



# COMP 336I Natural Language Processing

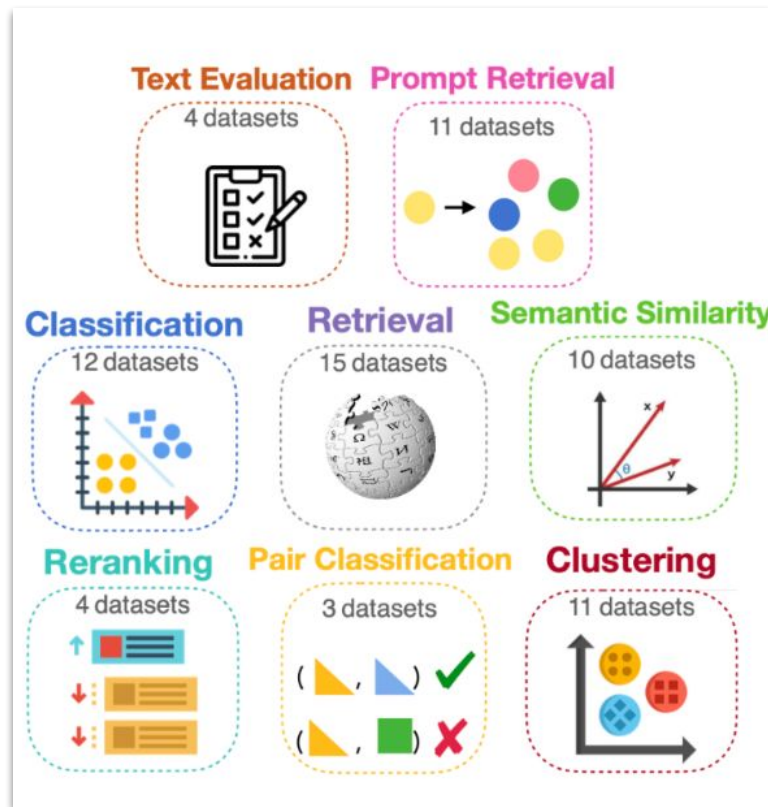
## Lecture 4: Word Embeddings

Spring 2024

# Lecture plan

- Recap of text classification
- Why word embeddings?
- What are word embeddings?
- Word vectors and word2vec

# Why word embeddings?



# Text embeddings for retrieval



Language models without retrieval

# Text embeddings for retrieval



**You**

who is the president of argentina?



**ChatGPT**

I did a [quick search](#) for more information and here's what I discovered.

The current President of Argentina as of 2024 is Javier Milei.

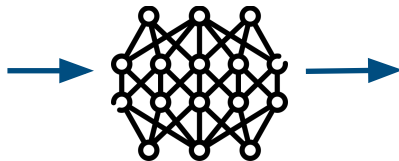


Retrieval-augmented language models

# Text embeddings for retrieval

ChatGPT

Who is the president of  
Argentina?



Javier Milei

**Retrieve**  
*update-to-date info*



Datastore

- + external knowledge during inference
- + update-to-date info
- + domain specific or private data

# Text embeddings for retrieval

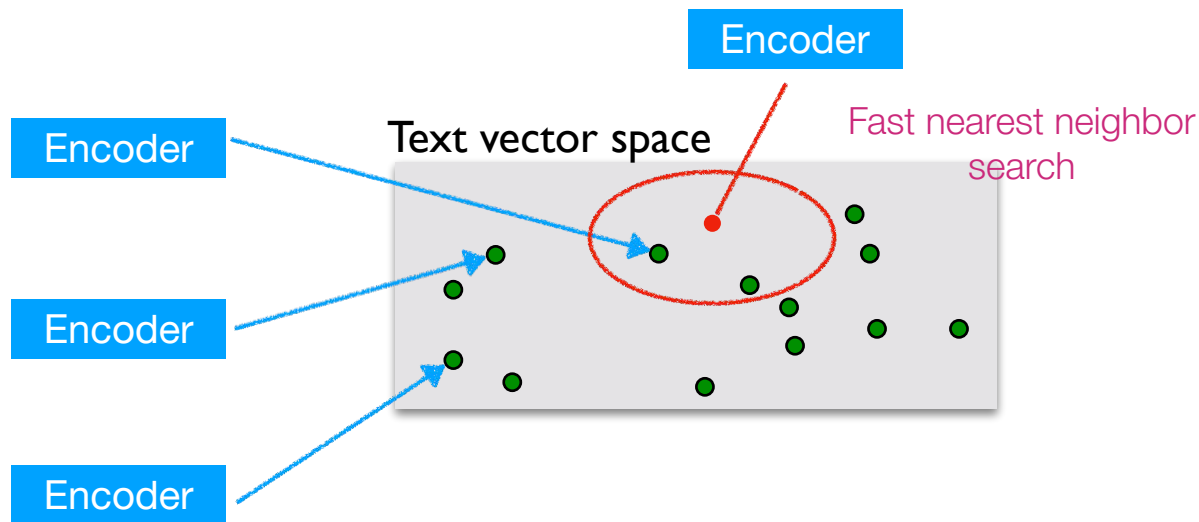


Voldemort cried, "Avada Kedavra!" A jet of green light issued ...from ...

Voldemort's wand just as a jet of red light ...

"The Boy Who Lived." He saw the mouth move and a flash of green ...

Harry felt Greenback collapse... on the floor as a jet of



# What are word embeddings?

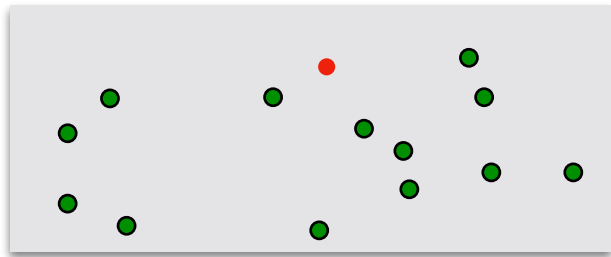
- Word embeddings: Vector representations of word meaning
- The big idea: model of meaning focusing on similarity

Each word = a vector

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Similar words are “**nearby**”  
in the vector space”





# What are word embeddings?

- Which among these models is capable of producing word embeddings?
  - word2vec
  - BERT
  - T5
  - GPT4

# How do we represent words in NLP models?

- n-gram models

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i | w_{i-1})$$

$$P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}$$

Each word is just a string  
or indices  $w_i$  in the  
vocabulary list

cat = the 5th word in  $V$

dog = the 10th word in  $V$

cats = the 118th word in  $V$


- Naive Bayes

$$\hat{P}(w_i | c_j) = \frac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V} \text{Count}(w, c_j) + \alpha|V|}$$

# How do we represent words in NLP models?

- Logistic regression

Var	Definition	Value in Fig. 5.2
$x_1$	count(positive lexicon) $\in$ doc)	3
$x_2$	count(negative lexicon) $\in$ doc)	2
$x_3$	$\begin{cases} 1 & \text{if "no" } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$	1
$x_4$	count(1st and 2nd pronouns $\in$ doc)	3
$x_5$	$\begin{cases} 1 & \text{if "!" } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$	0
$x_6$	log(word count of doc)	$\ln(64) = 4.15$

string match 

# What do words mean?

- **Synonyms:** couch/sofa, car/automobile, filbert/hazelnut
- **Antonyms:** dark/light, rise/fall, up/down
- Some words are not synonyms but they share some element of meaning
  - cat/dog, car/bicycle, cow/horse
- Some words are not similar but they are **related**
  - coffee/cup, house/door, chef/menu
- **Affective meanings or connotations:**

**valence:** the pleasantness of the stimulus

**arousal:** the intensity of emotion provoked by the stimulus

**dominance:** the degree of control exerted by the stimulus

vanish	disappear	9.8
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

SimLex-999

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

(Osgood et al., 1957)

# Lexical resources

## WordNet Search - 3.1

[- WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options: (Select option to change) ▼ Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

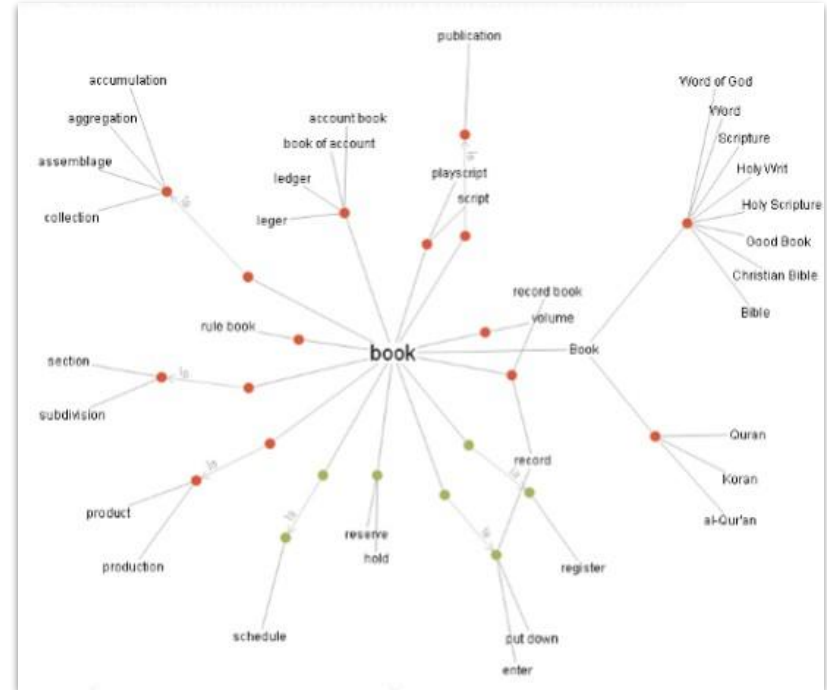
Display options for sense: (gloss) "an example sentence"

**Noun**

- S: (n) **mouse** (any of numerous small rodents typically resembling diminutive rats having pointed snouts and small ears on elongated bodies with slender usually hairless tails)
- S: (n) **shiner, blue eye, mouse** (a swollen bruise caused by a blow to the eye)
- S: (n) **mouse** (person who is quiet or timid)
- S: (n) **mouse, computer mouse** (a hand-operated electronic device that controls the coordinates of a cursor on your computer screen as you move it around on a pad; on the bottom of the device is a ball that rolls on the surface of the pad) *"a mouse takes much more room than a trackball"*

## Verb

- **S: (v) sneak, mouse, creep, pussyfoot** (to go stealthily or furtively) *"..stead of sneaking around spying on the neighbor's house"*
- **S: (v) mouse** (manipulate the mouse of a computer)



(-) Huge amounts of human labor to create and maintain

# Lexical resources

**WordNet Search - 3.1**  
- [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Word to search for:

Display Options:

**Your search did not return any results.**

(-) hard to keep update-to-date

# Distributional hypothesis



- “The meaning of a word is its use in the language”
- “If A and B have almost identical environments we say that they are synonyms.”
- “You shall know a word by the company it keeps”

[Wittgenstein PI 43]

[Harris 1954]

[Firth 1957]

# Distributional hypothesis

**Distributional hypothesis:** words that occur in similar **contexts** tend to have similar meaning



J.R.Firth 1957

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

One of the most successful ideas of modern statistical NLP!

When a word  $w$  appears in a text, its context is the set of words that appear nearby (within a fixed-size window).

*...government debt problems turning into **banking** crises as happened in 2009...*

*...saying that Europe needs unified **banking** regulation to replace the hodgepodge...*

*...India has just given its **banking** system a shot in the arm...*

These context words will represent “*banking*”.



# Distributional hypothesis

## “Ongchoi”

- Ongchoi is delicious sautéed with garlic
- Ongchoi is superb over rice
- Ongchoi leaves with salty sauces

Q:What do you think ‘Ongchoi’ means?

- A) a savory snack
- B) a green vegetable
- C) an alcoholic beverage
- D) a cooking sauce

# Distributional hypothesis

## “Ongchoi”

- Ongchoi is delicious sautéed with garlic
- Ongchoi is superb over rice
- Ongchoi leaves with salty sauces

You may have seen these  
sentences before:

Spinach **sautéed with garlic over rice**  
chard stems and **leaves** are **delicious**  
collard greens and other **salty** leafy greens

# Distributional hypothesis

## “Ongchoi”

- Ongchoi is a leafy green like spinach, chard or collard greens

空心菜  
*kangkong*  
rau muống  
...



# How can do the same thing computationally?

- Count the words in the context of ongchoi
- See what other words occur in those contexts

We can represent a word's context using vectors!

# Words and vectors

- First solution: Let's use **word-word co-occurrence counts** to represent the meaning of words!
- Each word is represented by the corresponding **row vector**

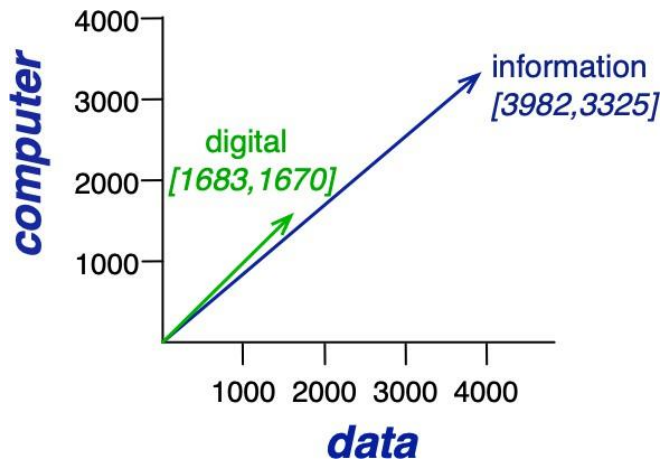
**context words:** 4 words to the left + 4 words to the right

is traditionally followed by **cherry** pie, a traditional dessert  
often mixed, such as **strawberry** rhubarb pie. Apple pie  
computer peripherals and personal **digital** assistants. These devices usually  
a computer. This includes **information** available on the internet

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Most entries are 0s  $\Rightarrow$  sparse vectors

# Measuring similarity



A common similarity metric: **cosine** of the angle between the two vectors (the larger, the more similar the two vectors are)

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|\mathbf{V}|} u_i v_i}{\sqrt{\sum_{i=1}^{|\mathbf{V}|} u_i^2} \sqrt{\sum_{i=1}^{|\mathbf{V}|} v_i^2}}$$

Q: Why cosine similarity instead of dot product  $\mathbf{u} \cdot \mathbf{v}$ ?

# Measuring similarity

What is the range of  $\cos(u, v)$  if  $u, v$  are **count vectors**?

- (a)  $[-1, 1]$
- (b)  $[0, 1]$
- (c)  $[0, +\infty)$
- (d)  $(-\infty, +\infty)$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2} \sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

# Measuring similarity

What is the range of  $\cos(u, v)$  if  $u, v$  are **count vectors**?

- (a)  $[-1, 1]$
- (b)  $[0, 1]$
- (c)  $[0, +\infty)$
- (d)  $(-\infty, +\infty)$

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{|V|} u_i v_i}{\sqrt{\sum_{i=1}^{|V|} u_i^2} \sqrt{\sum_{i=1}^{|V|} v_i^2}}$$

The answer is (b). Cosine similarity ranges between -1 and 1 in general. In this model, all the values of  $u_i, v_i$  are non-negative.



# Sparse vs. dense vectors

- The vectors in the word-word occurrence matrix are
  - **Long:** vocabulary size
  - **Sparse:** most are 0's

# Sparse vs. dense vectors

- The vectors in the word-word occurrence matrix are
  - **Long:** vocabulary size
  - **Sparse:** most are 0's
- Alternative: we want to represent words as short (50-300 dimensional) & dense (real-valued) vectors
  - The basis for modern NLP systems

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

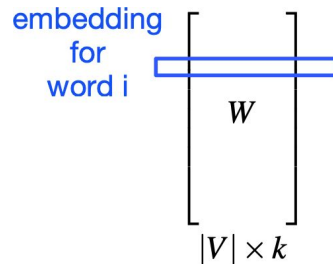
# Why dense vectors?

- Short vectors are easier to use as **features** in ML systems
- Dense vectors generalize better than explicit counts (points in real space vs points in integer space)
- Sparse vectors can't capture higher-order co-occurrence
  - $w_1$  co-occurs with “car”,  $w_2$  co-occurs with “automobile”
  - They should be similar but they aren't because “car” and “automobile” are distinct dimensions
- In practice, they work better!

# How to get short dense vectors?

- Count-based methods: **Singular value decomposition (SVD)** of count matrix

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



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

We can approximate the full matrix by only keeping the top  $k$  (e.g., 100) singular values!

# How to get short dense vectors?

- **Count-based methods:** Singular value decomposition (SVD) of count matrix
- **Prediction-based methods:**
  - Vectors are created by training a classifier to predict whether a word  $c$  (“pie”) is likely to appear in the context of a word  $w$  (“cherry”)
  - Examples: **word2vec** (Mikolov et al., 2013), Glove (Pennington et al., 2014), FastText (Bojanowski et al., 2017)

# How to get short dense vectors?

- Goal: represent words as **short** (50-300 dimensional) & **dense** (real-valued) vectors

## Count-based approaches

- Used since the 90s
- Sparse word-word co-occurrence PPMI matrix
- Decomposed with SVD

## Prediction-based approaches

- Formulated as a machine learning problem
- Word2vec (Mikolov et al., 2013)
- GloVe (Pennington et al., 2014)

Underlying theory: Distributional Hypothesis (*Firth*, '57)  
**“Similar words occur in similar contexts”**

Word embeddings: word2vec

# Word embeddings: the learning problem

- Word embeddings are learned representations from text for representing words

- Input: a large text corpora,  $V, d$

- $V$ : a pre-defined vocabulary
- $d$ : dimension of word vectors (e.g. 300)
- Text corpora:
  - Wikipedia + Gigaword 5: 6B tokens
  - Twitter: 27B tokens
  - Common Crawl: 840B tokens

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

- Output:  $f : V \rightarrow \mathbb{R}^d$

Each word is represented by a low-dimensional (e.g.,  $d = 300$ ), real-valued vector

Each coordinate/dimension of the vector doesn't have a particular interpretation



# Trained word embeddings available

- word2vec: <https://code.google.com/archive/p/word2vec/>
- GloVe: <https://nlp.stanford.edu/projects/glove/>
- FastText: <https://fasttext.cc/>

## Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the [Public Domain Dedication and License](http://www.opendatacommons.org/licenses/pddl/1.0/) v1.0 whose full text can be found at: <http://www.opendatacommons.org/licenses/pddl/1.0/>.
  - [Wikipedia 2014 + Gigaword 5](#) (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): [glove.6B.zip](#)
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): [glove.42B.300d.zip](#)
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): [glove.840B.300d.zip](#)
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): [glove.twitter.27B.zip](#)
- Ruby [script](#) for preprocessing Twitter data

Differ in algorithms, text corpora, dimensions, cased/uncased...  
Applied to many other languages

# Word embeddings

- Basic property: similar words have similar vectors

word  $w^* = \text{"sweden"}$

$$\arg \max_{w \in V} \cos(e(w), e(w^*))$$

Word	Cosine distance
-----	-----
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

$\cos(u, v)$  ranges between -1 and 1

# Word embeddings

- Basic property: similar words have similar vectors

Nearest words to  
frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



litoria



leptodactylidae



rana

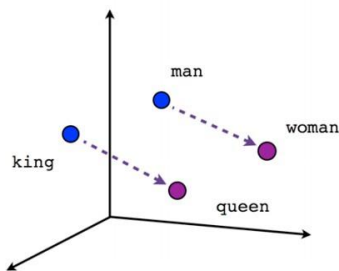


eleutherodactylus

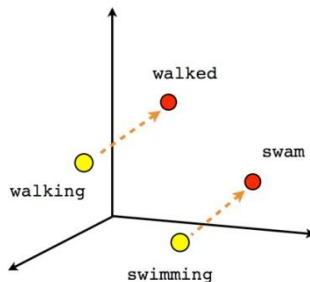
(Pennington et al, 2014): GloVe: Global Vectors for Word Representation

# Word embeddings

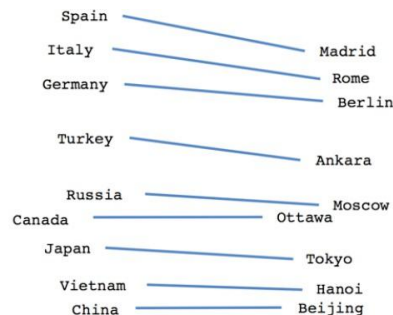
- Basic property: similar words have similar vectors
- They have some other nice properties too!



Male-Female



Verb tense



Country-Capital

$$v_{\text{man}} - v_{\text{woman}} \approx v_{\text{king}} - v_{\text{queen}}$$

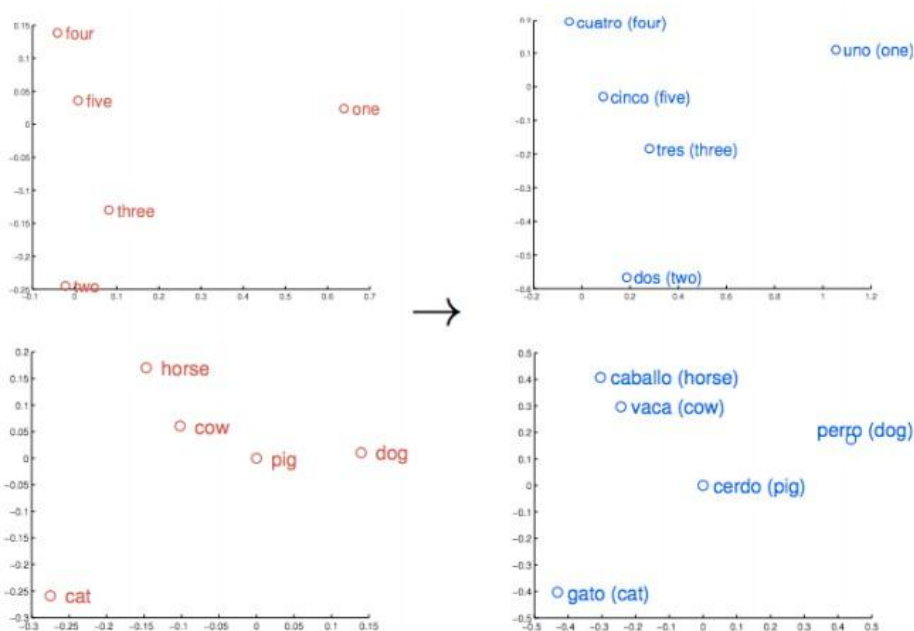
$$v_{\text{Paris}} - v_{\text{France}} \approx v_{\text{Rome}} - v_{\text{Italy}}$$

Word analogy test:  $a : a^* :: b : b^*$

# Word embeddings

- They have some other nice properties too!

$$v(\text{cuatro}) \approx Wv(\text{four})$$



(Mikolov et al, 2013): Exploiting Similarities among Languages for Machine Translation

# Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Ask “Paris : France :: Tokyo : x”

- x = Japan

Ask “father : doctor :: mother : x”

- x = nurse

Ask “man : computer programmer :: woman : x”

- x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

# Word embeddings: the learning problem

- Word embeddings are learned representations from text for representing words

- Input: a large text corpora,  $V, d$ 
  - $V$ : a pre-defined vocabulary
  - $d$ : dimension of word vectors (e.g. 300)
  - Text corpora:
    - Wikipedia + Gigaword 5: 6B tokens
    - Twitter: 27B tokens
    - Common Crawl: 840B tokens
- Output:  $f : V \rightarrow \mathbb{R}^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Each word is represented by a low-dimensional (e.g.,  $d = 300$ ), real-valued vector

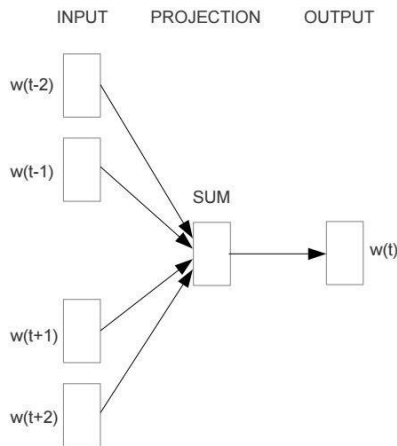
Each coordinate/dimension of the vector doesn't have a particular interpretation

# word2vec

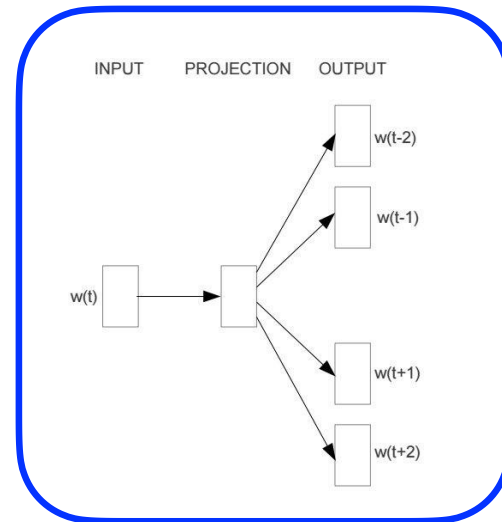
- (Mikolov et al 2013a): Efficient Estimation of Word Representations in Vector Space
- (Mikolov et al 2013b): Distributed Representations of Words and Phrases and their Compositionality



Thomas Mikolov



Continuous Bag of Words (CBOW)

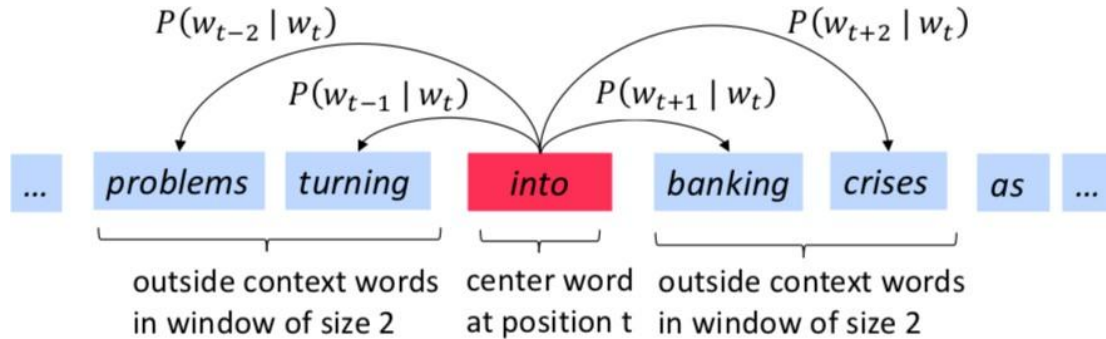


Skip-gram



# Skip-gram

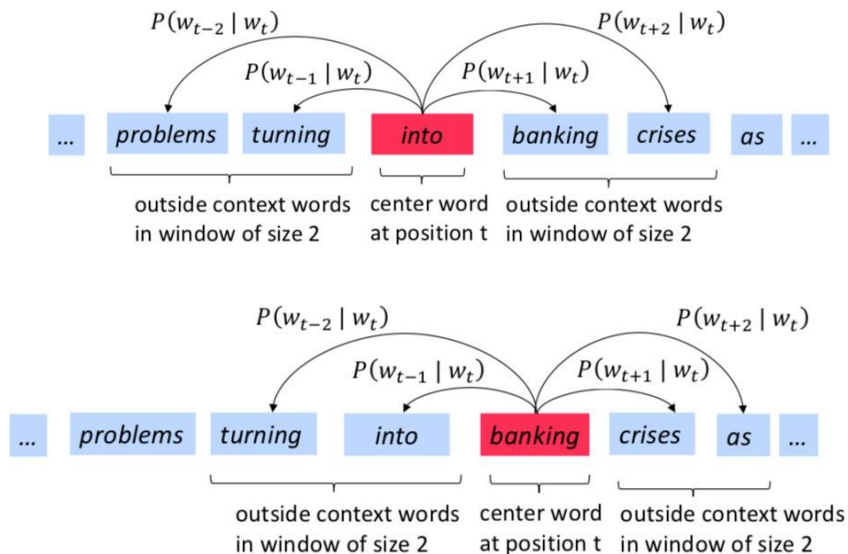
- Assume that we have a large corpus  $w_1, w_2, \dots, w_T \in V$
- Key idea:** Use each word to **predict** other words in its context ← A classification problem!
- Context: a fixed window of size  $2m$  ( $m = 2$  in the example)



$P(b | a)$  = given the center word is  $a$ , what is the probability that  $b$  is a context word?

$P(\cdot | a)$  is a probability distribution defined over  $V$ :  $\sum_{w \in V} P(w | a) = 1$

# Skip-gram



Convert the training data into:

(into, problems)

(into, turning)

(into, banking)

(into, crises)

(banking, turning)

(banking, into)

(banking, crises)

(banking, as)

...

Our goal is to find parameters that can maximize


$$\underbrace{P(\text{problems} | \text{into}) \times P(\text{turning} | \text{into}) \times P(\text{banking} | \text{into}) \times P(\text{crises} | \text{into})}_{\text{green}} \times \underbrace{P(\text{turning} | \text{banking}) \times P(\text{into} | \text{banking}) \times P(\text{crises} | \text{banking}) \times P(\text{as} | \text{banking})}_{\text{blue}} \dots$$

# Skip-gram: objective function

- For each position  $t = 1, 2, \dots, T$ , predict context words within context size  $m$ , given center word  $w_t$ :

$$\mathcal{L}(\theta) = \prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

all the parameters to be optimized



- It is equivalent as minimizing the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

# How to define $P(w_{t+j} \mid w_t; \theta)$ ?


- Use two sets of vectors for each word in the vocabulary

$\mathbf{u}_a \in \mathbb{R}^d$ : vector for center word  $a$ ,  $\forall a \in V$

$\mathbf{v}_b \in \mathbb{R}^d$ : vector for context word  $b$ ,  $\forall b \in V$

- Use inner product  $\mathbf{u}_a \cdot \mathbf{v}_b$  to measure how likely word  $a$  appears with context word  $b$

Softmax we have seen in multinomial logistic regression!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$


Recall that  $P(\cdot \mid a)$  is a probability distribution defined over  $V$ ...

# Skip-gram: objective function

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- In this formulation, we don't care about the classification task itself like we do for the logistic regression model we saw previously.
- The key point is that the parameters used to optimize this training objective—when the training corpus is large enough—can give us very good representations of words (following the principle of distributional hypothesis)!

# How many parameters in this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of  $\theta$ )?

- (a)  $d|V|$
- (b)  $2d|V|$
- (c)  $2m|V|$
- (d)  $2md|V|$

$d$  = dimension of each vector

# How many parameters in this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

How many parameters does this model have (i.e. what is size of  $\theta$ )?

- (a)  $d|V|$
- (b)  $2d|V|$
- (c)  $2m|V|$
- (d)  $2md|V|$

$d$  = dimension of each vector

The answer is (b).

Each word has two  $d$ -dimensional vectors, so it is  $2 \times |V| \times d$ .

# word2vec formulation

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Q: Why do we need two vectors for each word instead of one?

Q: Which set of vectors are used as word embeddings?



# word2vec formulation

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Q: Why do we need two vectors for each word instead of one?

A: because one word is not likely to appear in its own context window, e.g.,  $P(\text{dog} \mid \text{dog})$  should be low. If we use one set of vectors only, it essentially needs to minimize  $\mathbf{u}_{\text{dog}} \cdot \mathbf{u}_{\text{dog}}$ .

Q: Which set of vectors are used as word embeddings?

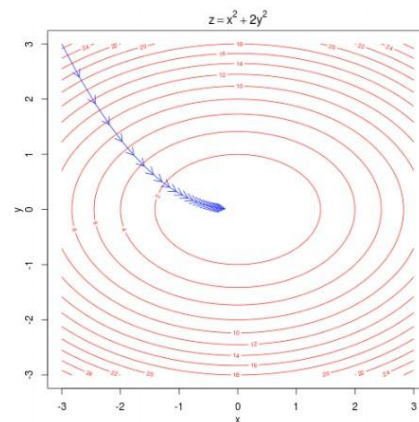
A: This is an empirical question. Typically just  $\mathbf{u}_w$  but you can also concatenate the two vectors..

# How to train this model?

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

- To train such a model, we need to compute the vector gradient  $\nabla_{\theta} J(\theta) = ?$
- Again,  $\theta$  represents all  $2d|V|$  model parameters, in one vector.

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix}$$



# Let's compute gradients for word2vec

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

Consider one pair of center/context words  $(t, c)$ :

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

We need to compute the gradient of  $y$  with respect to

$$\mathbf{u}_t \text{ and } \mathbf{v}_k, \forall k \in V$$

# Let's compute gradients for word2vec

$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

$$\begin{aligned} y &= -\log(\exp(\mathbf{u}_t \cdot \mathbf{v}_c)) + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \\ &= -\mathbf{u}_t \cdot \mathbf{v}_c + \log\left(\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)\right) \end{aligned}$$

Recall that

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$$\begin{aligned} \frac{\partial y}{\partial \mathbf{u}_t} &= \frac{\partial(-\mathbf{u}_t \cdot \mathbf{v}_c)}{\partial \mathbf{u}_t} + \frac{\partial(\log \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k))}{\partial \mathbf{u}_t} \\ &= -\mathbf{v}_c + \frac{\frac{\partial \sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\partial \mathbf{u}_t}}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \\ &= -\mathbf{v}_c + \frac{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k) \cdot \mathbf{v}_k}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \end{aligned}$$

$$= -\mathbf{v}_c + \sum_{k \in V} \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_k)}{\sum_{k' \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_{k'})} \mathbf{v}_k$$

$$= -\mathbf{v}_c + \sum_{k \in V} P(k \mid t) \mathbf{v}_k$$

# Let's compute gradients for word2vec

What about context vectors?

$$\frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k | t) - 1) \mathbf{u}_t & k = c \\ P(k | t) \mathbf{u}_t & k \neq c \end{cases} \quad y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

# Overall algorithm

- Input: text corpus, embedding size  $d$ , vocabulary  $V$ , **context size  $m$**
- Initialize  $\mathbf{u}_i, \mathbf{v}_i$  randomly  $\forall i \in V$
- Run through the training corpus and for each training instance  $(t, c)$ :

- Update  $\mathbf{u}_t \leftarrow \mathbf{u}_t - \eta \frac{\partial y}{\partial \mathbf{u}_t} \quad \frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k | t) \mathbf{v}_k$

- Update  $\mathbf{v}_k \leftarrow \mathbf{v}_k - \eta \frac{\partial y}{\partial \mathbf{v}_k}, \forall k \in V \quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k | t) - 1) \mathbf{u}_t & k = c \\ P(k | t) \mathbf{u}_t & k \neq c \end{cases}$

Q: Can you think of any issues with this algorithm?

# Skip-gram with negative sampling (SGNS)

**Problem:** every time you get one pair of  $(t, c)$ , you need to update  $\mathbf{v}_k$  with all the words in the vocabulary! This is very expensive computationally.

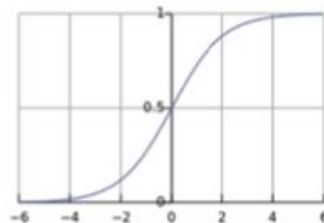
$$\frac{\partial y}{\partial \mathbf{u}_t} = -\mathbf{v}_c + \sum_{k \in V} P(k | t) \mathbf{v}_k \quad \frac{\partial y}{\partial \mathbf{v}_k} = \begin{cases} (P(k | t) - 1) \mathbf{u}_t & k = c \\ P(k | t) \mathbf{u}_t & k \neq c \end{cases}$$

**Negative sampling:** instead of considering all the words in  $V$ , let's randomly sample  $K$  (5-20) negative examples.

softmax: 
$$y = -\log \left( \frac{\exp(\mathbf{u}_t \cdot \mathbf{v}_c)}{\sum_{k \in V} \exp(\mathbf{u}_t \cdot \mathbf{v}_k)} \right)$$

Negative sampling: 
$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



# Skip-gram with negative sampling (SGNS)

**Key idea: Convert the  $|V|$ -way classification into a set of binary classification tasks.**

Every time we get a pair of words  $(t, c)$ , we don't predict  $c$  among all the words in the vocabulary. Instead, we predict  $(t, c)$  is a positive pair, and  $(t, c')$  is a negative pair for a small number of sampled  $c'$ .

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

$P(w)$ : sampling according to the frequency of words

Similar to **binary logistic regression**, but we need to optimize  $\mathbf{u}$  and  $\mathbf{v}$  together.

$$P(y = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c) \quad p(y = 0 \mid t, c') = 1 - \sigma(\mathbf{u}_t \cdot \mathbf{v}_{c'}) = \sigma(-\mathbf{u}_t \cdot \mathbf{v}_{c'})$$



# Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in  $\theta$  for every  $(t, c)$  pair?

- (a)  $Kd$
- (b)  $2Kd$
- (c)  $(K + 1)d$
- (d)  $(K + 2)d$

# Understanding SGNS

$$y = -\log(\sigma(\mathbf{u}_t \cdot \mathbf{v}_c)) - \sum_{i=1}^K \mathbb{E}_{j \sim P(w)} \log(\sigma(-\mathbf{u}_t \cdot \mathbf{v}_j))$$

In skip-gram with negative sampling (SGNS), how many parameters need to be updated in  $\theta$  for every  $(t, c)$  pair?

- (a)  $Kd$
- (b)  $2Kd$
- (c)  $(K + 1)d$
- (d)  $(K + 2)d$

The answer is (d).

We need to calculate gradients with respect to  $\mathbf{u}_t$  and  $(K + 1) \mathbf{v}_i$  (one positive and  $K$  negatives).

## L6: Word Embeddings (cont'd)