We call this a "one-hot" representation. Its problem:

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

One-hot encoding

- Represent each word as a vector of zeros, except for one element
- Set the value in the vector corresponding to the index in the set to 1:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday  = [1 0 0 0 0]
Tuesday = [0 1 0 0 0]
is      = [0 0 1 0 0]
a       = [0 0 0 1 0]
today   = [0 0 0 0 1]
```

- This is also known as a 1-of-K encoding (with K the vocabulary size)

* Example reproduced with permission from Mikolov.

Related: Bag-of-words representation

- A related representation is the bag-of-words representation for documents
- It simply sums one-hot representation over all words in the document:

```
vocabulary = (Monday, Tuesday, is, a, today)
Monday Monday         = [2 0 0 0 0]
today is a Monday     = [1 0 1 1 1]
today is a Tuesday    = [0 1 1 1 1]
is a Monday today     = [1 0 1 1 1]
```

- Indeed, you could build bags-of-n-grams representations, too

* Example reproduced with permission from Mikolov.

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

“You shall know a word by the company it keeps”

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

How to make neighbors represent words?

Answer: With a cooccurrence matrix X

- 2 options: full document vs windows
- Word - document cooccurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
- Instead: Window around each word → captures both syntactic (POS) and semantic information

Window based cooccurence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Window based cooccurence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problems with simple cooccurrence vectors

Increase in size with vocabulary

Very high dimensional: require a lot of storage

Subsequent classification models have sparsity issues

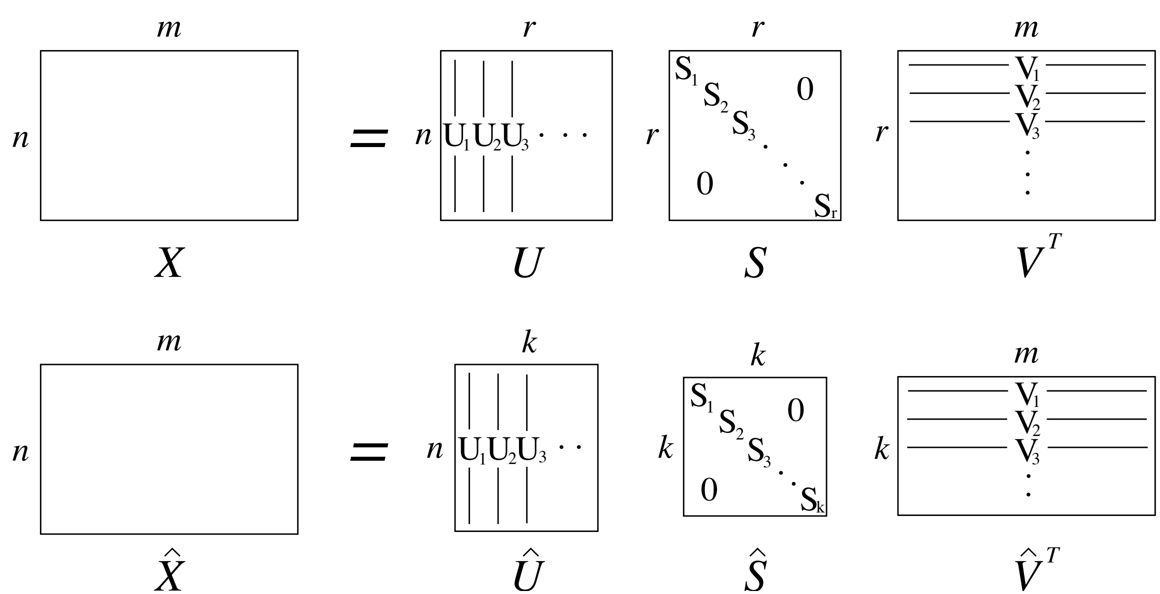
→ Models are less robust

Solution: Low dimensional vectors

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually around 25 – 1000 dimensions
- How to reduce the dimensionality?

Method 1: Dimensionality Reduction on X

Singular Value Decomposition of cooccurrence matrix *X*.



$\hat{X}$ is the best rank k approximation to X , in terms of least squares.

Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

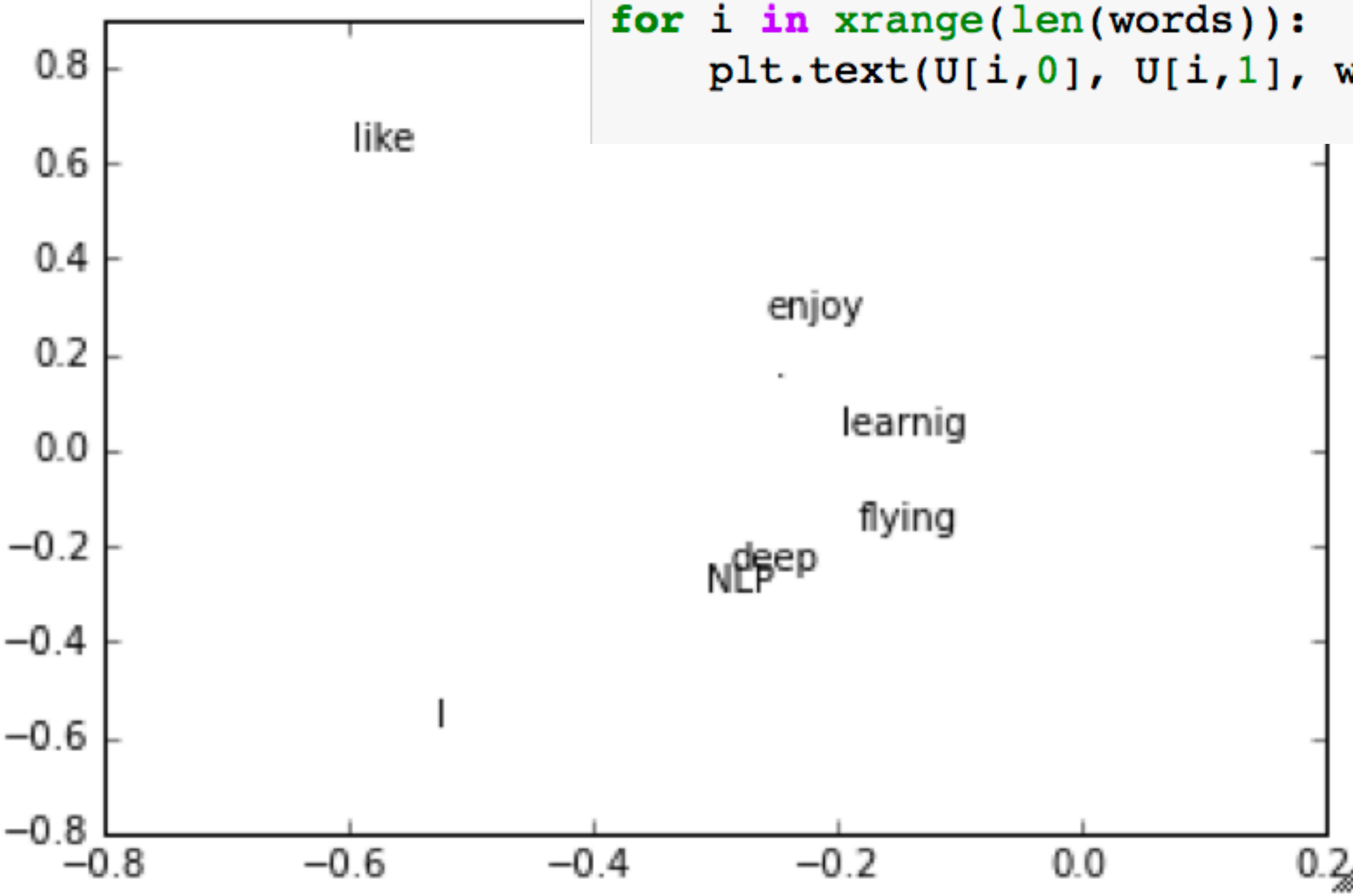
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Word meaning is defined in terms of vectors

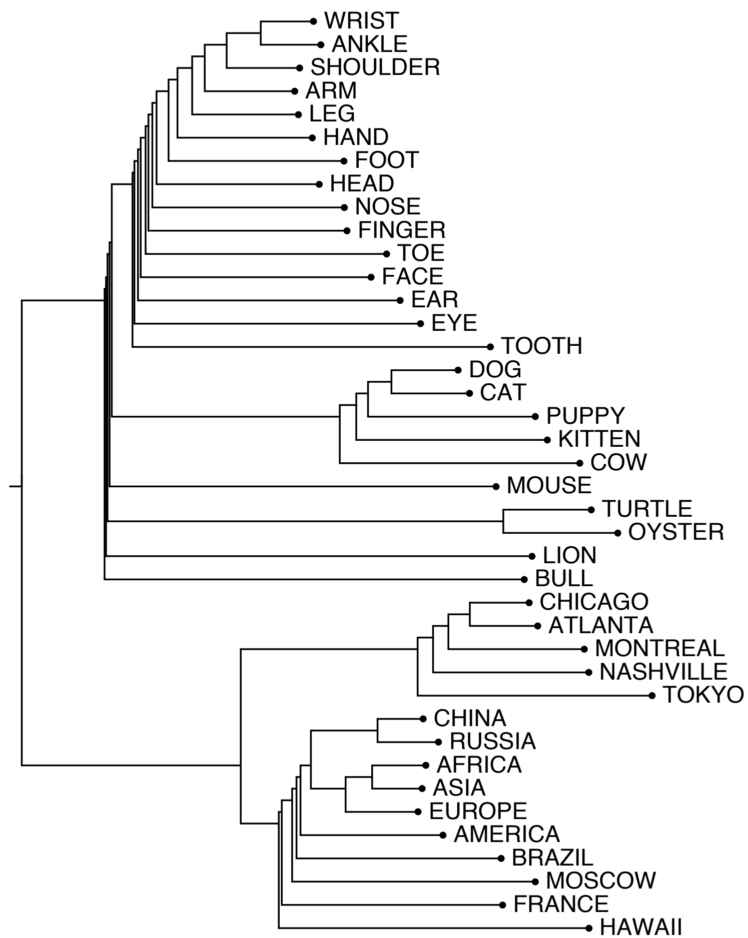
- In all subsequent models, including deep learning models, a word is represented as a dense vector

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Hacks to X

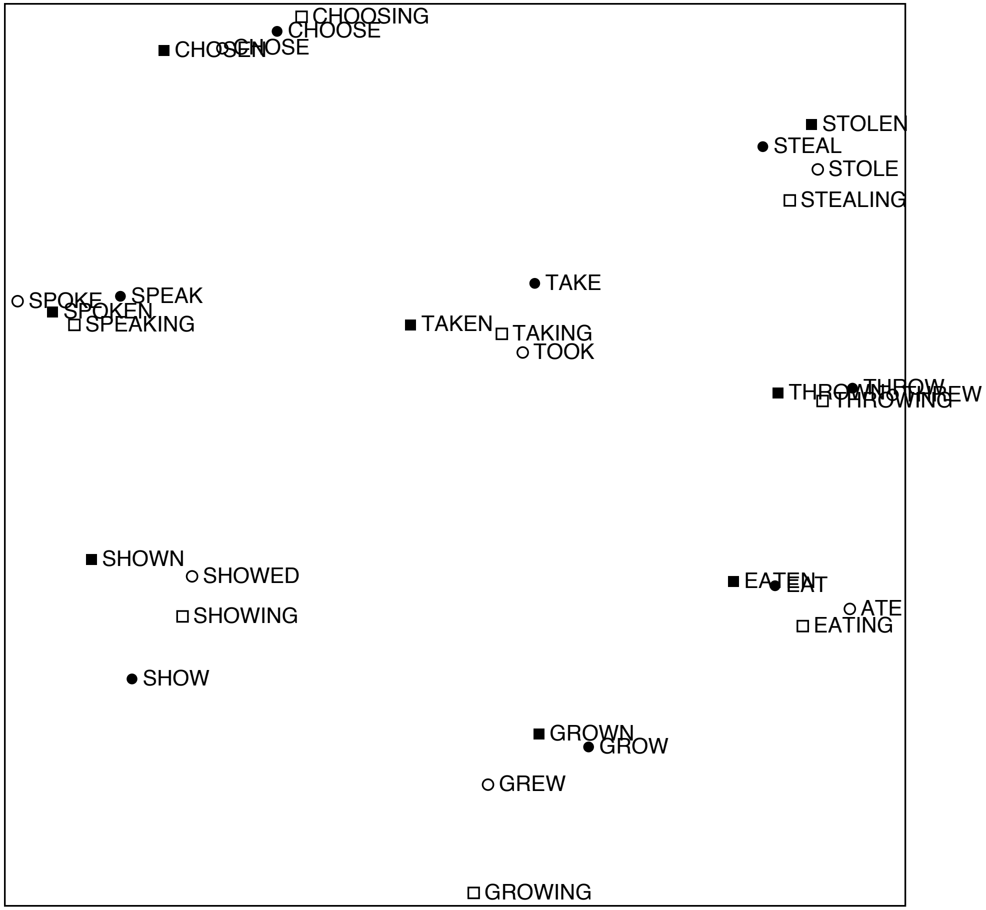
- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t~100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- +++

Interesting semantic patters emerge in the vectors



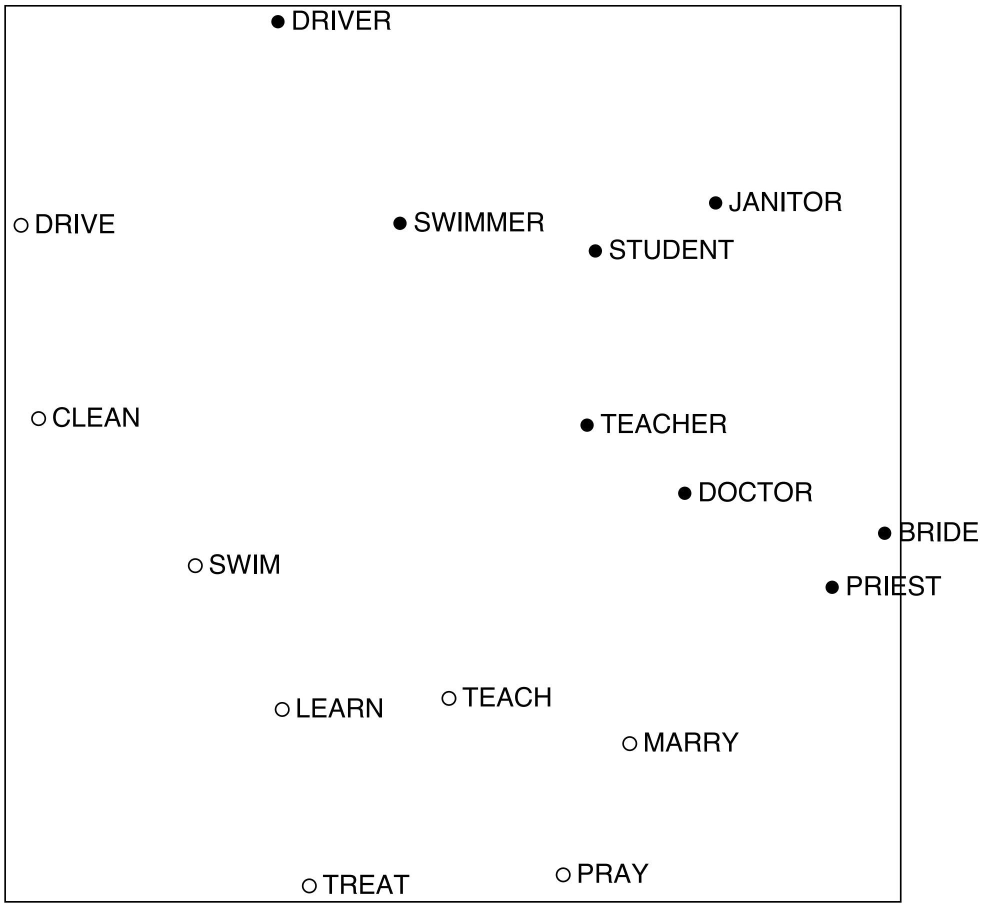
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting syntactic patters emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting semantic patters emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Problems with SVD

Computational cost scales quadratically for n x m matrix:

O(mn^2) flops (when $n<m$)

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models

Distributed Representations

- Distributed representations of words can be obtained from various neural network based language models:
  - Feedforward neural net language model
  - Recurrent neural net language model

Language models

Language models

- Language models aim to predict a word given its surrounding words
- In other words, they aim to build a distribution $p(\mathcal{W}) = p(w_1, w_2, \ldots, w_{|\mathcal{W}|})$

- Standard language models are based on n-grams:
  - The likelihood of a sentence: $p(\mathcal{W}) = \prod_{w_i \in \mathcal{W}} p(w_i | w_{i-1}, w_{i-2}, \ldots, w_{i-n})$
  - All of the probabilities are obtained by counting over a large corpus:

$$p(w_i | w_{i-1}, w_{i-2}) = \frac{C(w_i, w_{i-1}, w_{i-2})}{C(w_{i-1}, w_{i-2})}$$

Language models

- Example of using trigram model to compute the probability of a sentence:

$$p(\text{“}NYU\ is\ an\ excellent\ university\text{”}) = p(\text{“}NYU\text{”}) \times p(\text{“}is\text{”}|\text{“}NYU\text{”}) \times p(\text{“}an\text{”}|\text{“}NYU\text{”}, \text{“}is\text{”})$$
$$\times\, p(\text{“}excellent\text{”}|\text{“}is\text{”}, \text{“}an\text{”}) \times p(\text{“}university\text{”}|\text{“}an\text{”}, \text{“}excellent\text{”})$$

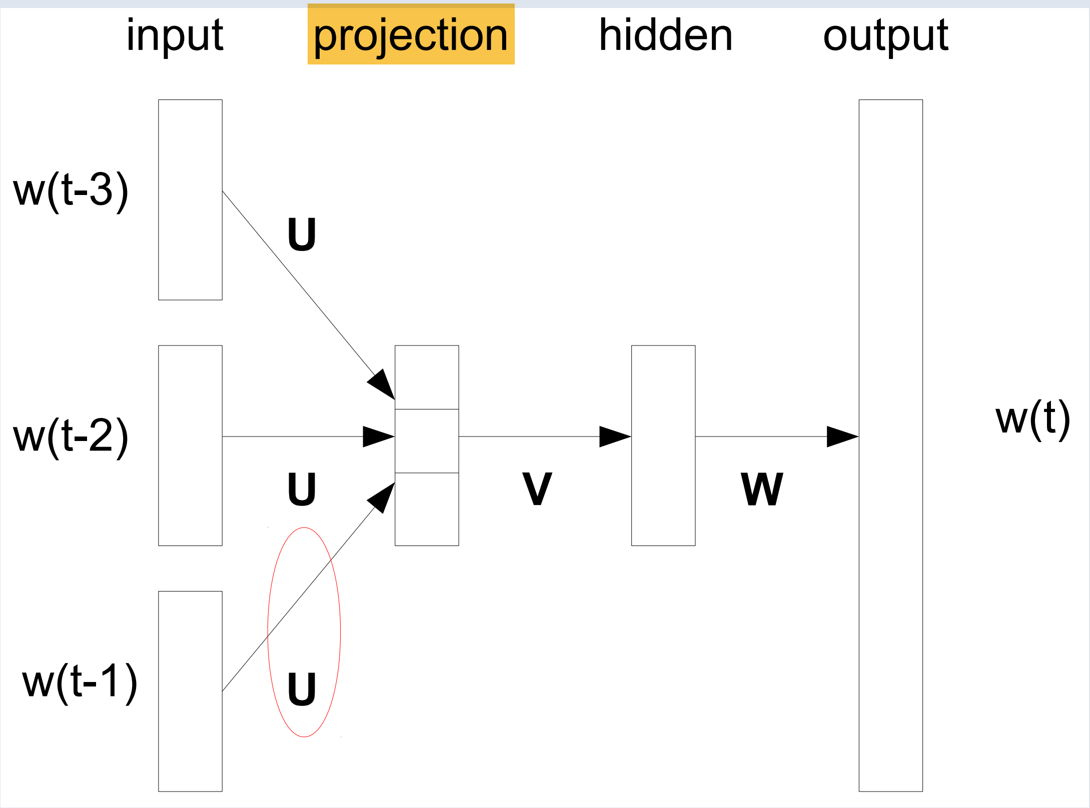
- To deal with non-observed trigrams, Kneser-Ney smoothing is often used
  - For bigrams, this smoother redefines the bigram probabilities as:

$$p_{KN}(w_t|w_{t-1}) = \frac{\max(C(w_{t-1}, w_t) - \delta, 0)}{\sum_{w'} C(w_{t-1}, w')} + \alpha p_{KN}(w_t)$$

  - This redistribution of (n-1)-gram to n-gram probabilities is applied recursively

A Feedforward Language Model

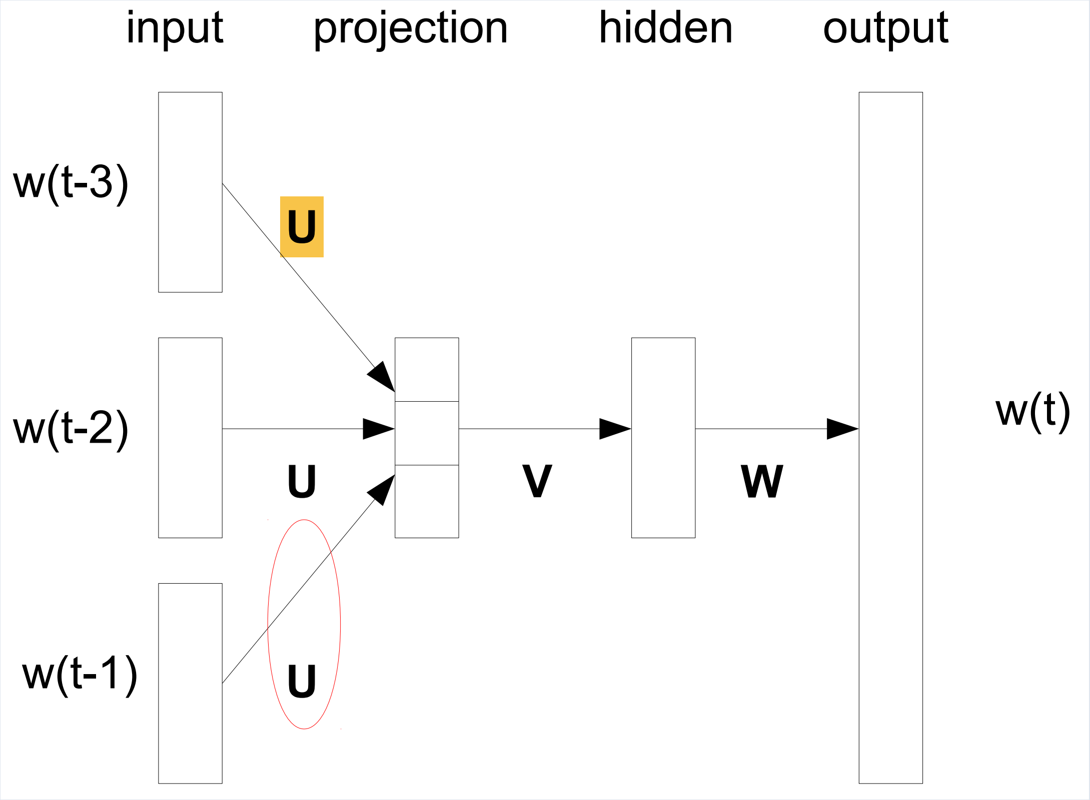
- We could use the following architecture for a word-prediction model:



* Figure reproduced with permission from Mikolov.

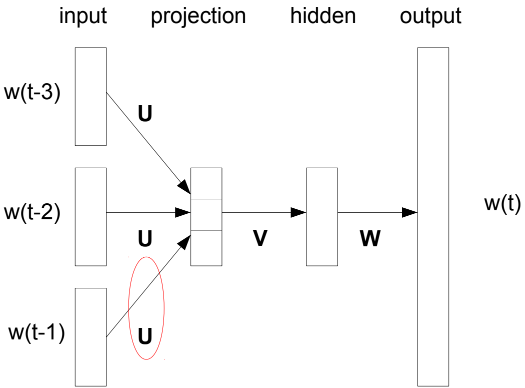
A Feedforward Language Model

- What does the matrix **U** actually model?



* Figure reproduced with permission from Mikolov.

Feedforward Neural Net Language Model



- Four-gram neural net language model architecture (Bengio 2001)
- The training is done using stochastic gradient descent and backpropagation
- The word vectors are in matrix **U**

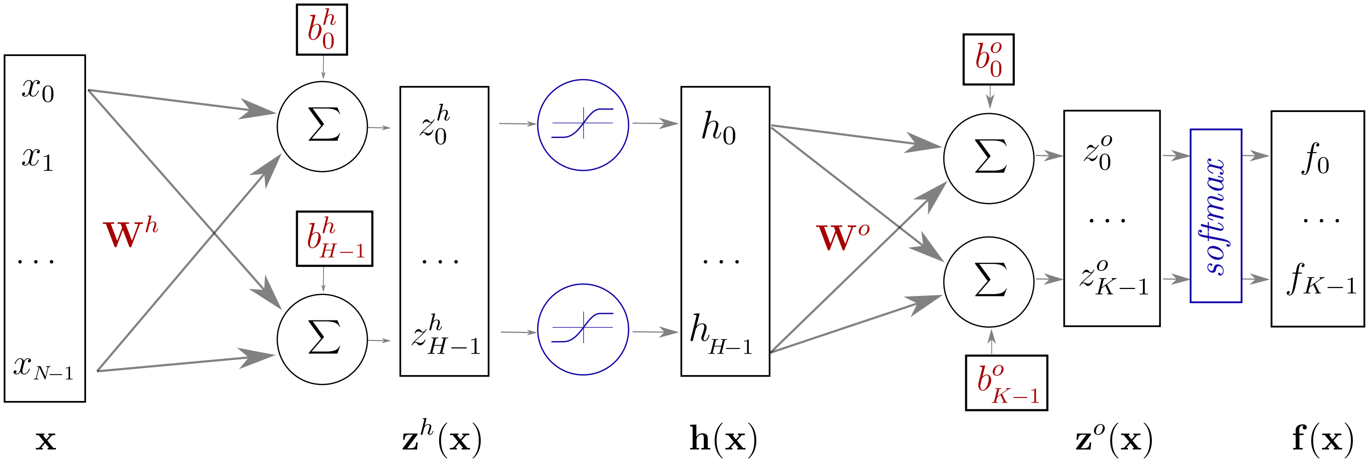
Embedding

- Each column of **U** is an "embedding" of the corresponding word
- You can thus think of each word as being represented by a point that is "embedded" in a high-dimensional space
- If the language model is trained well, then words that can be used interchangeably should have similar embeddings

Efficient Learning

- The training complexity of the feedforward NNLM is high:
  - Propagation from projection layer to the hidden layer
  - Softmax in the output layer
- Using this model just for obtaining the word vectors is very inefficient

One Hidden Layer Network



Keras implementation

```
model = Sequential()
model.add(Dense(H, input_dim=N))  # weight matrix dim [N * H]
model.add(Activation("tanh"))
model.add(Dense(K))               # weight matrix dim [H x K]

model.add(Activation("softmax"))
```

Efficient Learning

- The full softmax can be replaced by:
  - Hierarchical softmax (Morin and Bengio)
  - Hinge loss (Collobert and Weston)
  - Noise contrastive estimation (Mnih et al.)
  - Negative sampling (our work)
- We can further remove the hidden layer: for large models, this can provide additional speedup 1000x
  - Continuous bag-of-words model
  - Continuous skip-gram model

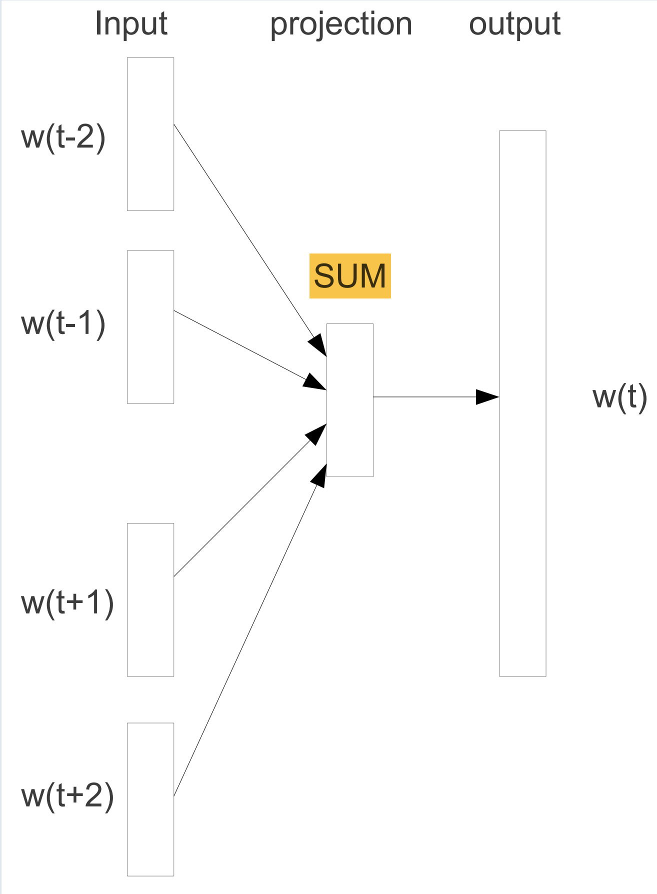
Word2vec

- Word2vec is a very simple, very efficient language model:
  - It does not concatenate word embeddings, but it sums them
  - It does not use the second hidden layer
  - It does not use a multi-class logistic loss (softmax) over predictions

- Because it is so simple, it can be trained on billions of words
- This has made it the de-facto standard in word embedding

Word2vec: Architectures

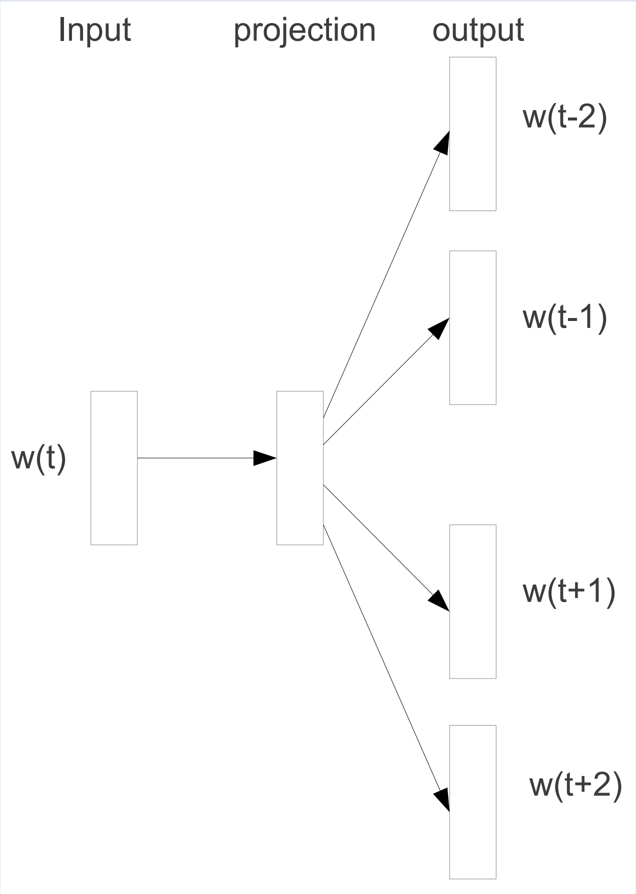
- "Continuous BoW" predicts current word given the surrounding words:



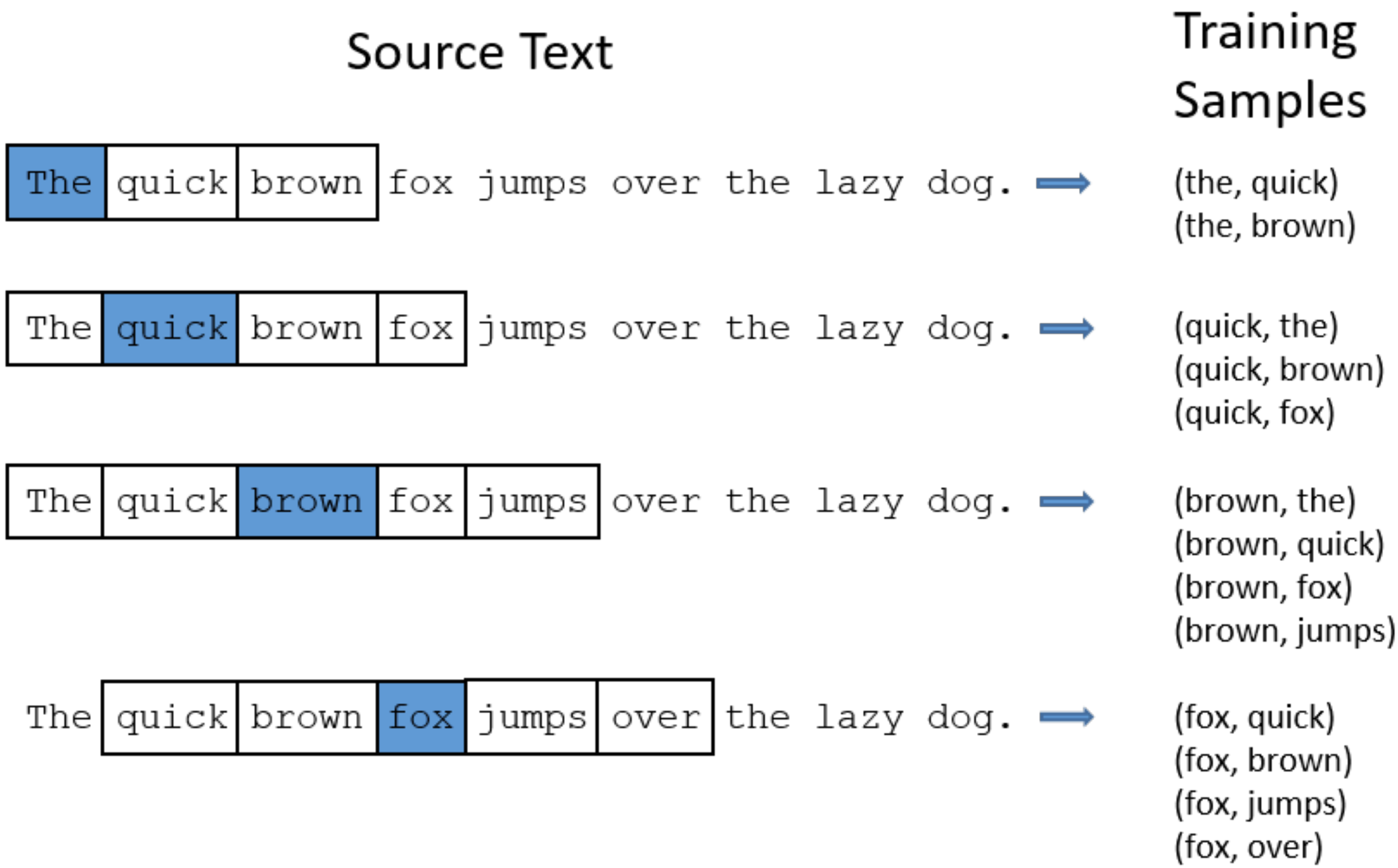
* Figure reproduced with permission from Mikolov.

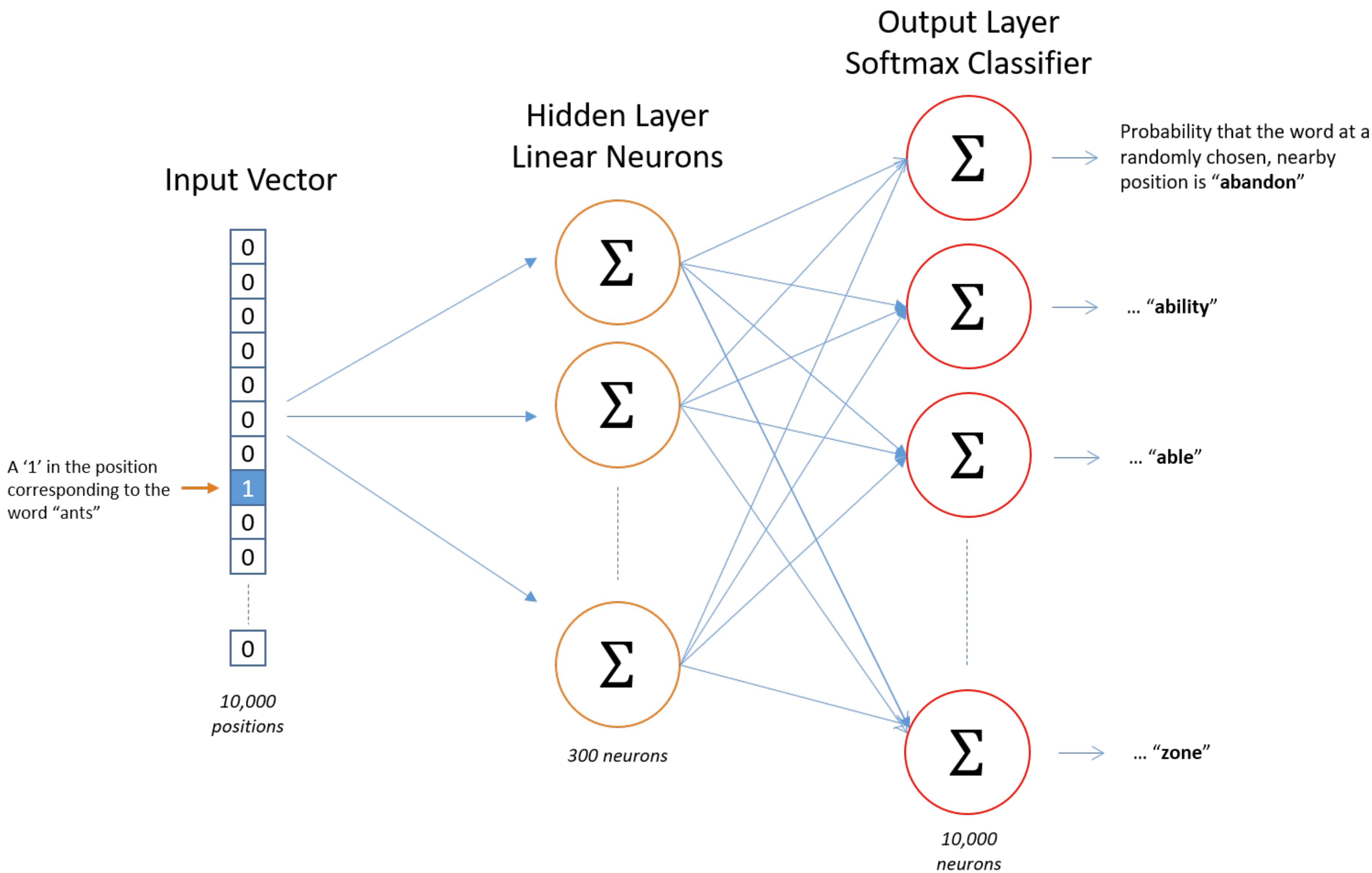
Word2vec: Architectures

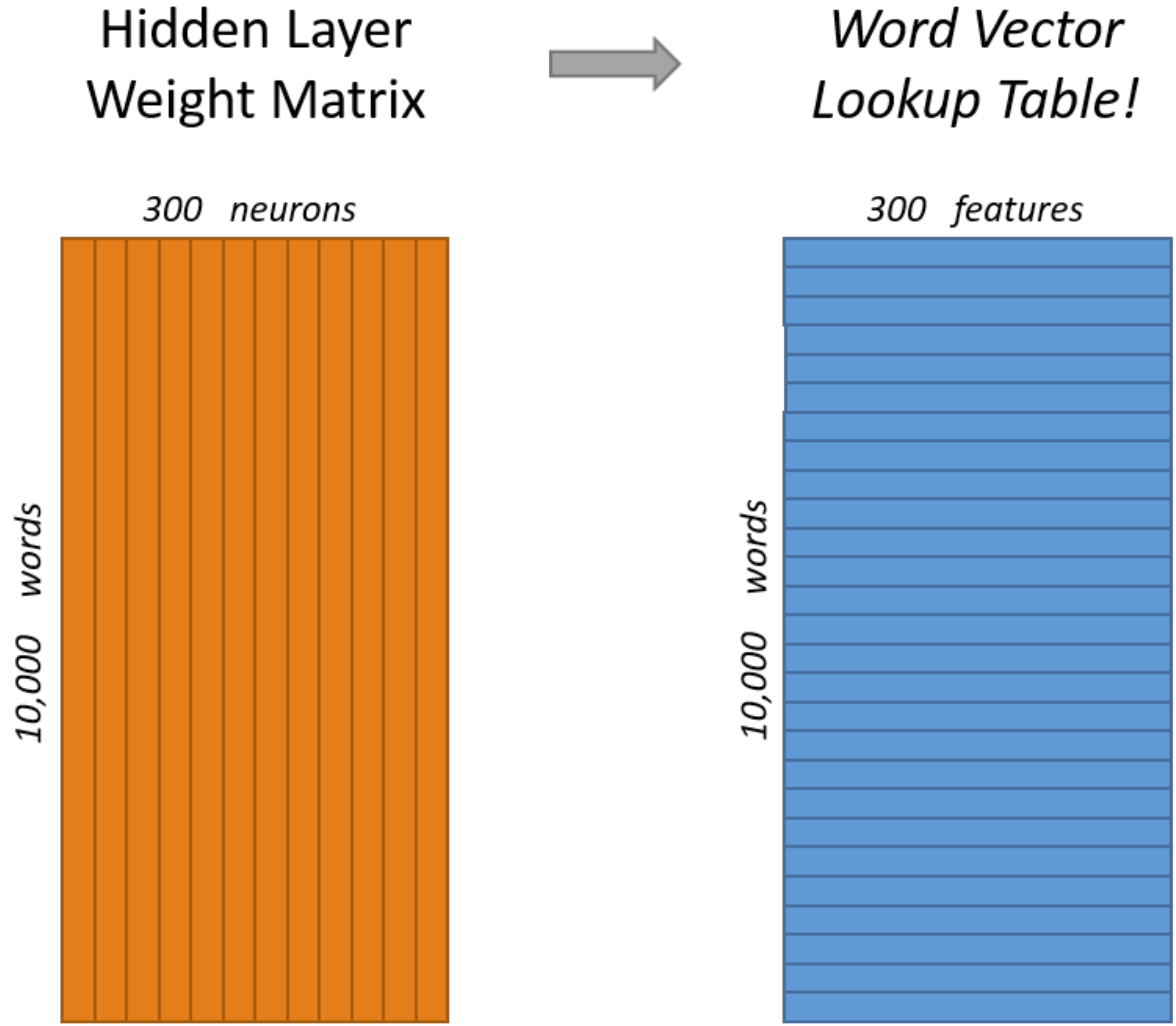
- "Skip-gram" predicts surrounding words given the current word:



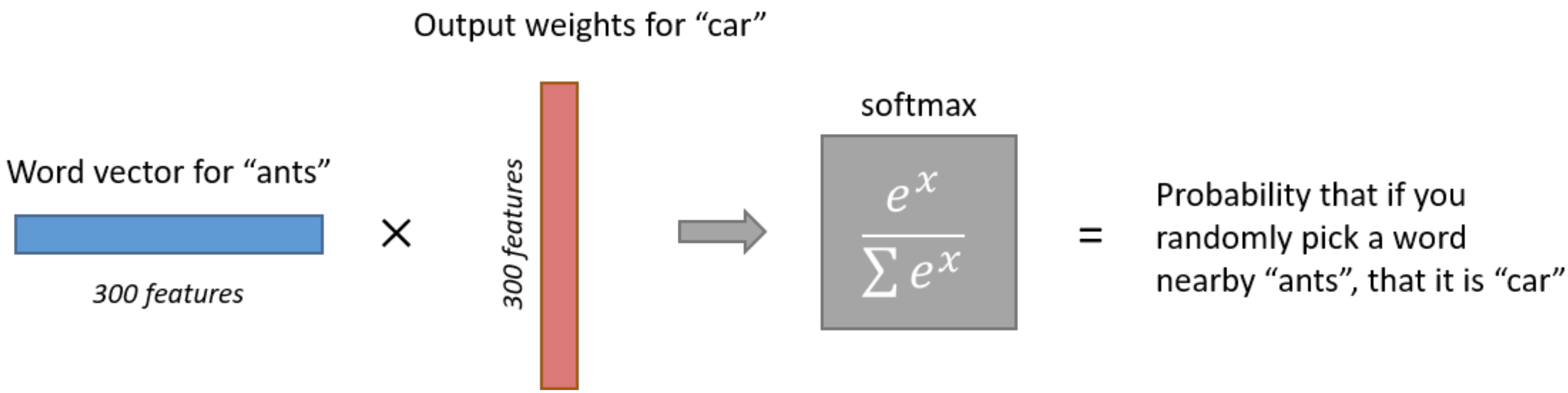
* Figure reproduced with permission from Mikolov.


Source Text
Training Samples
The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)


Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "**abandon**"
... "**ability**"
... "**able**"
... "**zone**"


Hidden Layer
Weight Matrix
Word Vector
Lookup Table!
300 neurons
10,000 words
300 features
10,000 words

$$\begin{bmatrix}0 & 0 & 0 & \boxed{1} & 0\end{bmatrix} \times \begin{bmatrix}17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & \boxed{12} & \boxed{19} \\ 11 & 18 & 25\end{bmatrix} = \begin{bmatrix}10 & 12 & 19\end{bmatrix}$$


Output weights for “car”
Word vector for “ants”
300 features
×
300 features
softmax
$$\frac{e^x}{\sum e^x}$$
=
Probability that if you randomly pick a word nearby “ants”, that it is “car”

Training the network

Find parameters $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize the negative log likelihood (or cross entropy)

The loss function for a given sample $s \in S$:

$$l(\mathbf{f}(\mathbf{x}^s; \theta), y^s) = nll(\theta; \mathbf{x}^s, y^s) = -\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s}$$

example $\quad y^s = 3$

$$-\log \mathbf{f}(\mathbf{x}^s; \theta)_{y^s} = \underbrace{\begin{bmatrix} f_0 \\ \cdots \\ f_3 \\ \cdots \\ f_{K-1} \end{bmatrix}}_{\mathbf{f}(\mathbf{x}^s; \theta)} = -\log f_3$$

Softmax Regression

$$P(y^{(i)} = k | x^{(i)}; \theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}$$

$$J(\theta) = -\left[\sum_{i=1}^{m} \sum_{k=1}^{K} 1\left\{ y^{(i)} = k \right\} \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \right]$$

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} \left[x^{(i)} \left(1\{ y^{(i)} = k \} - P(y^{(i)} = k | x^{(i)}; \theta) \right) \right]$$

http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

Maximize Conditional Log Likelihood: Gradient ascent

$$P(Y=1|X,W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$l(\mathbf{w}) = \sum_j y^j (w_0 + \sum_i^n w_i x_i^j) - \ln(1 + exp(w_0 + \sum_i^n w_i x_i^j))$$

$$\frac{\partial l(w)}{\partial w_i} = \sum_j \left[\frac{\partial}{\partial w_i} y^j (w_0 + \sum_i w_i x_i^j) - \frac{\partial}{\partial w_i} \ln \left(1 + \exp(w_0 + \sum_i w_i x_i^j) \right) \right]$$

$$= \sum_j \left[y^j x_i^j - \frac{x_i^j \exp(w_0 + \sum_i w_i x_i^j)}{1 + \exp(w_0 + \sum_i w_i x_i^j)} \right]$$

$$= \sum_j x_i^j \left[y^j - \frac{\exp(w_0 + \sum_i w_i x_i^j)}{1 + \exp(w_0 + \sum_i w_i x_i^j)} \right]$$

$$\boxed{\frac{\partial l(w)}{\partial w_i} = \sum_j x_i^j \left(y^j - P(Y^j = 1 | x^j, w) \right)}$$

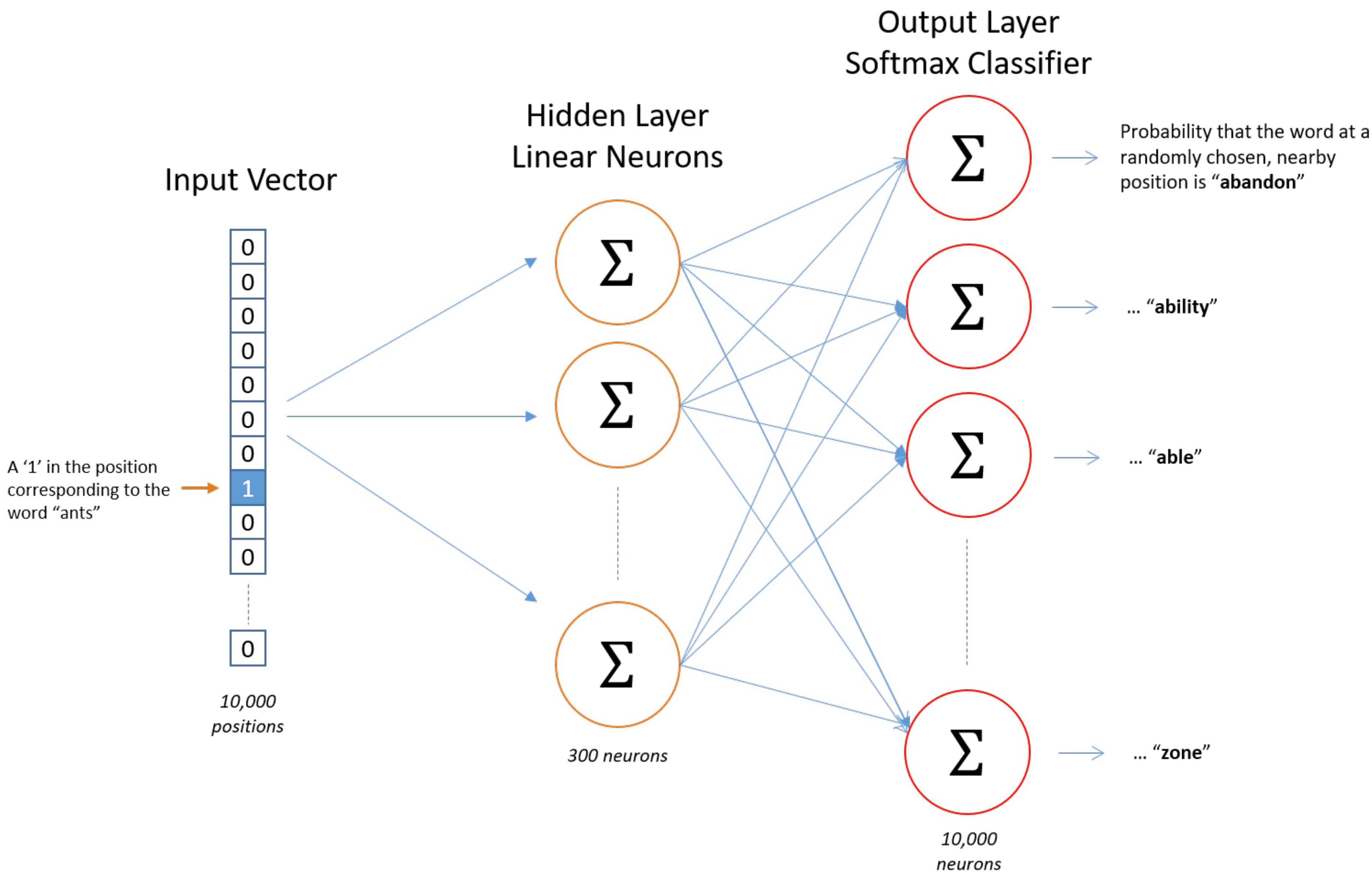
Word2vec: Loss function

- Word2vec minimizes a binary logistic loss on positive and negative samples:

$$\ell(\mathbf{U}) = \sum_{c \in \mathcal{C}} \log \sigma \left(\mathbf{u}_{w_t}^\top \mathbf{u}_{w_c} \right) + \sum_{j=1}^{K} \log \sigma \left(-\mathbf{u}_{w_t}^\top \mathbf{u}_{v'} \right) \quad \text{with } v' \sim P(\mathcal{V})$$

- Learning with SGD in this loss is very efficient:
  - Take a word and its context from a text corpus
  - Sample K words (typically, $5<K<20$) from the unigram distribution
  - Compute the loss and the (sparse!) gradient update

- Multithreaded implementation allows for training speed of 5M words / sec

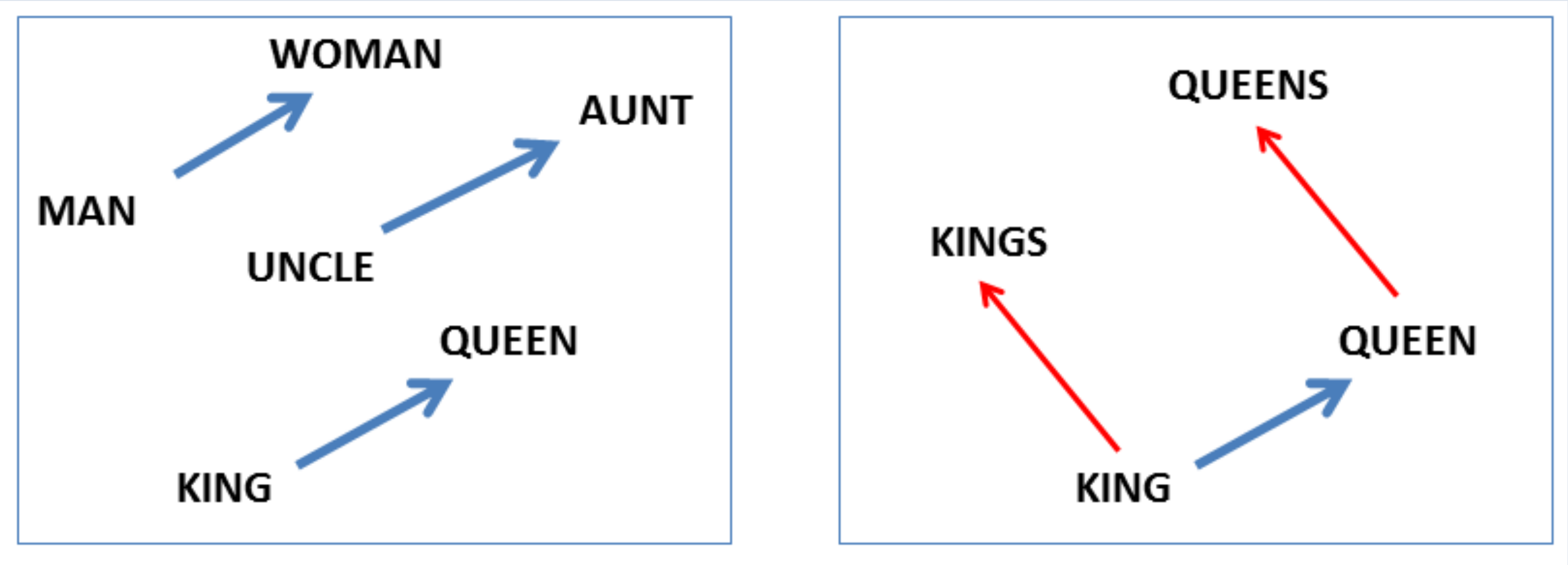

Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "**abandon**"
... "**ability**"
... "**able**"
... "**zone**"

Efficient Learning - Summary

- Efficient multi-threaded implementation of the new models greatly reduces the training complexity
- The training speed is in order of 100K - 5M words per second
- Quality of word representations improves significantly with more training data

Linguistic regularities

- Vector space implicitly encode regularities among words:



- We can exploit these regularities to do "linguistic arithmetic"

* Figure reproduced with permission from Mikolov.

- The resulting distributed representations of words contain surprisingly a lot of syntactic and semantic information
- There are multiple degrees of similarity among words:
  - $\mathbf{KING}$ is similar to $\mathbf{QUEEN}$ as $\mathbf{MAN}$ is similar to $\mathbf{WOMAN}$
  - $\mathbf{KING}$ is similar to $\mathbf{KINGS}$ as $\mathbf{MAN}$ is similar to $\mathbf{MEN}$
- Simple vector operations with the word vectors provide very intuitive results

Linguistic Regularities - Results

- Regularity of the learned word vector space is evaluated using test set with about 20K questions
- The test set contains both syntactic and semantic questions
- We measure TOP1 accuracy (input words are removed during search)
- We compare our models to previously published word vectors

Linguistic regularities

- Make dataset with analogies: "A is to B like C is to D"
- Answer the question "A is to B like C is to ?" by finding the word embedding that is closest to the embedding of B - A + C

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Our NNLM	100	6B	2.5 days	51
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

* Table reproduced with permission from Mikolov.

Linguistic regularities

- Some examples of regularities:

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

* Table reproduced with permission from Mikolov.

Linguistic regularities

- Compositionally by vector addition:

Expression	*Nearest tokens*
Czech + currency	koruna, Czech crown, Polish zloty, CTK
Vietnam + capital	Hanoi, Ho Chi Minh City, Viet Nam, Vietnamese
German + airlines	airline Lufthansa, carrier Lufthansa, flag carrier Lufthansa
Russian + river	Moscow, Volga River, upriver, Russia
French + actress	Juliette Binoche, Vanessa Paradis, Charlotte Gainsbourg

* Table reproduced with permission from Mikolov.

From Words to Phrases and Beyond

- Often we want to represent more than just individual words: phrases, queries, sentences
- The vector representation of a query can be obtained by:
  - Forming the phrases
  - Adding the vectors together

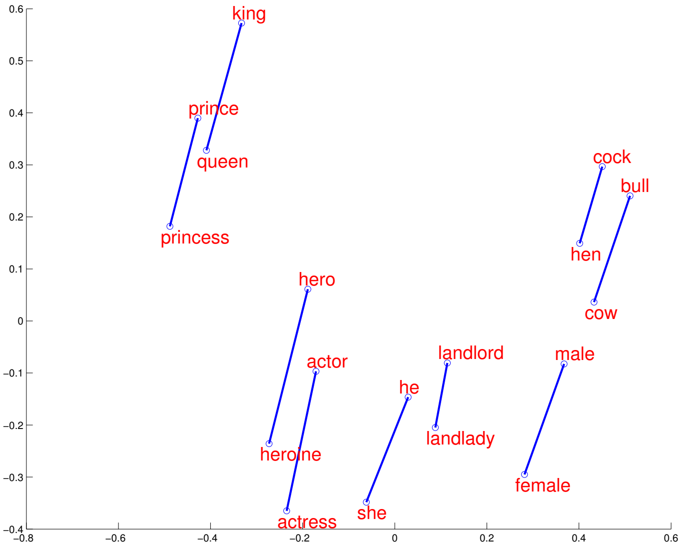
From Words to Phrases and Beyond

- Example query:
  restaurants in mountain view that are not very good
- Forming the phrases:
  restaurants in (mountain view) that are (not very good)
- Adding the vectors:
  restaurants + in + (mountain view) + that + are + (not very good)
- Very simple and efficient
- Will not work well for long sentences or documents

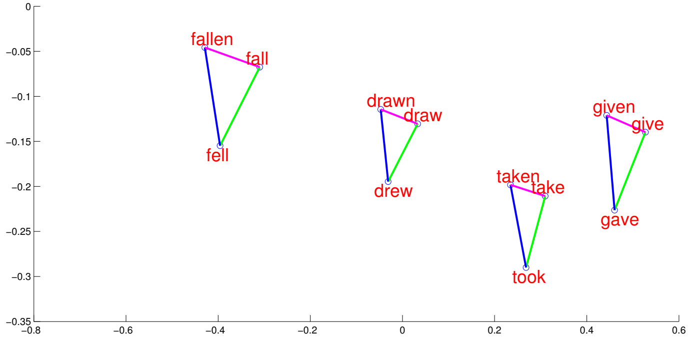
Visualization of Regularities in Word Vector Space

- We can visualize the word vectors by projecting them to 2D space
- PCA can be used for dimensionality reduction
- Although a lot of information is lost, the regular structure is often visible

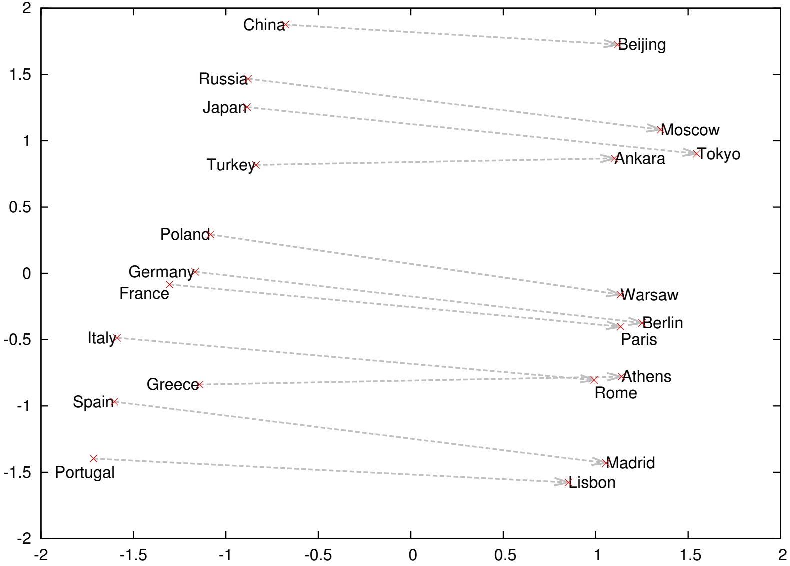
Visualization of Regularities in Word Vector Space



Visualization of Regularities in Word Vector Space



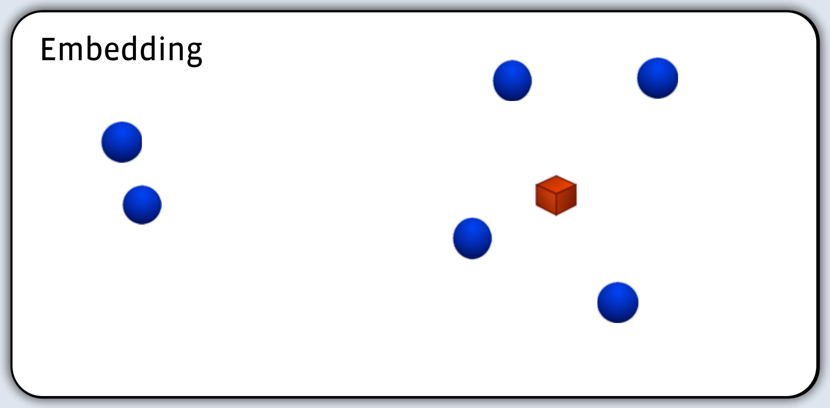
Visualization of Regularities in Word Vector Space



Visualizing graphs

Introduction

- Presume we are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- How do we learn a representation (embedding) for the vertices $v \in \mathcal{V}$?
  - We would like vertices that are connected to have similar embeddings
  - These embeddings we could then use in learning deep networks
  - If the embeddings are low-dimensional: use them to visualize the graph!



t-Stochastic Neighbor Embedding

- Convert the graph to a probability distribution over vertices:

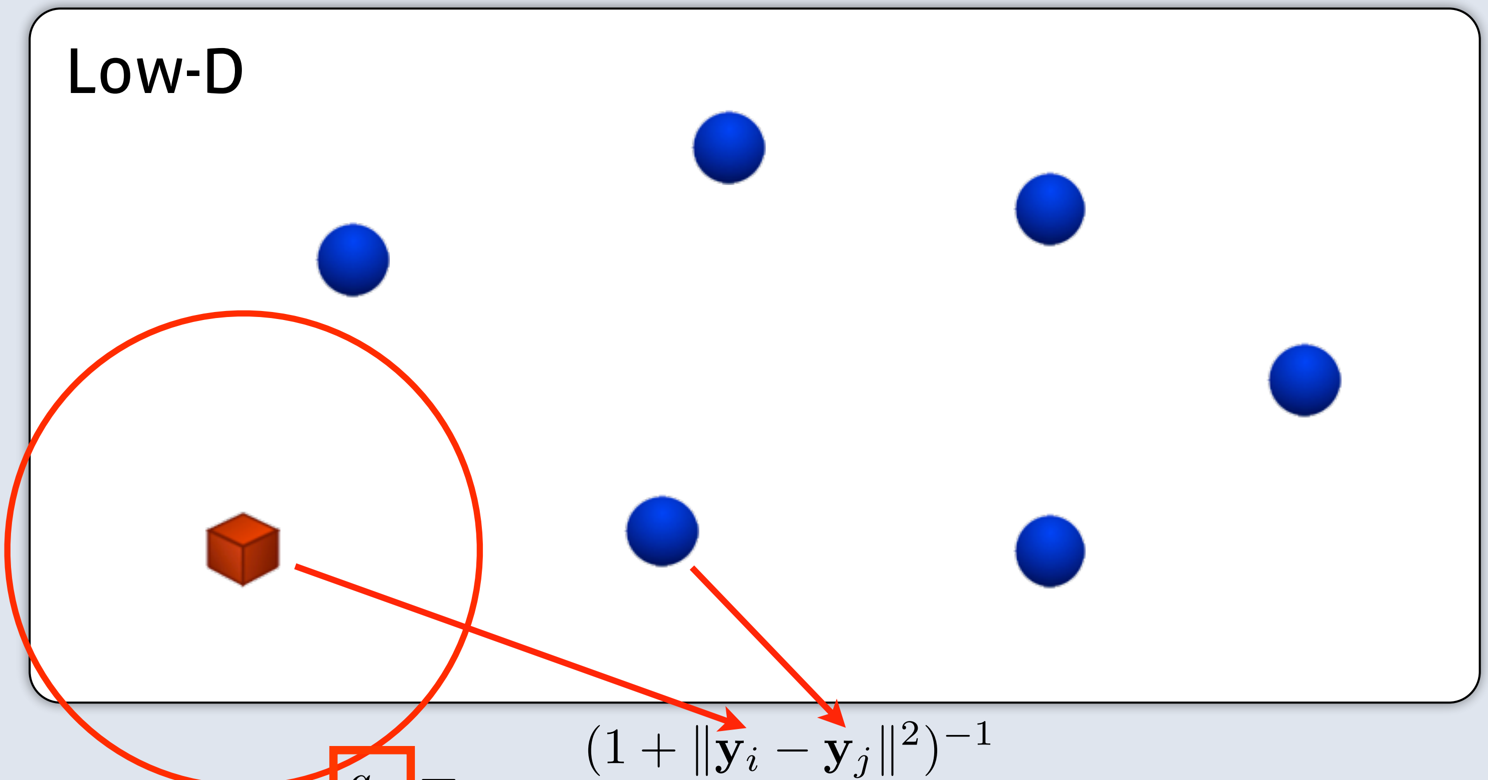
$$p_{ij} = \frac{\exp(e_{ij})}{\sum_{k \neq l} \exp(e_{kl})}$$

- Strongly connected nodes will have large probabilities p_{ij}

- When given high-dimensional data, we can compute similar probabilities:
  - For instance, set $e_{ij} = -\|\mathbf{x}_i - \mathbf{x}_j\|^2$ with $\mathbf{x}_i \in \mathbb{R}^D$
  - Note that this is essentially a (normalized) Gaussian kernel matrix

t-Stochastic Neighbor Embedding

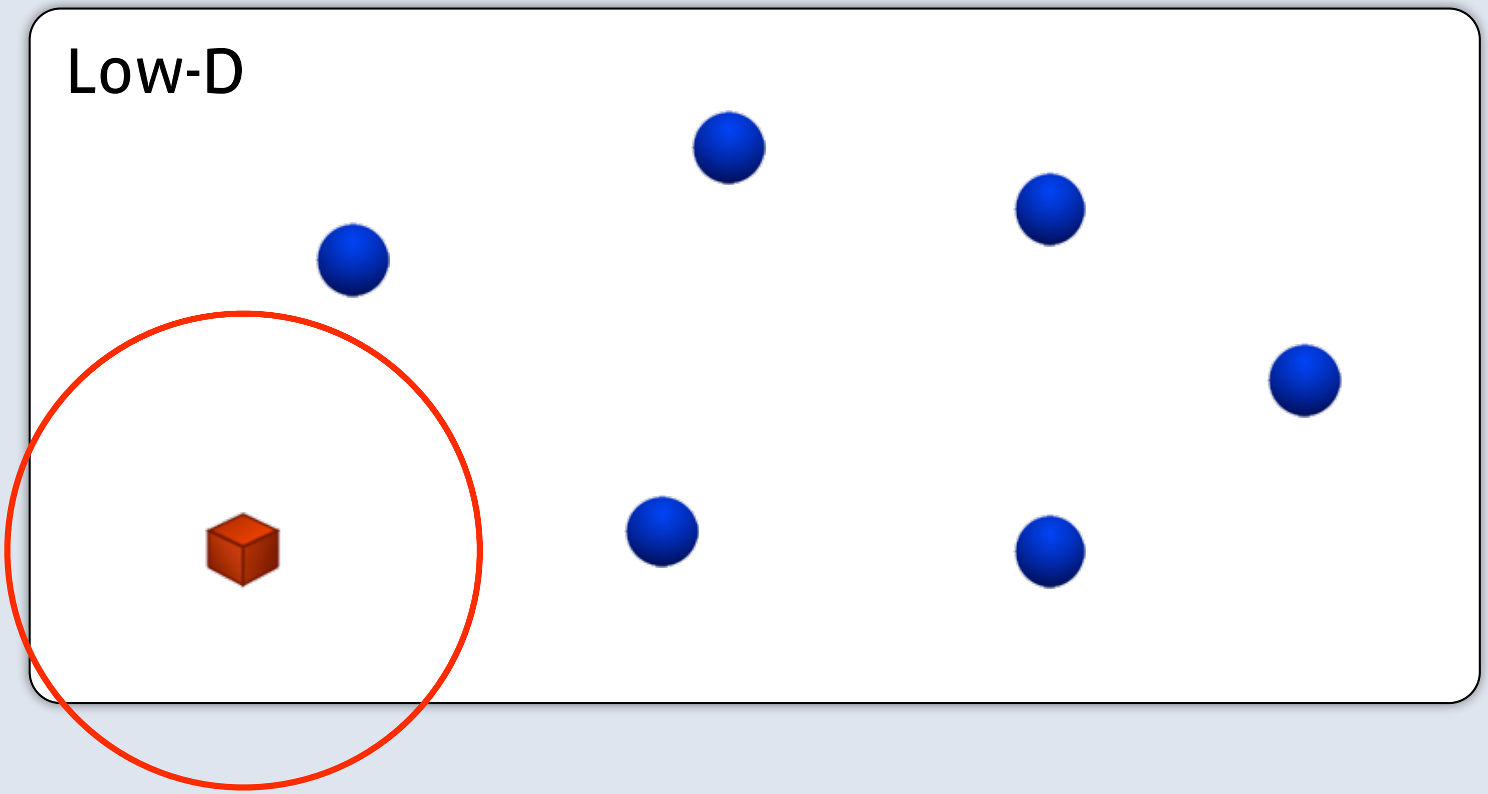
- Measure pairwise similarities between the embeddings:



$$\boxed{q_{ij}} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

t-Stochastic Neighbor Embedding

- Move points around to minimize: $KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$



t-Stochastic Neighbor Embedding

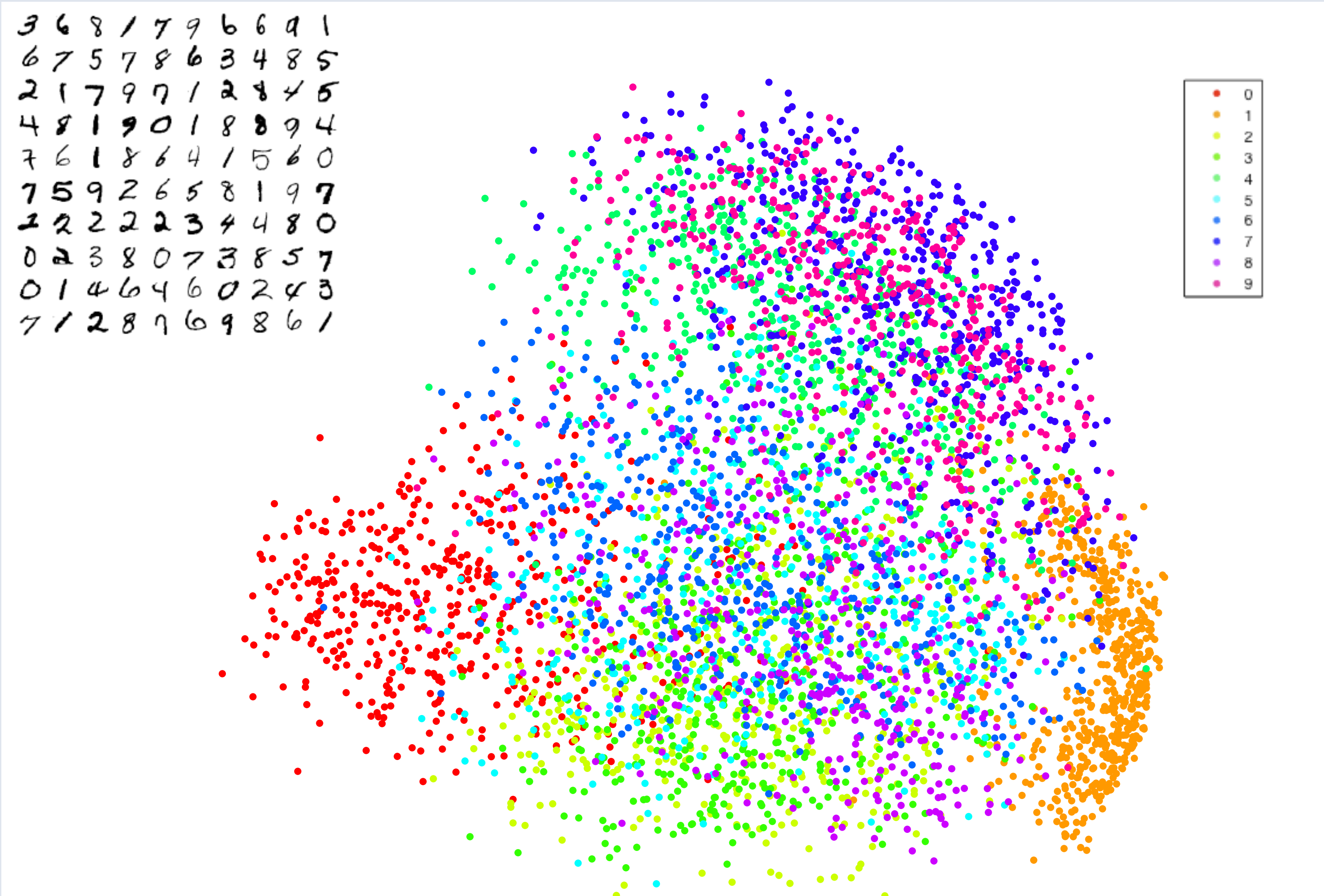
- Kullback-Leibler divergence: $KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$
  - Large p_{ij} modeled by small q_{ij}? Big penalty!
  - Small p_{ij} modeled by large q_{ij}? Not-so-big penalty.
- t-SNE makes sure connected vertices are close together in the embedding!

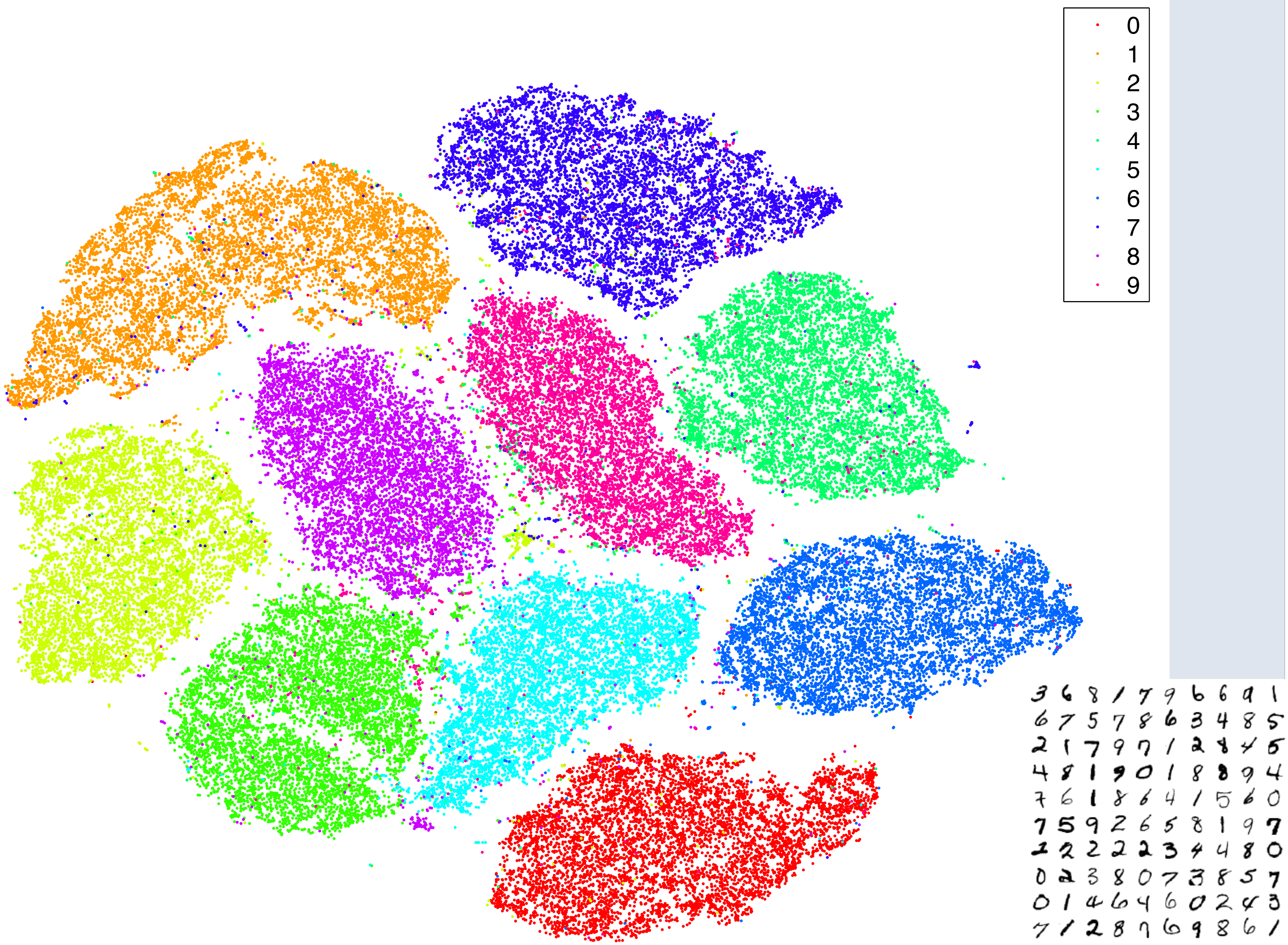
Visualization experiment

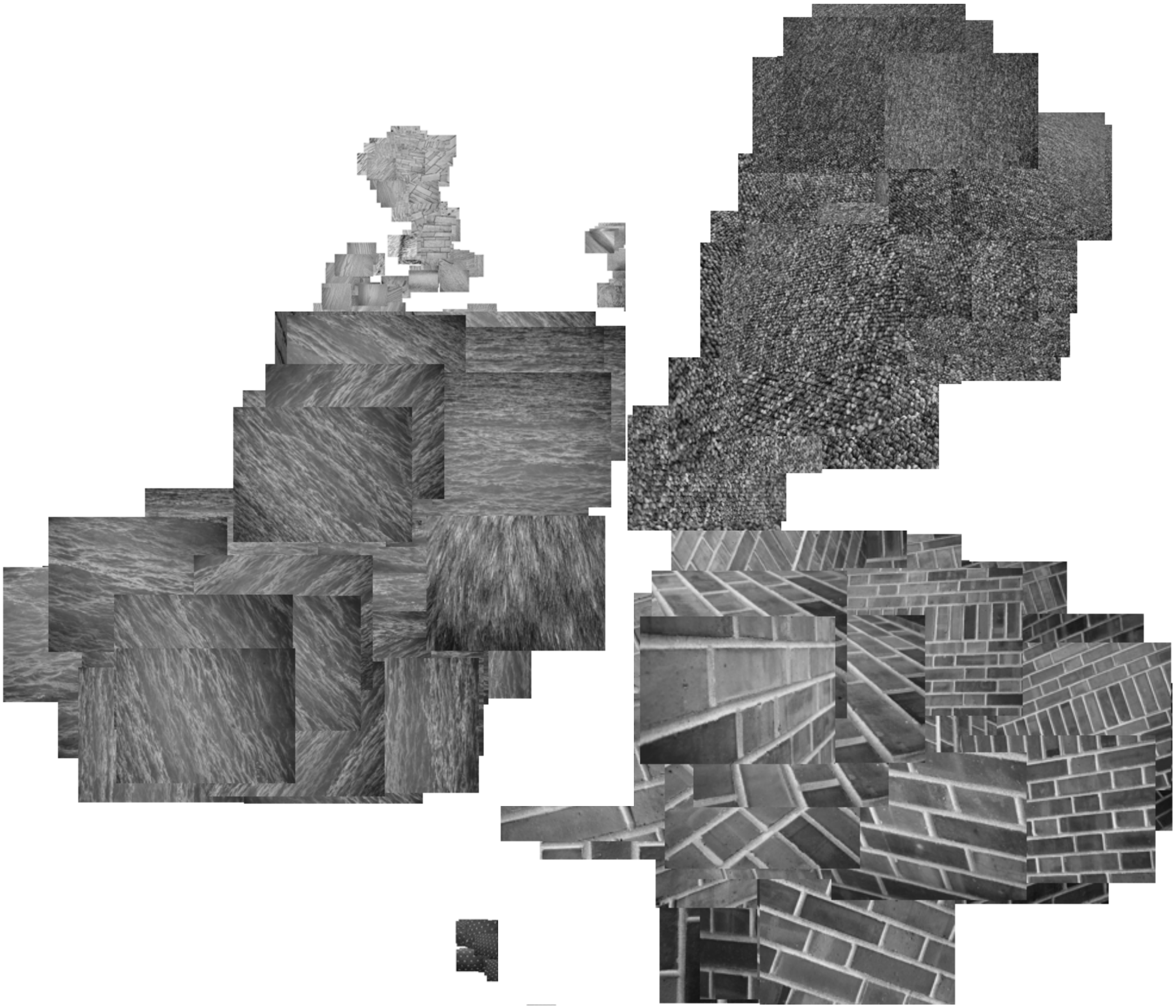
- Suppose we are given the MNIST dataset of handwritten digits

- Can we make a scatter plot that shows some of the structure in this data?
  - Approach 1:
    - Apply PCA on the digits and make a scatter plot in which the data is projected onto its first two principal components
  - Approach 2:
    - Construct a k-nearest neighbor graph (or simply compute a Gaussian kernel) and run t-SNE to learn a 2D embedding that you can show in a scatter plot

Principal Components Analysis

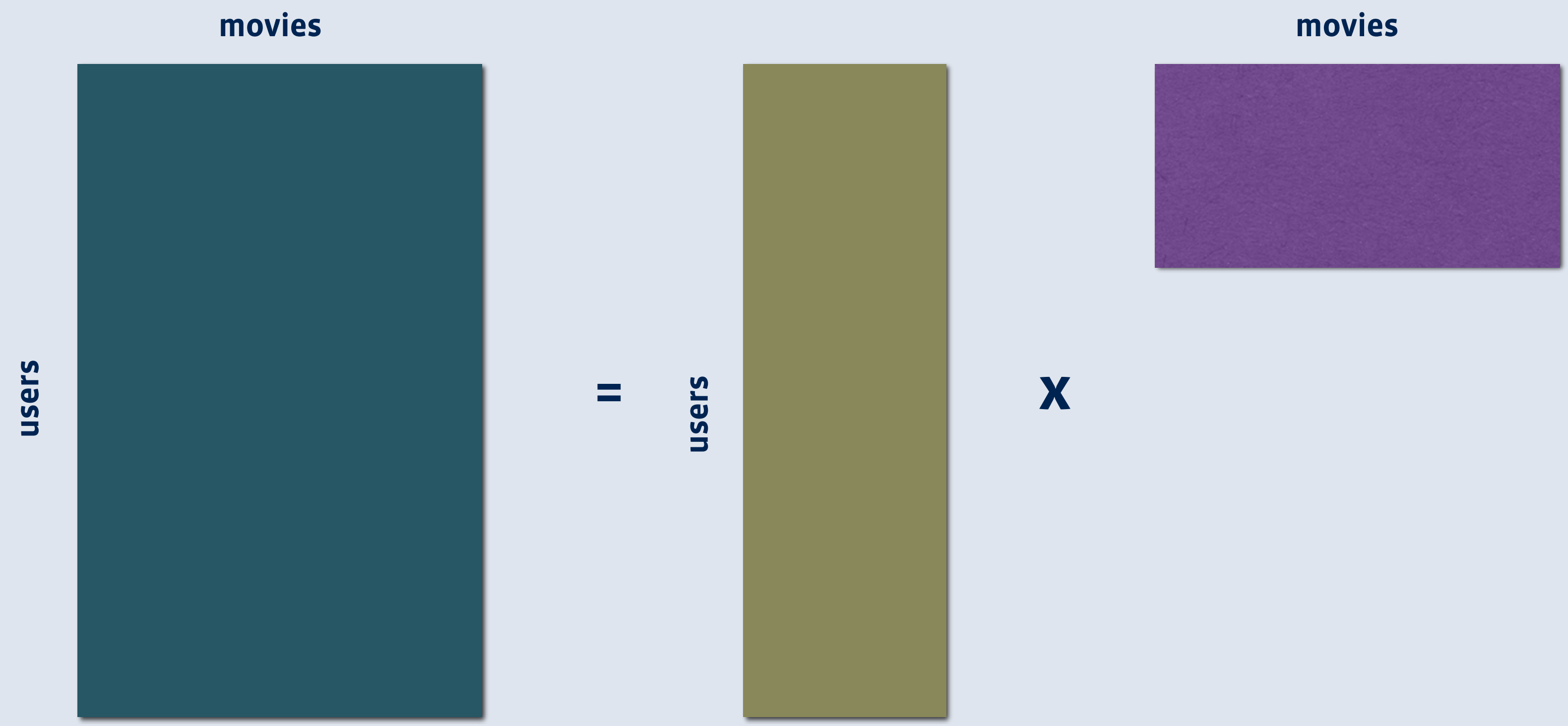


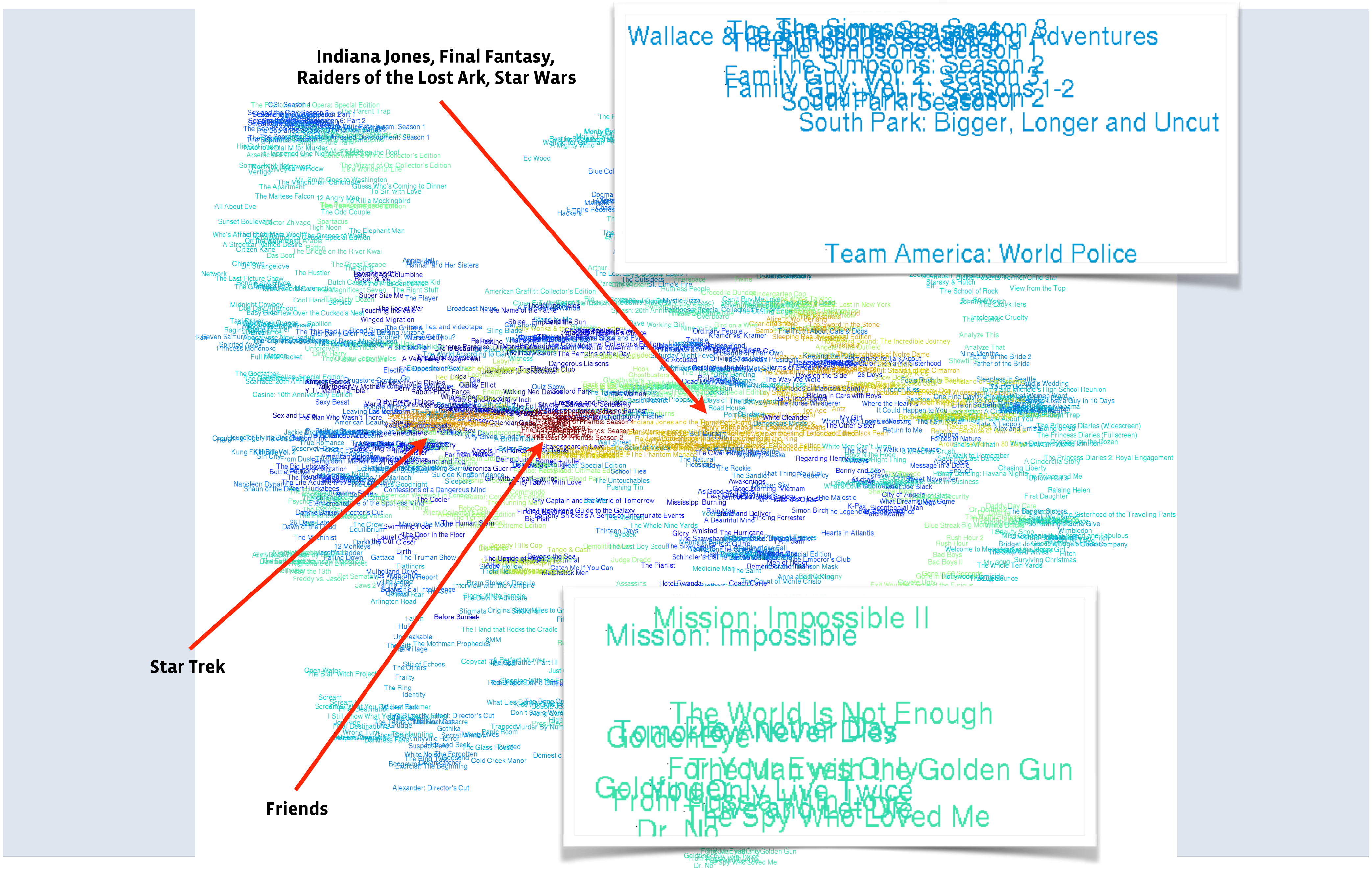

0
1
2
3
4
5
6
7
8
9



Visualizing movies

- Suppose you have a collection of user-movie ratings in a large *rating matrix*
- *Decompose* the rating matrix to obtain *user features* and *movie features*:




Indiana Jones, Final Fantasy,
Raiders of the Lost Ark, Star Wars
Star Trek
Friends
The Simpsons: Season 1
The Simpsons: Season 2
South Park: Bigger, Longer and Uncut
Team America: World Police
Mission: Impossible II
Mission: Impossible
The World Is Not Enough
The Spy Who Loved Me
Dr. No

Conclusions

- Embeddings provide a way for deep learners to work on discrete data
- word2vec is a popular embedding model for word representations
- t-SNE is a popular embedding model for visualization of graphs
- Many other embedding techniques exist, which differ in the exact choice for the loss function that they optimize

References

- Reading material:
  - O. Levy and Y. Goldberg. **Neural Word Embedding as Implicit Matrix Factorization**. Advances in Neural Information Processing 27:2177-2185, 2014 (first two sections).
  - L.J.P. van der Maaten and G.E. Hinton. **Visualizing High-Dimensional Data Using t-SNE**. Journal of Machine Learning Research 9(Nov):2579-2605, 2008.
- Source code:
  - Word2vec: https://code.google.com/archive/p/word2vec/
  - t-SNE: https://lvdmaaten.github.io/tsne/