CS 489/698: Introduction to Natural Language Processing

Lecture 3: Edit Distance and Word Rerepresentations

Instructor: Freda Shi

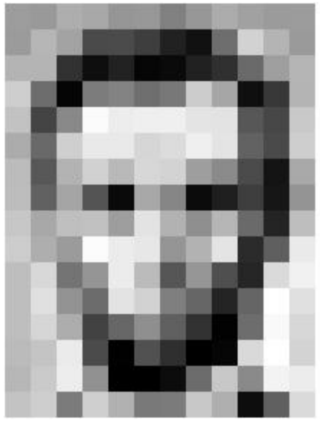
fhs@uwaterloo.ca

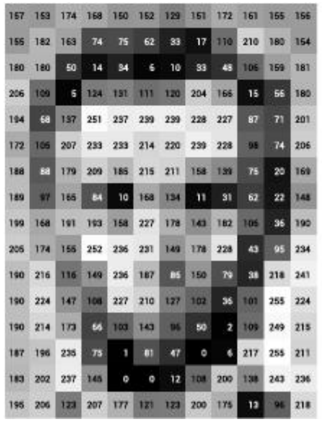
January 12th, 2026

UNIVERSITY OF WATERLOO

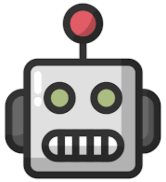
Recap: Digital Representations

How does a human and a computer see?





157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

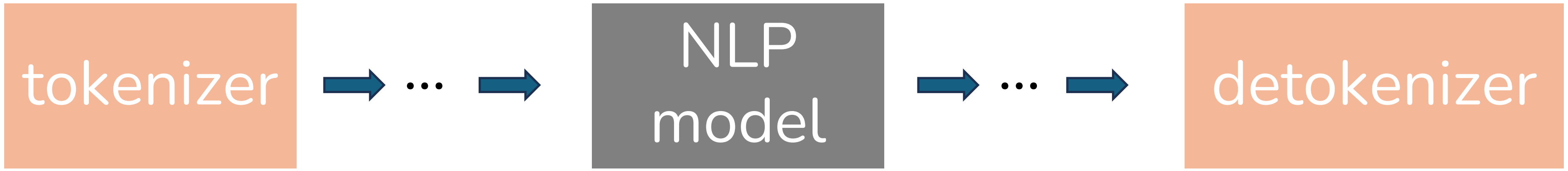


Recap: Tokenization in Modern NLP Systems

Natural first step: convert tokens into numerical indices for further processing.

Can we do better than assigning each word a unique index?

Raw text input



Raw text output

Recap: Byte-Level BPE Tokenization

Consider UTF-8 encoding of “Hello world!”

We will work with the sequence of

48 65 6C 6C 6F 20 57 6F 72 6C 64 21

Base vocabulary: $2^8 = 256$ entries from 00 to FF.

At each step, evaluate frequency of consecutive vocabulary entry pairs, and add a new entry.

Not much different from character-based BPE!

Character	UTF-8 Hex
H	48
e	65
l	6C
l	6C
o	6F
(space)	20
W	57
o	6F
r	72
l	6C
d	64
!	21

Recap: Byte-Level BPE Tokenization

What about non-English characters?

Consider UTF-8 encoding of “Hello 世界”.
(world)

We will work with the sequence of

48 65 6C 6C 6F 20 E4 B8 96 E7 95 8C

Base vocabulary: $2^8 = 256$ entries from 00 to FF.

Character	UTF-8 Hex
H	48
e	65
l	6C
l	6C
o	6F
(space)	20
世	E4 B8 96
界	E7 95 8C

It is possible to have an entry of something like 96 E7, which corresponds to “combination of sub-characters”.

Outline of Today’s Lecture

Edit Distance

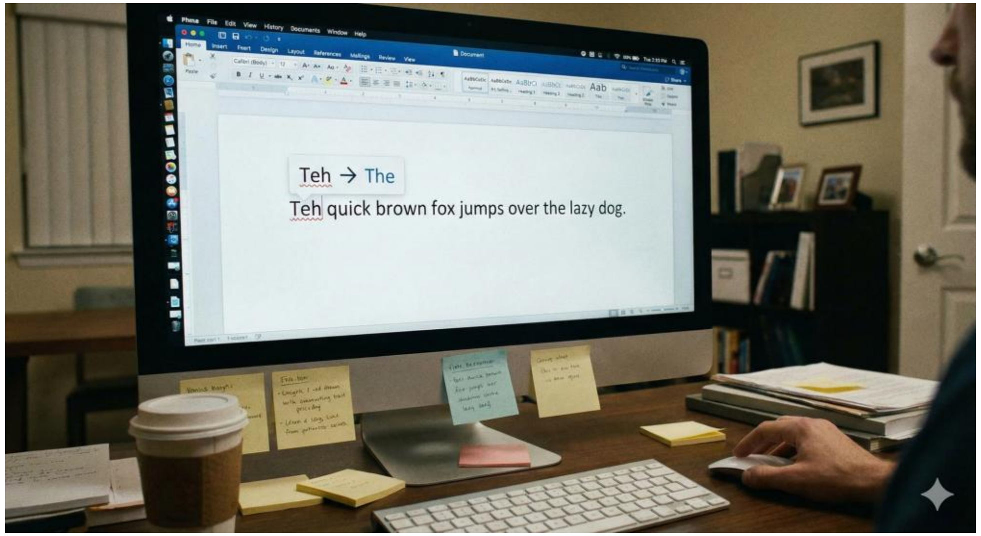
- Comparing similarity of two sequences

Vector representations of tokens/words (i.e., distributional semantics)

- Comparing similarity of two sequences

Similarity between Strings

How to measure the similarity between two strings (e.g., for typo correction)?



[Image generated with Nano Banana Pro]

Computers are good at exact matches, but bad at “almost” matches.

We need to design algorithms to assign a numerical score to measure similarity.

Edit Distance: A Proposal

The minimum number of single-character edits required to change one string A into another B.

Three allowed operations:

- Insertion (add a character)
- Deletion (remove a character)
- Substitution (replace a character with another)

Also known as the Levenshtein Distance.

Examples of Edits

- *teh* → *the*
  - Delete *e at position 2*, add *e at position 3*; or
  - Add *h at position 2*, delete *h at position 4*; or
  - Substitute *e at position 2* with *h*, substitute *h at position 3* with *e*.

- *cat* → *bat*
  - Substitute *c* at position 1 with *b*

- *cat* → *cats*
  - Add *s* at position 4

Unified Algorithmic Solution

- How about *sitting* → *extension*?

Dynamic programming:

Let $f[i][j]$ represent the edit distance (minimal number of edits) between the first i characters of A and the first j characters of B.

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø										
1 s										
2 i										
3 t										
4 t										
5 i										
6 n										
7 g										

Example: *sitting → extension*

$$f[i][j] = \min\begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 ∅	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1									
2 i	2									
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 ∅	**1 e**	**2 x**	**3 t**	**4 e**	**5 n**	**6 s**	**7 i**	**8 o**	**9 n**
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1	1								
2 i	2									
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	**1 e**	**2 x**	**3 t**	**4 e**	**5 n**	**6 s**	**7 i**	**8 o**	**9 n**
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5				
2 i	2									
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 ∅	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5			
2 i	2									
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 ∅	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2									
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2								
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5				
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6			
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

Two counterintuitive solutions!

	0 ∅	**1 e**	**2 x**	**3 t**	**4 e**	**5 n**	**6 s**	**7 i**	**8 o**	**9 n**
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6			
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6	5		
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6	5		
3 t	3									
4 t	4									
5 i	5									
6 n	6									
7 g	7									

Example: *sitting → extension*

$$f[i][j] = \min\begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 Ø	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 Ø	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6	5	6	7
3 t	3	3	3	2	3	4	5	6	6	7
4 t	4	4	4	3	3	4	5	6	7	7
5 i	5	5	5	4	4	4	5	5	6	7
6 n	6	6	6	5	5	4	5	6	7	6
7 g	7	7	7	6	6	5	5	6	7	7

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 ∅	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6	5	6	7
3 t	3	3	3	2	3	4	5	6	6	7
4 t	4	4	4	3	3	4	5	6	7	7
5 i	5	5	5	4	4	4	5	5	6	7
6 n	6	6	6	5	5	4	5	6	7	6
7 g	7	7	7	6	6	5	5	6	7	7

Example: *sitting → extension*

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

Edge cases:

- $f[0][i] = f[i][0] = i$

	0 ∅	1 e	2 x	3 t	4 e	5 n	6 s	7 i	8 o	9 n
0 ∅	0	1	2	3	4	5	6	7	8	9
1 s	1	1	2	3	4	5	5	6	7	8
2 i	2	2	2	3	4	5	6	5	6	7
3 t	3	3	3	2	3	4	5	6	6	7
4 t	4	4	4	3	3	4	5	6	7	7
5 i	5	5	5	4	4	4	5	5	6	7
6 n	6	6	6	5	5	4	5	6	7	6
7 g	7	7	7	6	6	5	5	6	7	7

Dynamic Programming Intuition

$$f[i][j] = \min \begin{cases} f[i-1][j] + 1 & \text{(deletion)} \\ f[i][j-1] + 1 & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise 1

The equation implies doing nothing when $A[i] = B[j]$ when calculating $f[i][j]$.

Why is this correct?

Intuition: in such cases, doing nothing may not be the unique best solution, but it is one of the best.

$f[4][3] = 3$ for *sitting* → *extension*, which could come from $f[3][2]$, or $f[3][3] + 1$.

Extension: Different Operations with Different Costs

The minimum cost of single-character edits required to change one word into the other. Each operation could have a different non-negative cost.

Three allowed operations:

- Insertion (add a character) with cost a.
- Deletion (remove a character) with cost b.
- Substitution (replace a character with another) with cost c.

Extension: Different Operations with Different Costs

The minimum cost of single-character edits required to change one word into the other. Each operation could have a different non-negative cost.

$$f[i][j] = \min \begin{cases} f[i-1][j] + a & \text{(deletion)} \\ f[i][j-1] + b & \text{(insertion)} \\ f[i-1][j-1] + cost & \text{(substitution, or do nothing)} \end{cases}$$

$cost = 0$ if $A[i] = B[j]$, otherwise c

Outline of Today’s Lecture

Edit Distance

- Comparing similarity of two sequences

Vector representations of tokens/words (i.e., distributional semantics)

- Comparing similarity of two sequences

Word Vectors

Until about 2010, in NLP, words meant atomic symbols.

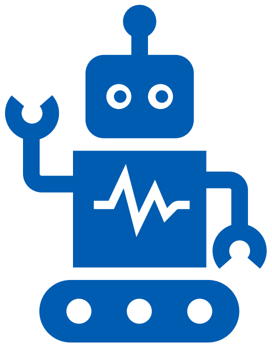
Nowadays, it's natural to think about word vectors when talking about words in NLP. Each word is represented by a vector.

Key idea: similar words are nearby in a good vector space.

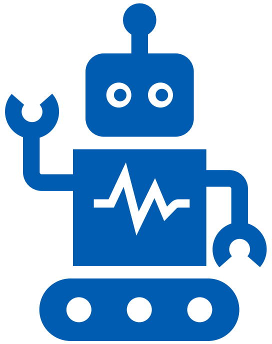
Visualization: https://projector.tensorflow.org/

How Models Represent Words

cat	chef	chicken	civic	cooked	…
↓	↓	↓	↓	↓	
17	91	253	104	5	…



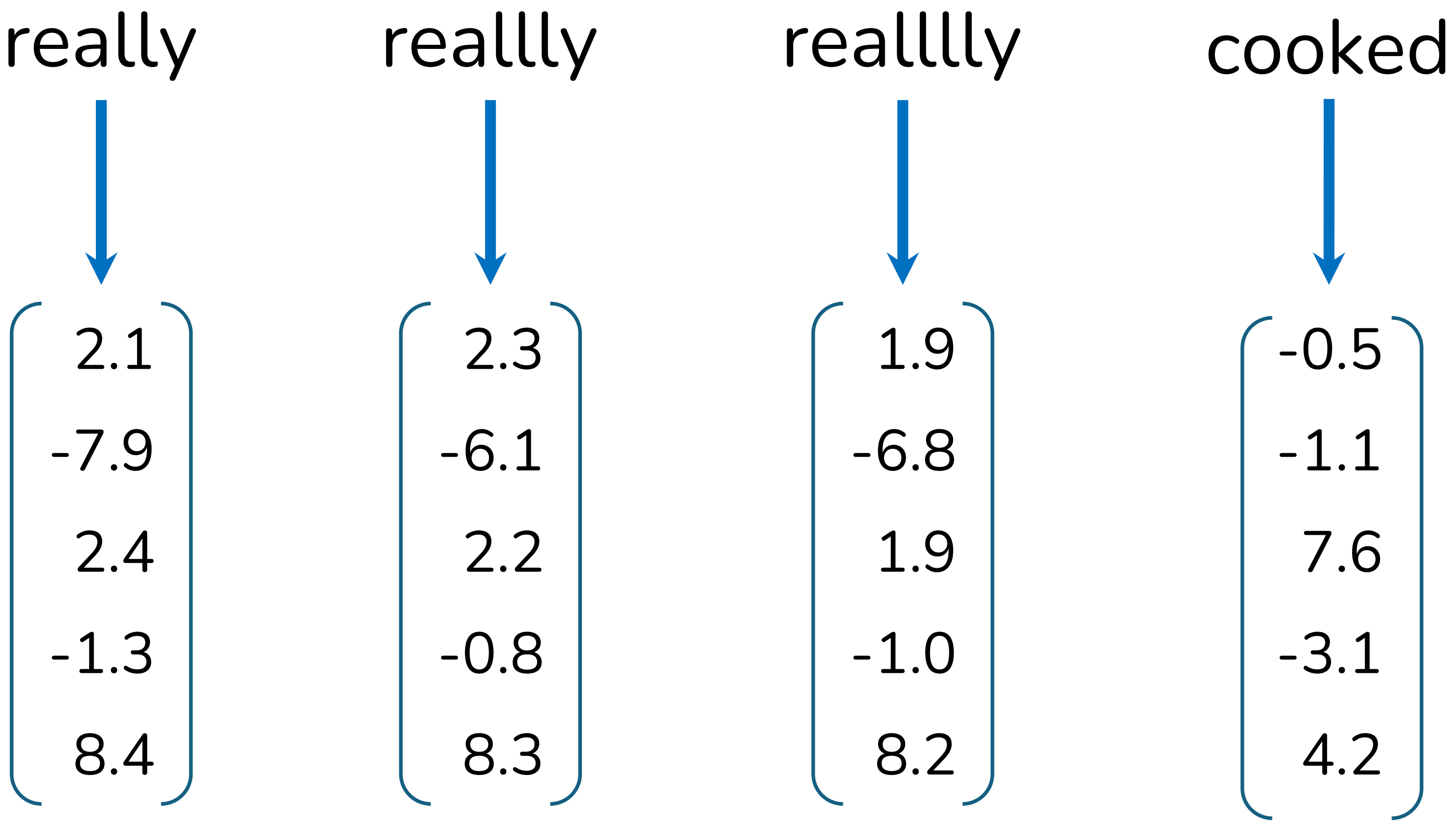
How Models Represent Words



cat	chef	chicken	civic	cooked	...
0.1	-0.1	-0.4	0.1	-0.5	...
7.9	2.1	2.4	0	-1.1	
2.4	3.8	9.7	-1.5	7.6	
-1.3	-0.1	-1.0	2.4	-3.1	
0.5	5.3	3.2	0.2	4.2	

Motivation

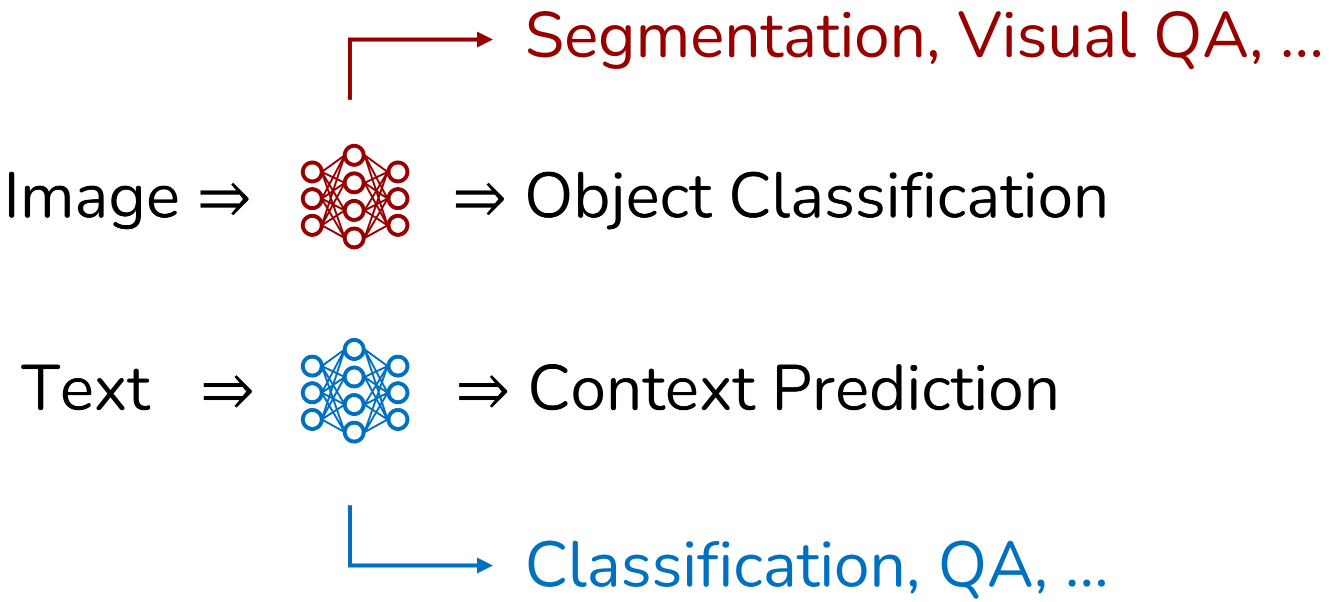
One of the key challenges for NLP is variability of language (multiple forms, same meaning).



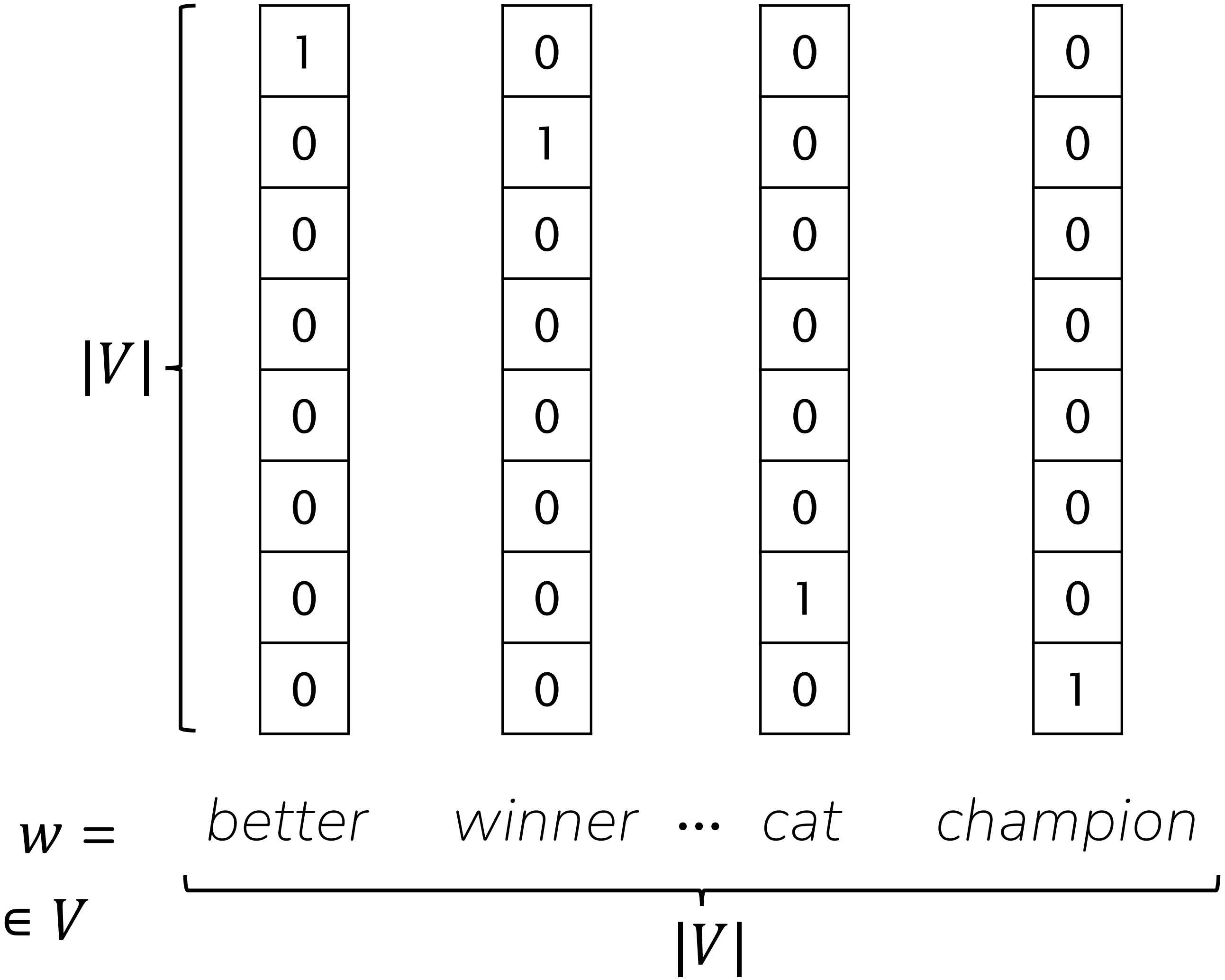
Representation Learning for Engineering

Engineering: these representations are often useful for downstream tasks!

Transfer learning:



How to represent a word



"One-hot" representation of words

$$Rep(w) \in \{0, 1\}^{|V|}$$

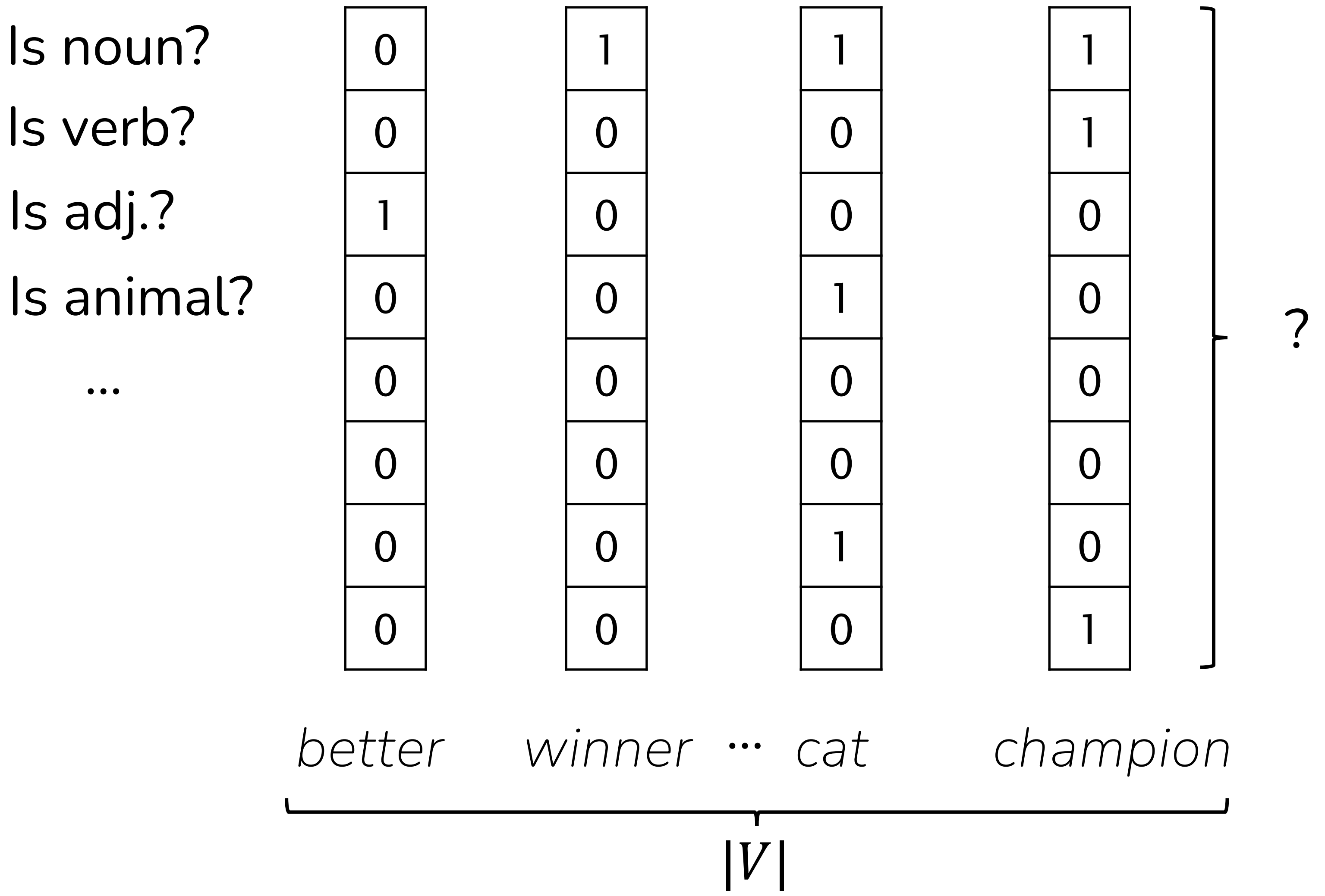
$|V|$ could be very large (e.g., 50K).

Word vectors are orthogonal.

What is an ideal word representation?

- It should probably capture information about usage and meaning:
  - Part of speech tags (noun, verb, adj., adv., etc.)
  - The intended sense
  - Semantic similarities (*winner* vs. *champion*)
  - Semantic relationships (antonyms, hypernyms, etc.)

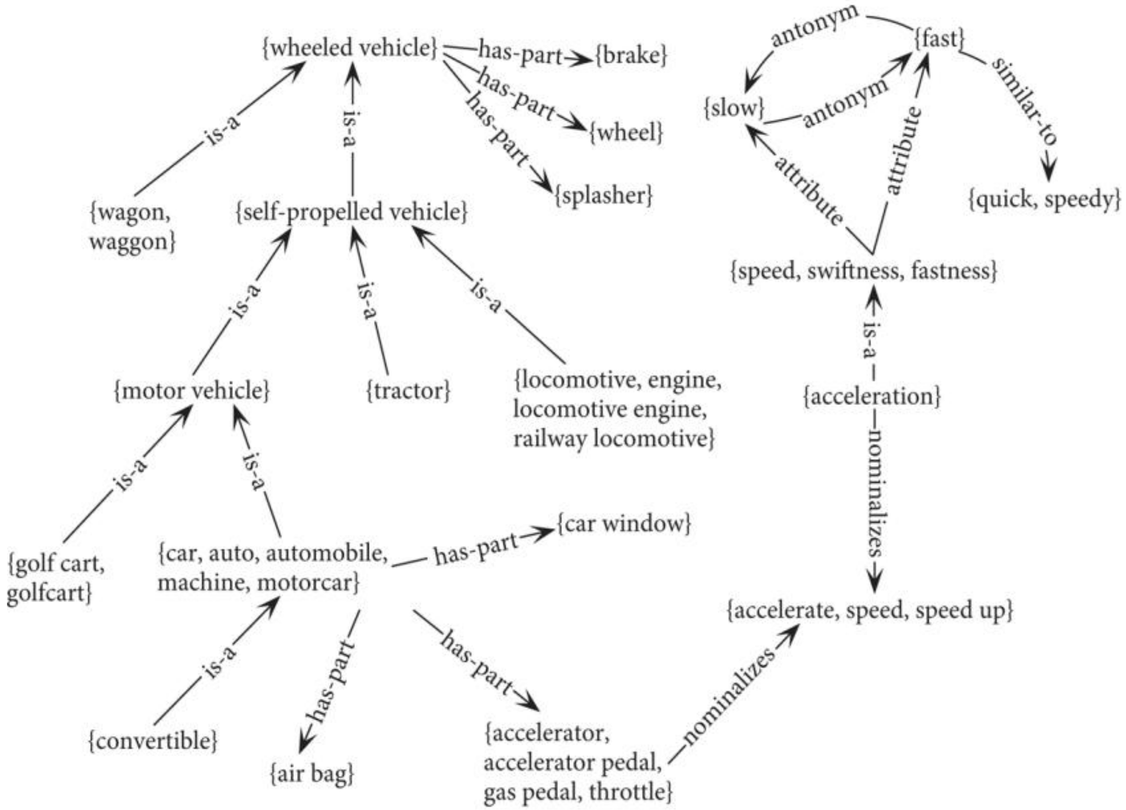
Features?


Is noun?
Is verb?
Is adj.?
Is animal?
...
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 1 0 0 1 0
1 1 0 0 0 0 0 1
?
better
winner
...
cat
champion
$|V|$

Features?

This could extend infinitely.

WordNet

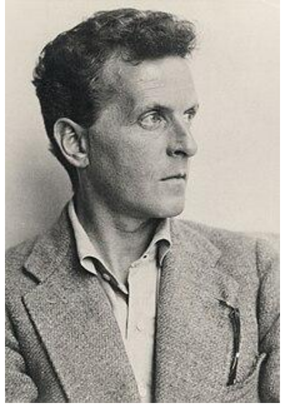


What is an ideal word representation?

- It should probably capture information about usage and meaning:
  - Part of speech tags (noun, verb, adj., adv., etc.)
  - The intended sense
  - Semantic similarities (*winner* vs. *champion*)
  - Semantic relationships (antonyms, hypernyms, etc.)

Distributional Semantics:
How much of this can we capture from context/data alone?

Distributional Hypothesis



"The meaning of a word is its use in the language."

--Ludwig Wittgenstein (1943)

"You shall know a word by the company it keeps."

-- J.R. Firth, A Synopsis of Linguistic Theory (1957)

The use of a word is defined by its contexts
(i.e., the words that appear around it).

Distributional Semantics

Consider a new word: *tezgüino*.

1. A bottle of *tezgüino* is on the table.
2. Everybody likes *tezgüino*.
3. Don't have *tezgüino* before you drive.
4. We make *tezgüino* out of corn.

What do you think *tezgüino* is?

loud
motor oil
tortillas
choices
wine

	1	2	3	4
tezgüino	1	1	1	1
loud	0	0	0	0
motor oil	1	0	0	1
tortillas	0	1	0	1
choices	0	1	0	0
wine	1	1	1	0

Distributional Hypothesis

How can we automate the process of constructing representations of word meaning from its “company”?

First solution: word-word cooccurrence counts.

Counting for Word Vectors

words we are computing vectors for:

context words:

	cat	chef	chicken	civic	cooked	council
the						
cat						
chicken						
city						
cook						

Counting for Word Vectors

… , the club may also employ a **chef** to prepare and cook food items .

… is up to remy , linguini , and the **chef** colette to cook for many people …

… cooking program the cook and the **chef** with simon bryant , who is …

context words:	chef
the	0
cat	0
chicken	0
city	0
cook	0

Counting for Word Vectors

… , the club may also employ a **chef** to prepare and cook food items .

… is up to remy , linguini , and the **chef** colette to cook for many people …

… cooking program the cook and the **chef** with simon bryant , who is …

Window size $w = 1$

context words:

	chef
the	2
cat	0
chicken	0
city	0
cook	0

Counting for Word Vectors

... , the club may also employ a **chef** to prepare and cook food items .

... is up to remy , linguini , and the **chef** colette to cook for many people ...

... cooking program the cook and the **chef** with simon bryant , who is ...

Window size $w = 4$

context words:

	chef
the	3
cat	0
chicken	0
city	0
cook	3

Counting for Word Vectors

words we are computing vectors for:

context words:

	cat	chef	chicken	civic	cooked	council
the	24708	7410	7853	16486	3463	316380
cat	2336	14	23	0	1	36
chicken	23	21	1640	1	181	7
city	116	89	62	943	7	27033
cook	12	113	34	6	34	51

Word Similarity

Once we have word vectors, we can compute word similarities.

Among many ways to define similarity of two vectors, a simple way is the dot product:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}^T \boldsymbol{v} = \sum_i u_i v_i$$

Dot product is large when the vectors have very large (in terms of absolute values) in the same dimensions.

Cosine similarity:

$$\cos(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{|\boldsymbol{u}||\boldsymbol{v}|} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i u_i^2}\sqrt{\sum_i v_i^2}}$$

With dot product as similarity function, let's find the most similar words ("nearest neighbors") to each word:

	cat	chef	chicken	civic	cooked	council
nearest neighbors	council	council	council	council	council	council
	cat	cat	cat	cat	cat	cat
	civic	civic	civic	civic	civic	civic
	chicken	chicken	chicken	chicken	chicken	chicken
	chef	chef	chef	chef	chef	chef
	cooked	cooked	cooked	cooked	cooked	cooked

Counting for Word Vectors

words we are computing vectors for:

context words:	cat	chef	chicken	civic	cooked	council
the	24708	7410	7853	16486	3463	316380
cat	2336	14	23	0	1	36
chicken	23	21	1640	1	181	7
city	116	89	62	943	7	27033
cook	12	113	34	6	34	51

Now use cosine similarity:

	cat	chef	chicken	civic	cooked	council
	cat	chef	chicken	civic	cooked	council
nearest neighbors	chef	civic	cooked	council	chef	civic
	cooked	cooked	chef	chef	civic	chef
	civic	council	civic	cooked	council	cooked
	council	cat	council	cat	cat	cat
	chicken	chicken	cat	chicken	chicken	chicken

Issues with Counting-Based Vectors

Raw frequency count is probably a bad representation!

Counts of common words are very large, but not very useful

- “the”, “it”, “they”
- Not very informative

There are many ways proposed for improving raw counts.

- Removing “stop words”.
- Down-weight less informative words.

TF-IDF

TF (Term Frequency) - IDF (Inverse Document Frequency)

- Information Retrieval (IR) workhorse!
- A common baseline model
- Sparse vectors
- Words are represented by (a simple function of) the counts of nearby words

TF-IDF

Consider a matrix of word counts across documents: term-document matrix.

document

word

$$\texttt{tf}(w, d) = \#\text{ of times word } w \text{ appears in document } d$$

$$\texttt{idf}(w) = \log\left(\frac{\#\text{ of documents}}{\#\text{ of documents in which word } w \text{ occurs}}\right)$$

$$\texttt{tf-idf}(w, d) = \texttt{tf}(w, d) \cdot \texttt{idf}(w)$$

Term Frequency

$$\text{tf}_{t,d} = \text{count}(t,d)$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

word vector

bag-of-words (document representation)

Inverse Document Frequency

IDF from 37 Shakespear plays:

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

$$\text{tf}_{t,d} = \text{count}(t,d)$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

TF-IDF

IDF from 37 Shakespear plays:

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Pointwise Mutual Information (PMI)

Consider two random variables, X and Y.

Do two events $X = x$ and $Y = y$ occur together more often than if they were independent?

$$\mathrm{PMI}(x, y) = \log_2 \frac{p_{X,Y}(x, y)}{p_X(x) p_Y(y)}$$

If they are independent, $\mathrm{PMI} = 0$.

PMI for Word Vectors

For words X and its context Y, each probability can be estimated using counts we already computed.

$$\mathrm{PMI}(x, y) = \log_2 \frac{p_{X,Y}(x, y)}{p_X(x) p_Y(y)}$$

$$p_{X,Y}(x, y) = \frac{\#(x, y)}{N - 1}, p_X(x) = \frac{\sum_z \#(x, z)}{N}, p_Y(y) = \frac{\sum_z \#(z, y)}{N}$$

N: total count of words

$\#(\cdot,\cdot)$: co-occurrence count of two words

Top co-occurrence counts with “chicken”

14464	,	1525	or	508	pork
7853	the	1225	for	500	meat
6276	and	1061	‘s	481	be
5931	.	940	fried	479	he
5213	a	906	on	452	such
3963	of	889	was	445	his
3282	in	869	that	417	at
2520	to	828	are	405	soup
2438	"	777	by	389	made
2339	is	746	from	384	rice
2127	with	710	it	375	but
1818	(	600	beef	350	has
1745	)	590	which	330	fish
1640	chicken	557	also	325	other
1594	as	531	an	318	this

Words with largest PMI with "chicken"

10.2	fried	7.0	robot	6.1	pig
9.7	chicken	6.9	burger	6.0	breeds
9.3	pork	6.8	recipe	6.0	vegetable
9.0	beef	6.6	vegetables	6.0	potato
8.7	soup	6.6	potatoes	5.9	goose
7.8	sauce	6.6	goat	5.9	dixie
7.7	curry	6.5	eggs	5.9	kung
7.6	cooked	6.4	cow	5.9	pie
7.5	lamb	6.4	pizza	5.8	menu
7.4	dish	6.4	rice	5.8	steamed
7.3	shrimp	6.3	ribs	5.8	tastes
7.3	egg	6.3	tomatoes	5.7	beans
7.2	sandwich	6.2	cheese	5.7	butter
7.2	dishes	6.2	duck	5.7	barn
7.2	meat	6.1	chili	5.7	breed

Positive PMI (PPMI)

Some have found benefit by truncating PMI at 0 (“positive PMI”).

$$\mathrm{PPMI}(x, y) = \max\big(0, \mathrm{PMI}(x, y)\big)$$

Negative PMI: words occur together less than we would expect, i.e., they are anticorrelated.

These anticorrelation may need more data to reliably estimate.

However, negative PMIs do seem reasonable.

Largest PMIs:		PMIs close to zero:		Smallest PMIs:	
10.2	fried	0.003	climbed	-4.6	users
9.7	chicken	0.003	detailing	-4.6	data
9.3	pork	0.002	turkish	-4.7	discussion
9.0	beef	0.002	oaks	-4.7	museum
8.7	soup	0.001	productivity	-4.7	below
7.8	sauce	0.000	swing	-4.8	editors
7.7	curry	-0.001	structures	-4.8	railway
7.6	cooked	-0.001	thirteenth	-4.8	committee
7.5	lamb	-0.001	commentators	-4.8	elected
7.4	dish	-0.001	palmer	-4.9	championship
7.3	shrimp	-0.002	obstacles	-5.0	archive
7.3	egg	-0.003	horns	-5.3	edits
7.2	sandwich	-0.003	burning	-6.1	deletion

Word2Vec

Learning representations with neural networks

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

[Mikolov et al., 2013]

Word2Vec

Instead of counting, train a classifier (neural network) to **predict** context (e.g. neighboring words).

- Training is **self-supervised**: no annotated data required, just raw text.
- Word embeddings learned via backpropagation.

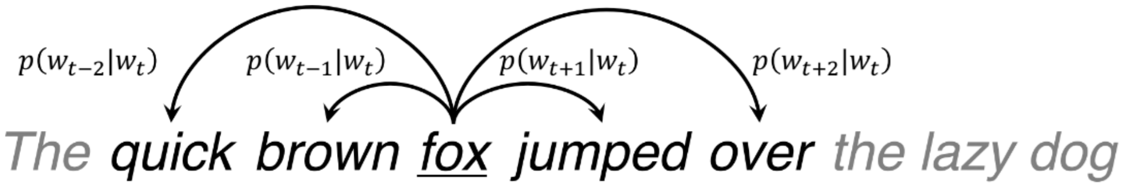
Word2Vec: Training Objectives

- CBOW (Continuous Bag-of-Words): learning representations that predict a word given "a bag of context" (many-to-one prediction).

$$P(w_t \mid w_{t+1}, \ldots, w_{t+k}, w_{t-1}, \ldots, w_{t-k})$$

- Skipgram: learning representations that predict the context given a word

$$P(w_{t+1}, \ldots, w_{t+k}, w_{t-1}, \ldots, w_{t-k} \mid w_t) = P(w_{t+1} \mid w_t) \ldots P(w_{t-k} \mid w_t)$$



Skipgram

Randomly initialized (to be learned with backpropagation).

$$p_\theta(\text{out} \mid \text{input}) = \frac{\exp(u_{\text{out}} \cdot w_{\text{input}})}{\sum_{v \in V} \exp(u_v \cdot w_{\text{input}})}$$

Just a (log) linear model!

softmax

a	1.2	-0.1	0.3	...	0.1
aardvark	0.2	0.7	-0.4	...	1.1
able	-0.7	0.5	0.6	...	-0.8
are	0.1	0.9	0.8	...	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	...	0.4

W

a	2.1	-0.5	1.3	...	1.4
aardvark	-0.4	-0.7	0.5	...	0.1
able	0.2	0.1	0.4	...	-0.7
are	0.5	0.8	0.1	...	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	...	0.6

U

$$\theta = \{W, U\}$$

$W \; : \; V \times d$ input embedding matrix

$U \; : \; V \times d$ output embedding matrix

Skipgram

$$p_\theta(\text{out} \mid \text{input}) = \frac{\exp(u_{\text{out}} \cdot w_{\text{input}})}{\sum_{v \in V} \exp(u_v \cdot w_{\text{input}})}$$

a	1.2	-0.1	0.3	...	0.1
aardvark	0.2	0.7	-0.4	...	1.1
able	-0.7	0.5	0.6	...	-0.8
are	0.1	0.9	0.8	...	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	...	0.4

$$W$$

a	2.1	-0.5	1.3	...	1.4
aardvark	-0.4	-0.7	0.5	...	0.1
able	0.2	0.1	0.4	...	-0.7
are	0.5	0.8	0.1	...	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	...	0.6

$$U$$

`it is a far , far better rest that I go to , than I have ever known`

$$L_t = -\log p_\theta(x_{t-2} \mid x_t) - \log p_\theta(x_{t-1} \mid x_t)$$
$$- \log p_\theta(x_{t+1} \mid x_t) - \log p_\theta(x_{t+2} \mid x_t)$$

Skipgram

$$p_\theta(\text{out} \mid \text{input}) = \frac{\exp(u_{\text{out}} \cdot w_{\text{input}})}{\sum_{v \in V} \exp(u_v \cdot w_{\text{input}})}$$

a	1.2	-0.1	0.3	...	0.1
aardvark	0.2	0.7	-0.4	...	1.1
able	-0.7	0.5	0.6	...	-0.8
are	0.1	0.9	0.8	...	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	...	0.4

W

a	2.1	-0.5	1.3	...	1.4
aardvark	-0.4	-0.7	0.5	...	0.1
able	0.2	0.1	0.4	...	-0.7
are	0.5	0.8	0.1	...	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	...	0.6

U

`it is a far , far better rest that I go to , than I have ever known`

$$L_t = -\log p_\theta(x_{t-2} \mid x_t) - \log p_\theta(x_{t-1} \mid x_t) - \log p_\theta(x_{t+1} \mid x_t) - \log p_\theta(x_{t+2} \mid x_t)$$

Skipgram

$$p_\theta(\text{out} \mid \text{input}) = \frac{\exp(u_{\text{out}} \cdot w_{\text{input}})}{\sum_{v \in V} \exp(u_v \cdot w_{\text{input}})}$$

W

a	1.2	-0.1	0.3	...	0.1
aardvark	0.2	0.7	-0.4	...	1.1
able	-0.7	0.5	0.6	...	-0.8
are	0.1	0.9	0.8	...	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	...	0.4

U

a	2.1	-0.5	1.3	...	1.4
aardvark	-0.4	-0.7	0.5	...	0.1
able	0.2	0.1	0.4	...	-0.7
are	0.5	0.8	0.1	...	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	...	0.6

it is a far , far better rest that I go to , than I have ever known

$$L_t = -\log p_\theta(x_{t-2} \mid x_t) - \log p_\theta(x_{t-1} \mid x_t) - \log p_\theta(x_{t+1} \mid x_t) - \log p_\theta(x_{t+2} \mid x_t)$$

CBOW

Use the context to predict the center word

a	1.2	-0.1	0.3	…	0.1
aardvark	0.2	0.7	-0.4	…	1.1
able	-0.7	0.5	0.6	…	-0.8
are	0.1	0.9	0.8	…	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	…	0.4

W

a	2.1	-0.5	1.3	…	1.4
aardvark	-0.4	-0.7	0.5	…	0.1
able	0.2	0.1	0.4	…	-0.7
are	0.5	0.8	0.1	…	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	…	0.6

U

$$p_\theta(x_t \mid x_{t-w}, \ldots, x_{t+w}) \propto \exp\left(u_{x_t} \cdot \frac{1}{2w} \sum_{k \in \{-w,\ldots,-1,1,w\}} w_{x_{t+k}}\right)$$

`it is a far , far better rest that I go to , than I have ever known`

$$L_t = -\log p_\theta(x_t \mid x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2})$$

CBOW

Use the context to predict the center word

a	1.2	-0.1	0.3	...	0.1
aardvark	0.2	0.7	-0.4	...	1.1
able	-0.7	0.5	0.6	...	-0.8
are	0.1	0.9	0.8	...	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	...	0.4

W

a	2.1	-0.5	1.3	...	1.4
aardvark	-0.4	-0.7	0.5	...	0.1
able	0.2	0.1	0.4	...	-0.7
are	0.5	0.8	0.1	...	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	...	0.6

U

$$p_\theta(x_t \mid x_{t-w}, \ldots, x_{t+w}) \propto \exp\left(u_{x_t} \cdot \frac{1}{2w} \sum_{k \in \{-w, \ldots, -1, 1, w\}} w_{x_{t+k}} \right)$$

`it is a far , far better rest that I go to , than I have ever known`

$$L_t = -\log p_\theta(x_t \mid x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2})$$

Skipgram with negative sampling

$$p_\theta(\text{out} \mid \text{input}) = \frac{\exp(u_{\text{out}} \cdot w_{\text{input}})}{\sum_{v \in V} \exp(u_v \cdot w_{\text{input}})}$$

a	1.2	-0.1	0.3	...	0.1
aardvark	0.2	0.7	-0.4	...	1.1
able	-0.7	0.5	0.6	...	-0.8
are	0.1	0.9	0.8	...	0.7
⋮			⋮		
zyzzyva	0.3	-0.2	0.7	...	0.4

W

a	2.1	-0.5	1.3	...	1.4
aardvark	-0.4	-0.7	0.5	...	0.1
able	0.2	0.1	0.4	...	-0.7
are	0.5	0.8	0.1	...	0.4
⋮			⋮		
zyzzyva	-0.3	0.3	0.2	...	0.6

U

- Vocabulary size V: 50K – 30M
- Very expensive O(|V|)

Skipgram with negative sampling

Treat the target word and a neighboring context word as positive examples.

Randomly sample other words outside of context to get negative samples.

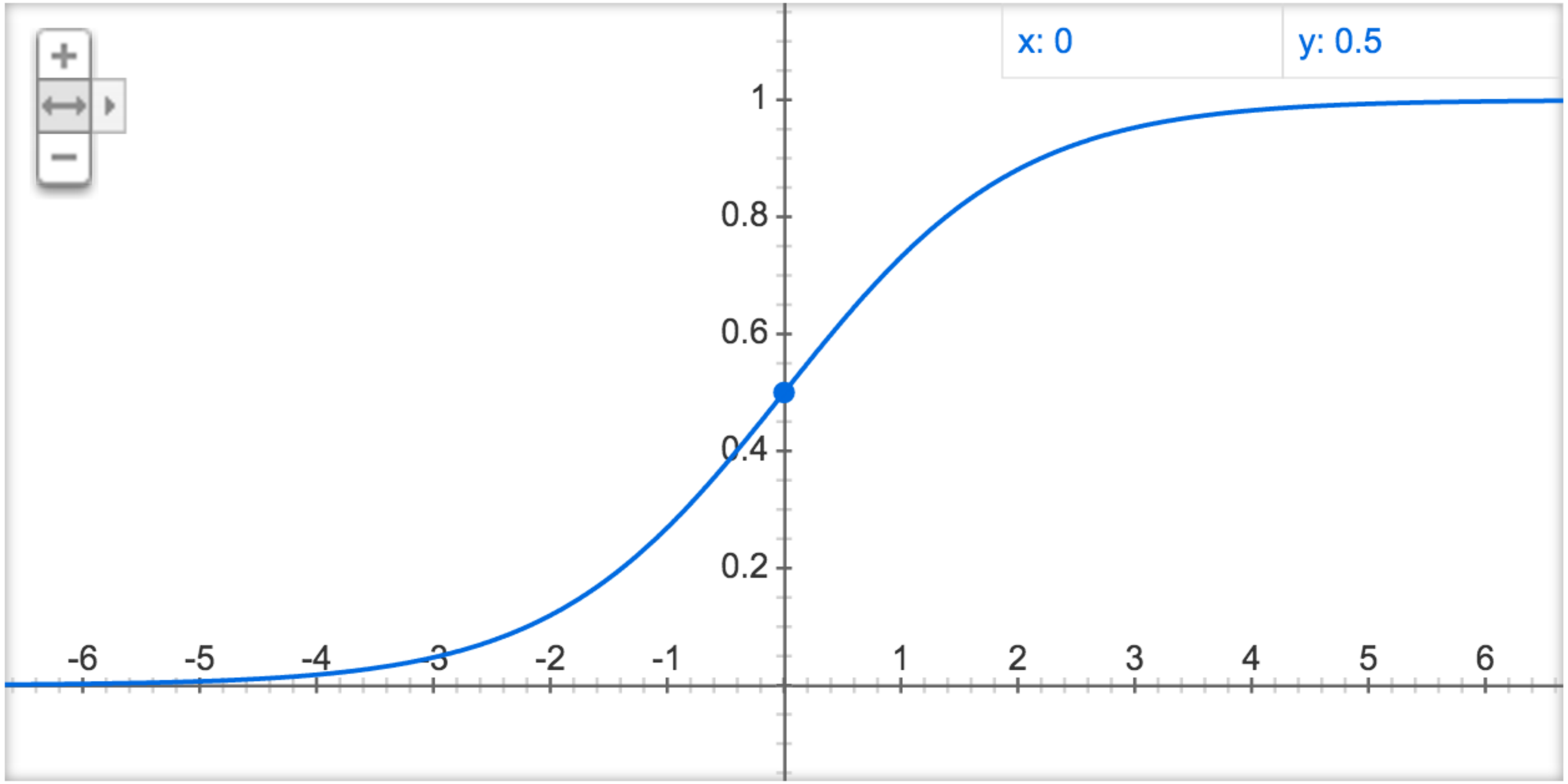
Learn to distinguish between positive and negative samples with a binary classifier.

$$\log p((x,y) \text{ is a true pair}) + \sum_{k \in C} \log p((x,k) \text{ is a negative pair})$$

C = Negative Samples

$$p((x,c) \text{ is a true pair}) = \sigma(u_c \cdot w_x) = \frac{1}{1+\exp(-u_c \cdot w_x)}$$

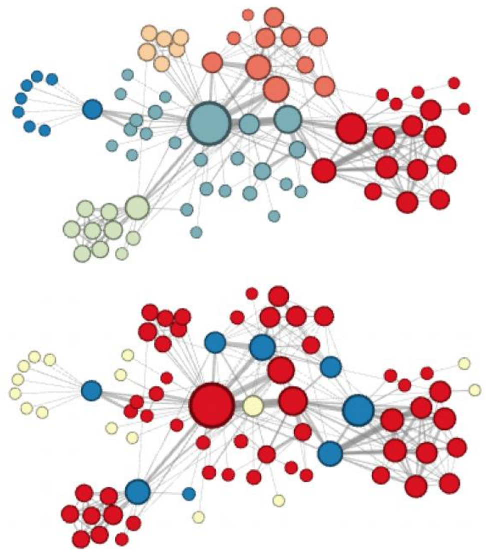
(logistic) sigmoid: $\sigma(x) = \dfrac{1}{1 + \exp\{-x\}}$



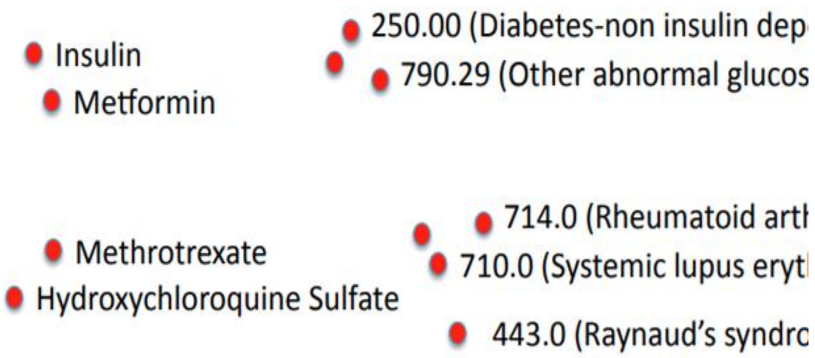
Extensions

"You shall know ~~a word~~ by the company it keeps"

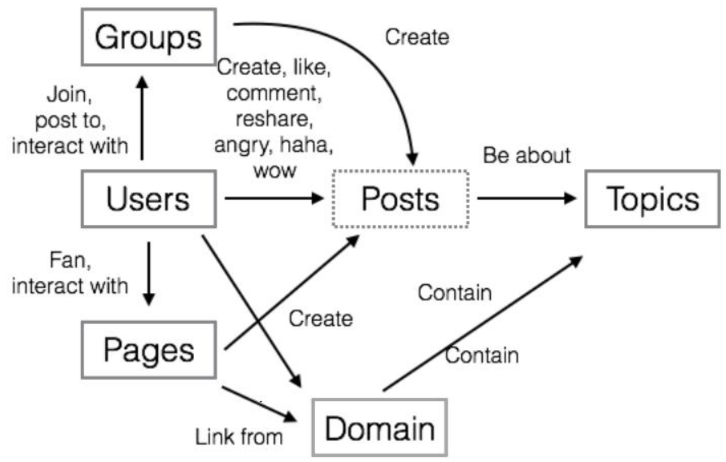
anything?



Node2Vec
[Grover and Leskovec 2016]



Concept2Vec
[Choi et al. 2016]



World2Vec
[Facebook AI Research]

Next

Building a simple text classifier

Ludwig the Cat