Text Embeddings

Deep Learning
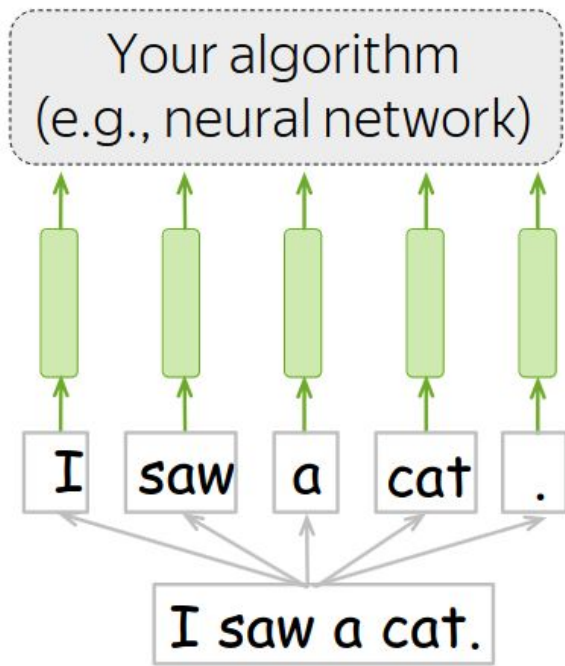
Aziz Temirkhanov
Lambda, HSE

Materials

- Stanford CS224N
- Stanford CS25
- NLP Course For You by Elena Voita

Lecture-blog and lots of additional materials are here:
https://lena-voita.github.io/nlp_course/word_embeddings.html

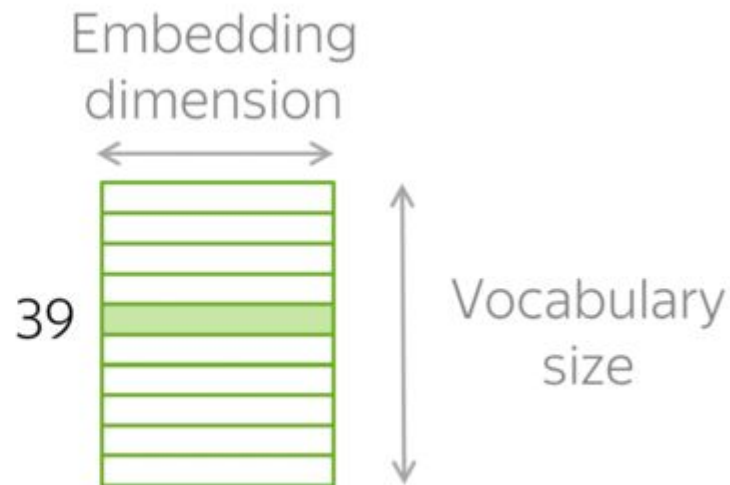NLP Course For You

Word Representations

Your algorithm (e.g., neural network) — Any algorithm for solving a task

Word representation - vector (input for your model/algorithm)

Sequence of tokens

Text (your input)

I saw a cat.

Look-up Table

Token index in
the vocabulary

39 1592 10 2548 5

I saw a cat .

Embedding
dimension

39

Vocabulary
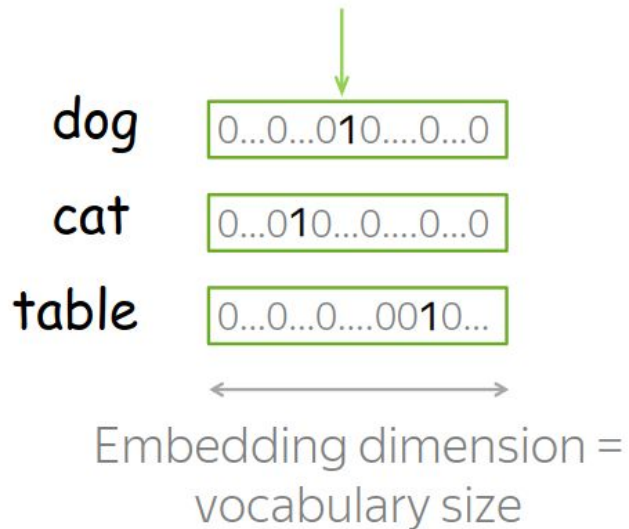size

UNK tokens

I saw a UNK .

I saw a &%! .

not in the vocabulary

Vocabulary is chosen in advance

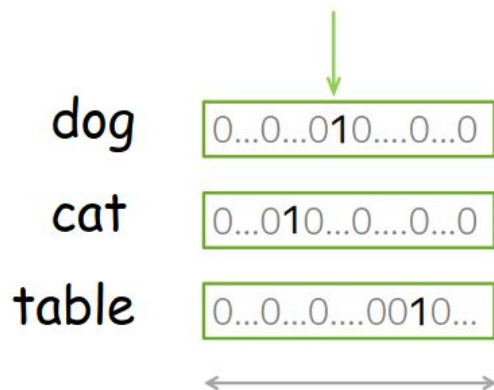Therefore, some tokens may be "unknown" – you can use a special token for them

One-Hot Vectors

One is 1, the rest are 0

dog   `0...0...010....0...0`

cat   `0...010...0....0...0`

table   `0...0...0....0010...`

Embedding dimension = vocabulary size

One-Hot Vectors

One is 1, the rest are 0

dog   0...0...0**1**0....0...0

cat   0...0**1**0...0....0...0

table   0...0...0....00**1**0...

Embedding dimension =
vocabulary size

Problems:

- Vector size is too large

- Vectors know nothing about meaning

  e.g., **cat** is as close to
  dog as it is to **table**!

Define Meaning

Do you know what the word tezgüino means ?

A bottle of tezgüino is on the table.

Everyone likes tezgüino.

Tezgüino makes you drunk.

We make tezgüino out of corn.

Can you understand what tezgüino means ?

Context

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

	(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	
wine	1	1	1	0	

<u>rows</u> are
similar

In-context semantics

(1) A bottle of _____ is on the table.

(2) Everyone likes _____ .

(3) _____ makes you drunk.

(4) We make _____ out of corn.

	(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	
wine	1	1	1	0	

This is the **distributional hypothesis**

rows are similar → meanings of the words are similar

Distributional Hypothesis

Words which frequently appear in **similar contexts** have **similar meaning**.
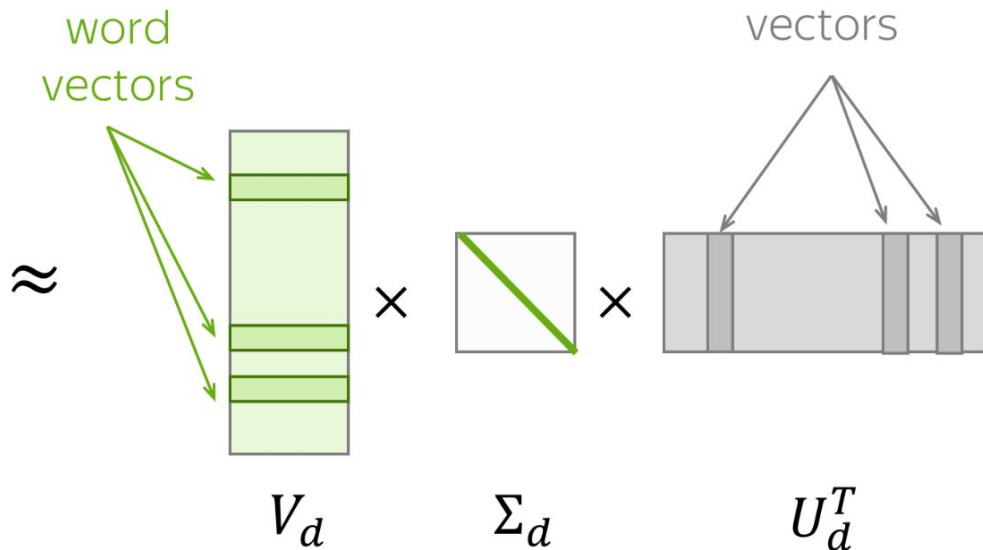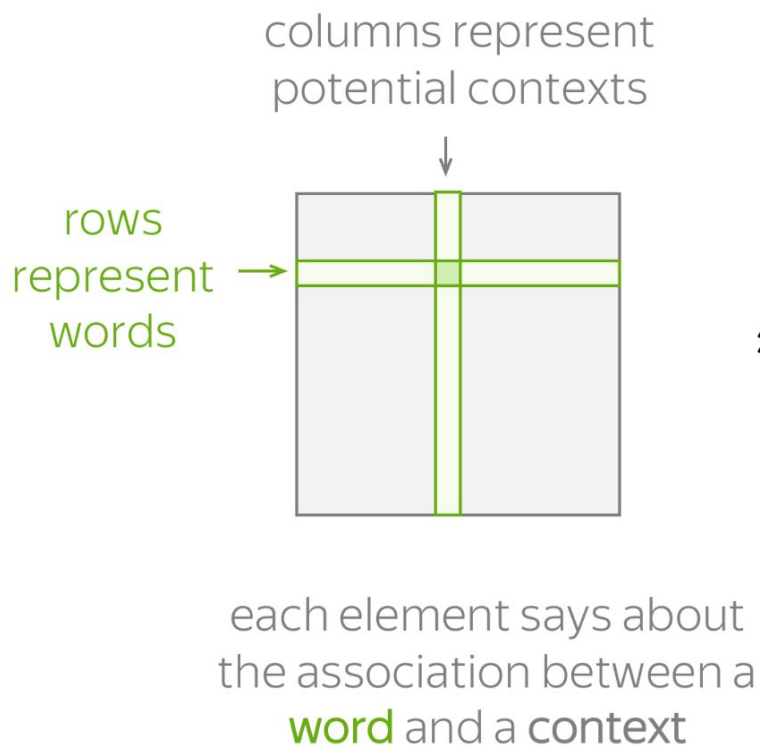
(Harris 1954, Firth 1957)

Main idea:

We have to put information about contexts into word vectors.

What comes next: different ways to do this

Count-Base Methods

columns represent
potential contexts

rows
represent
words

word
vectors

context
vectors

$\approx$

$\times$

$\times$

V_d

Σ_d

U_d^T

each element says about
the association between a
word and a **context**

Reduce dimensionality:
Truncated Singular Value Decomposition (SVD)

Count-Base Methods

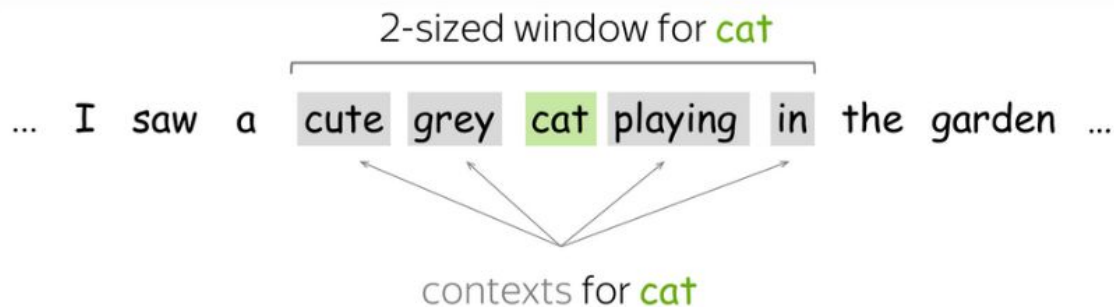To define a count-based method, we need to define two things:

- possible contexts (including what does it mean that a word appears in a context),
- the notion of association, i.e., formulas for computing matrix elements.

Below we provide a couple of popular ways of doing this.

Need to define:

- what is context
- how to compute matrix elements

Co-Occurrence Count

2-sized window for cat

... I saw a cute grey cat playing in the garden ...

contexts for cat

Context:
- surrounding words in a L-sized window

Matrix element:
- N(w, c) – number of times word w appears in context c

The simplest approach is to define contexts as each word in an L-sized window. Matrix element for a word-context pair (w, c) is the number of times w appears in context c. This is the very basic (and very, very old) method for obtaining embeddings.

Positive Pointwise Mutual Information

Here contexts are defined as before, but the measure of the association between word and context is more clever: positive PMI (or PPMI for short). PPMI measure is widely regarded as state-of-the-art for pre-neural distributional-similarity models.

Context:

- surrounding words in a L-sized window

Matrix element:

- $PPMI(w, c) = \max(0, PMI(w, c))$, where

$$PMI(w, c) = \log \frac{P(w, c)}{P(w)P(c)} = \log \frac{N(w, c)|(w, c)|}{N(w)N(c)}$$

Latent Semantic Analysis

Latent Semantic Analysis (LSA) analyzes a collection of documents. While in the previous approaches contexts served only to get word vectors and were thrown away afterward, here we are also interested in context, or, in this case, document vectors. LSA is one of the simplest topic models: cosine similarity between document vectors can be used to measure similarity between documents.

<u>Context:</u>

- document d (from a collection D)

<u>Matrix element:</u>

- tf-idf(w, d, D) = tf$(w, d) \cdot$ idf(w, D)

$$N(w, d) \qquad \log \frac{|D|}{|\{d \in D: w \in d\}|}$$

term frequency   inverse document frequency

Word2Vec

Let's remember our main idea:

> We have to put information about contexts into word vectors.

Word2Vec uses this idea differently from count-based methods:

> How: **Learn** word vectors by teaching them to **predict contexts**.
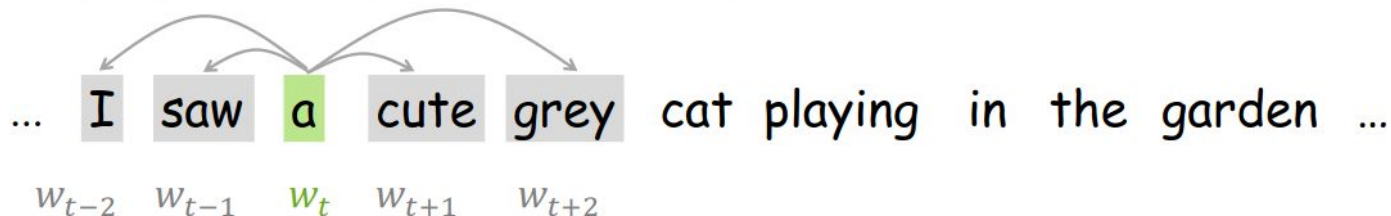
Word2Vec

How: **Learn** word vectors by teaching them to **predict contexts**.

- Learned parameters: word vectors

- Goal: make each vector "know" about the contexts of its word

- How: train vectors to predict possible contexts from words (or, alternatively, words from contexts)

Word2Vec Pipeline

- take a huge text corpus
- go over the text with a sliding window, moving one word at a time.
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

$$P(w_{t-2}|w_t) \quad P(w_{t-1}|w_t) \quad P(w_{t+1}|w_t) \quad P(w_{t+2}|w_t)$$

... I saw a cute grey cat playing in the garden ...

w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}

context words   central word   context words

Word2Vec Pipeline

- take a huge text corpus
- go over the text with a sliding window, moving one word at a time.
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

$$P(w_{t-2}|w_t) \quad P(w_{t-1}|w_t) \quad P(w_{t+1}|w_t) \quad P(w_{t+2}|w_t)$$

... I saw a cute grey cat playing in the garden ...

w_{t-2} $\quad$ w_{t-1} $\quad$ w_t $\quad$ w_{t+1} $\quad$ w_{t+2}

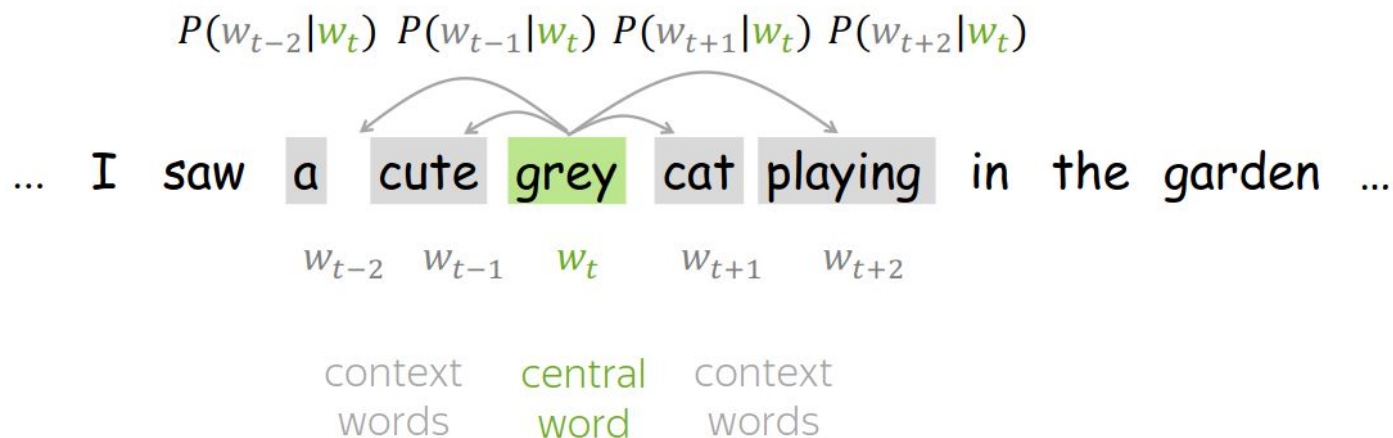context words $\qquad$ central word $\qquad$ context words

Word2Vec Pipeline

- take a huge text corpus
- go over the text with a sliding window, moving one word at a time.
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

$$P(w_{t-2}|w_t) \ P(w_{t-1}|w_t) \ P(w_{t+1}|w_t) \ P(w_{t+2}|w_t)$$

... I saw a cute grey cat playing in the garden ...

w_{t-2} $\quad$ w_{t-1} $\quad$ w_t $\quad$ w_{t+1} $\quad$ w_{t+2}

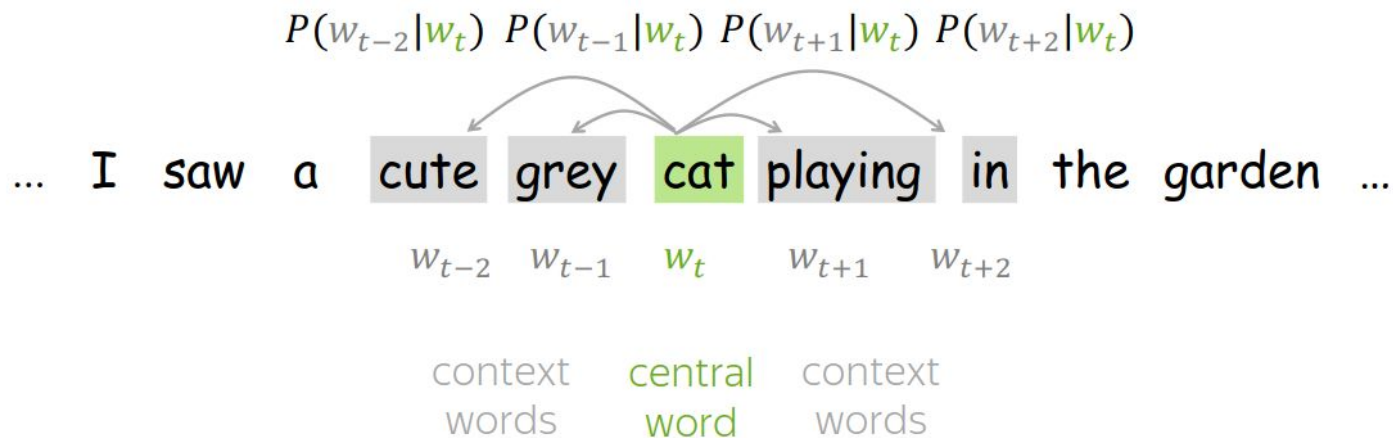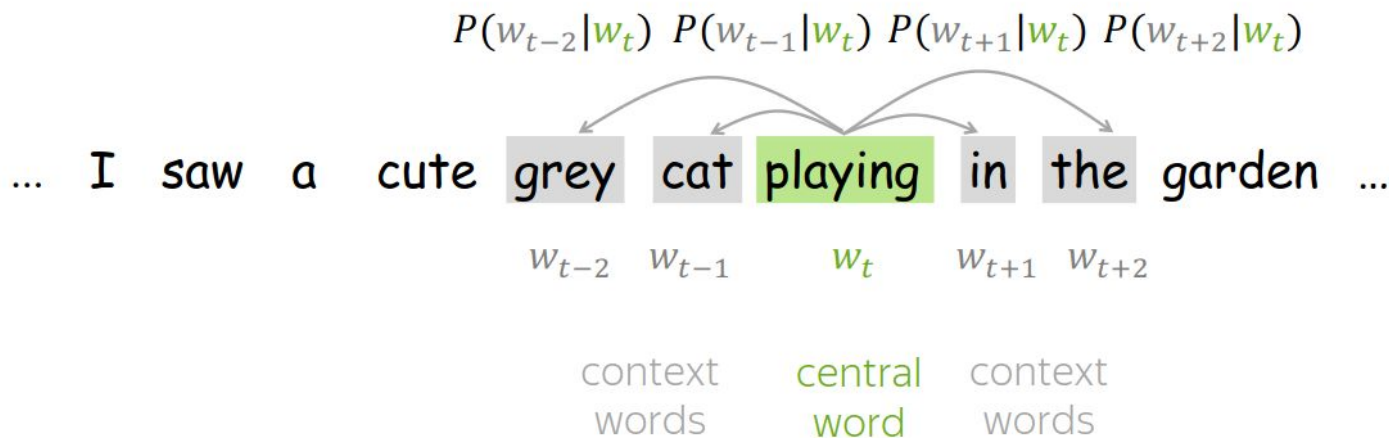context words $\qquad$ central word $\qquad$ context words

Word2Vec Pipeline

- take a huge text corpus
- go over the text with a sliding window, moving one word at a time.
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

$$P(w_{t-2}|w_t) \; P(w_{t-1}|w_t) \; P(w_{t+1}|w_t) \; P(w_{t+2}|w_t)$$

... I saw a cute grey cat playing in the garden ...

w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}

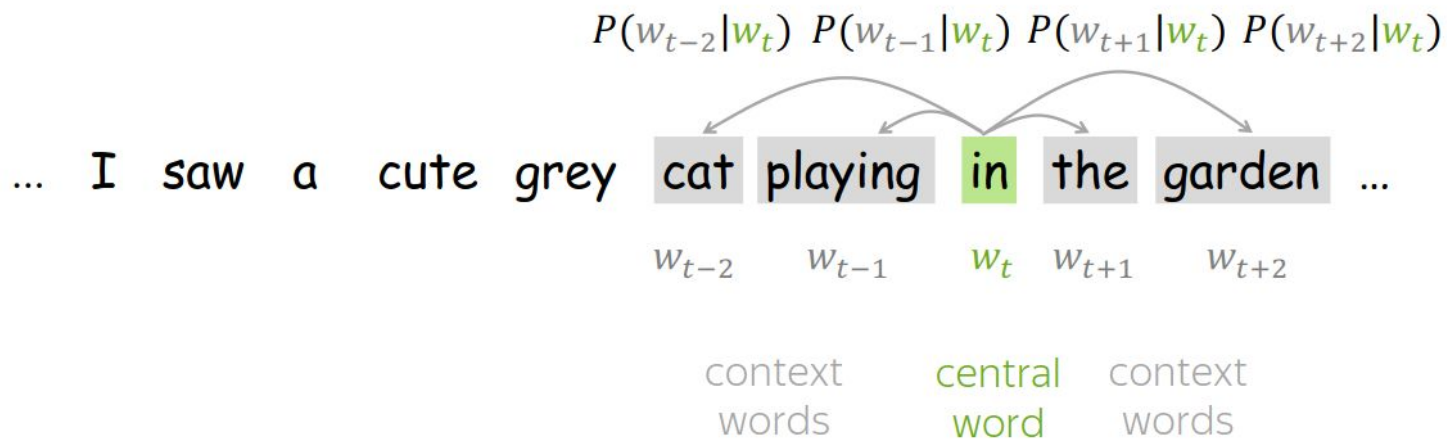context words   central word   context words

Word2Vec Pipeline

- take a huge text corpus
- go over the text with a sliding window, moving one word at a time.
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

$$P(w_{t-2}|w_t) \; P(w_{t-1}|w_t) \; P(w_{t+1}|w_t) \; P(w_{t+2}|w_t)$$

... I saw a cute grey cat playing in the garden ...

$$w_{t-2} \quad w_{t-1} \qquad w_t \qquad w_{t+1} \quad w_{t+2}$$

context words   central word   context words

Word2Vec Pipeline

- take a huge text corpus
- go over the text with a sliding window, moving one word at a time.
- for the central word, compute probabilities of context words;
- adjust the vectors to increase these probabilities.

$$P(w_{t-2}|w_t) \ P(w_{t-1}|w_t) \ P(w_{t+1}|w_t) \ P(w_{t+2}|w_t)$$

... I saw a cute grey cat playing in the garden ...

w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}

context words   central word   context words

Word2Vec loss

For each position $t = 1, \ldots, T$ in a text corpus, Word2Vec predicts context words within a m-sized window given the central word w_t:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} P(w_{t+j} | w_t, \theta),$$

where θ are all variables to be optimized. The objective function (aka loss function or cost function) $J(\theta)$ is the average negative log-likelihood:

$$\text{Loss} = J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \ne 0}} \log P(w_{t+j} | w_t, \theta)$$

agrees with our plan above ⟼ go over text | with a sliding window | compute probability of the context word given the central

Word2Vec loss

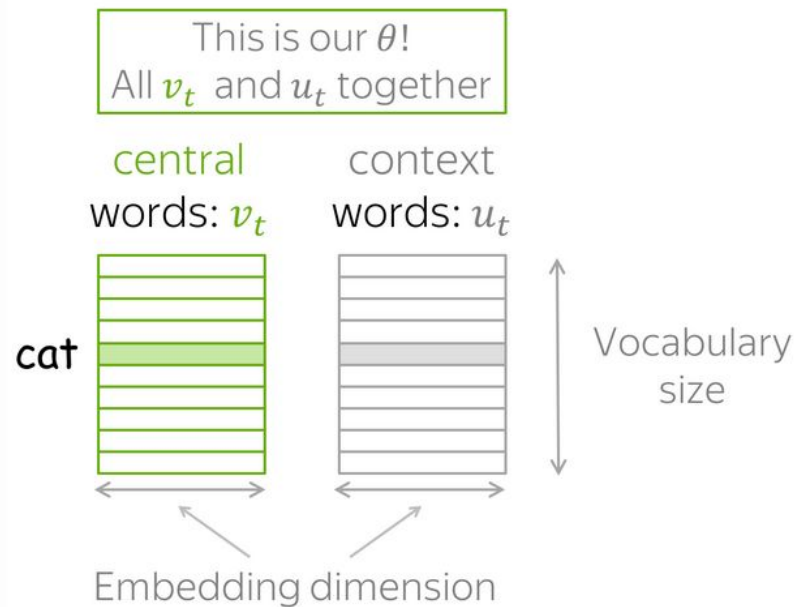How to calculate $P(w_{t+j}|w_t, \theta)$?

For each word w we will have two vectors:

- v_w when it is a central word;
- u_w when it is a context word.

(Once the vectors are trained, usually we throw away context vectors and use only word vectors.)

Then for the central word c (c - central) and the context word o (o - outside word) probability of the context word is

This is our θ!
All v_t and u_t together

central words: v_t context words: u_t

cat

Vocabulary size

Embedding dimension

Word2Vec loss

For the central word c and context word o (o - outside):

Dot product: measures similarity of o and c
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$
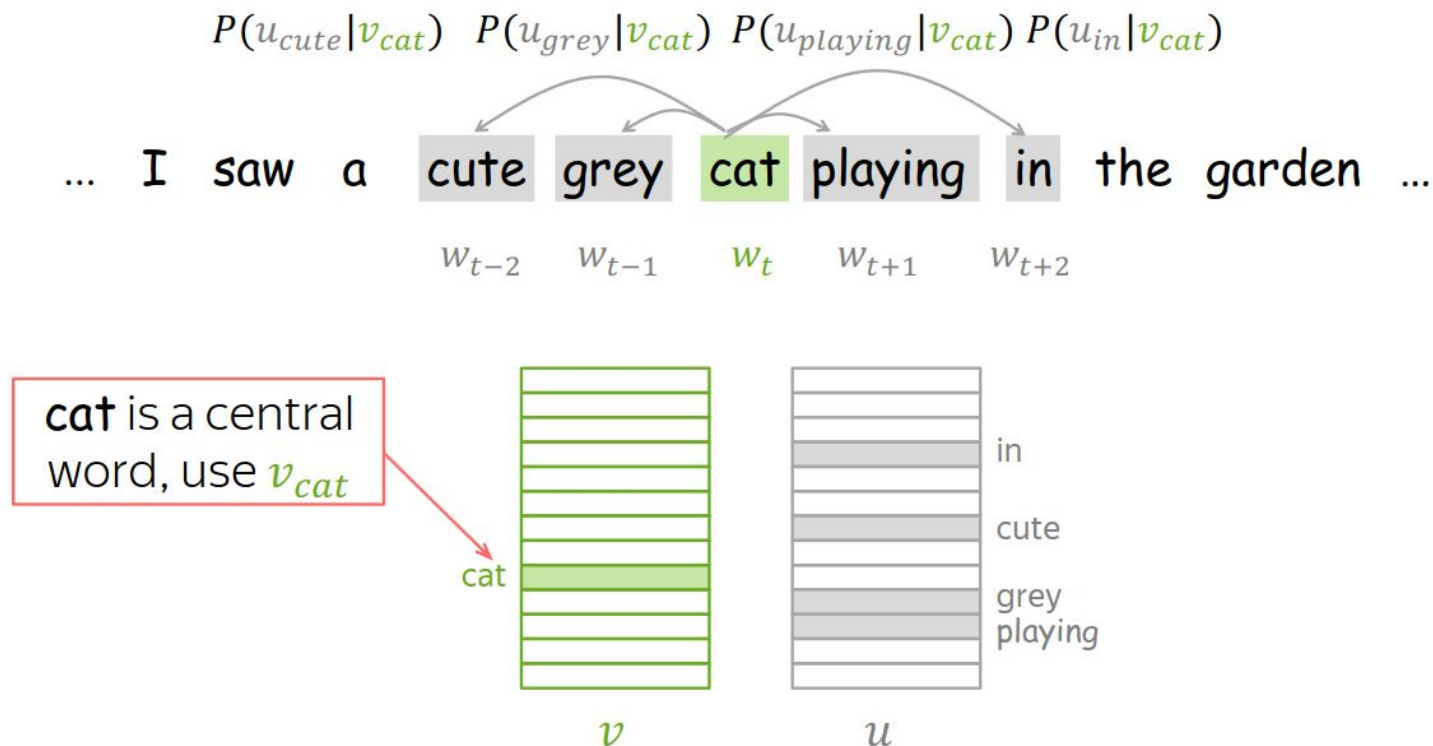
Normalize over entire vocabulary to get probability distribution

Let us recall our plan:

- ...
- adjust the vectors to increase these probabilities.
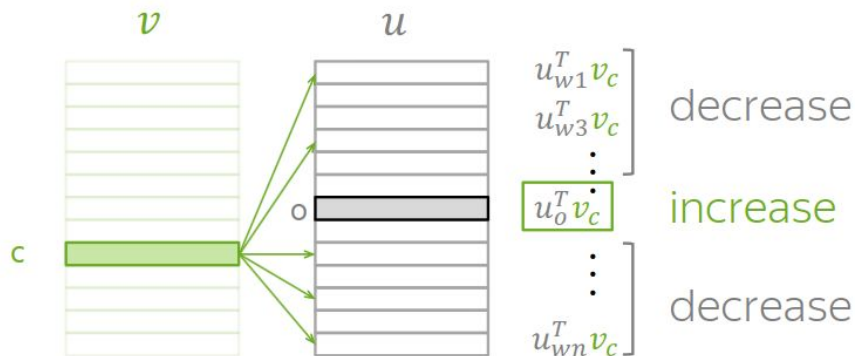
v u

$\left.\begin{array}{c} u_{w1}^T v_c \\ u_{w3}^T v_c \\ \vdots \end{array}\right]$ decrease

$u_o^T v_c$ increase

$\left.\begin{array}{c} \vdots \\ u_{wn}^T v_c \end{array}\right]$ decrease

Word2Vec training

$$P(u_{cute}|v_{cat}) \quad P(u_{grey}|v_{cat}) \quad P(u_{playing}|v_{cat}) \; P(u_{in}|v_{cat})$$

... I saw a | cute | grey | cat | playing | in | the garden ...

$w_{t-2} \qquad w_{t-1} \qquad w_t \qquad w_{t+1} \qquad w_{t+2}$

cat is a central word, use v_{cat}

cat

v

in

cute

grey
playing

u

Word2Vec training

$$P(u_{grey}|v_{playing}) \quad P(u_{cat}|v_{playing}) \quad P(u_{in}|v_{playing}) \quad P(u_{the}|v_{playing})$$

... I saw a cute grey cat playing in the garden ...

$$w_{t-2} \quad w_{t-1} \quad w_t \quad w_{t+1} \quad w_{t+2}$$
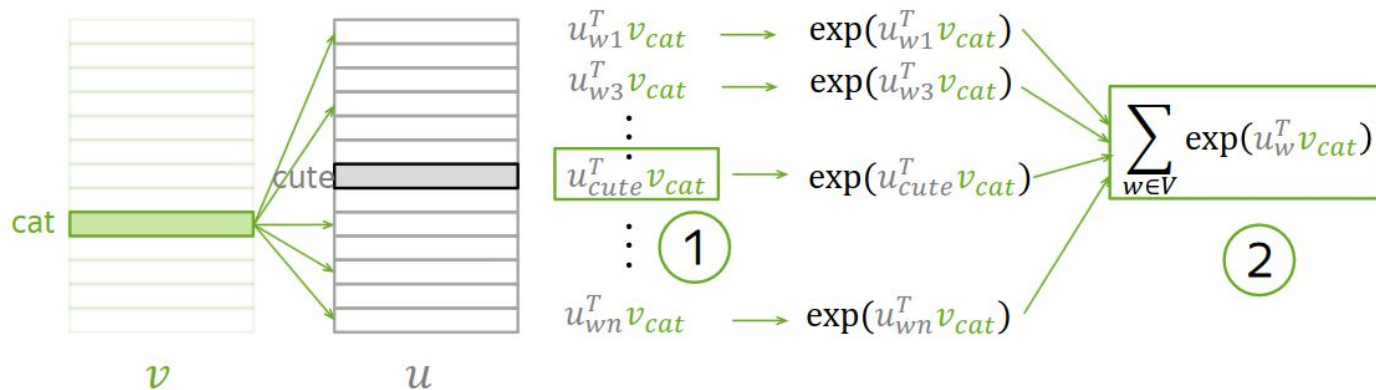
the
in

cat
grey

cat is context
again, use u_{cat}

playing

v u
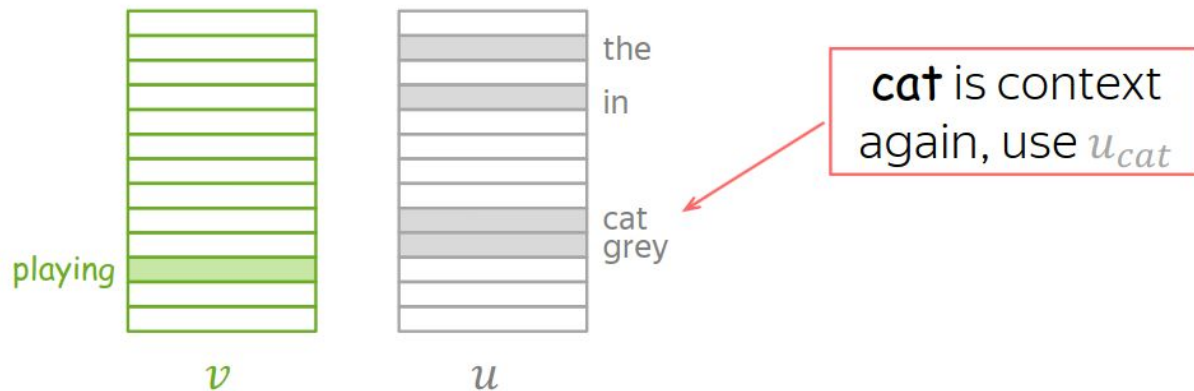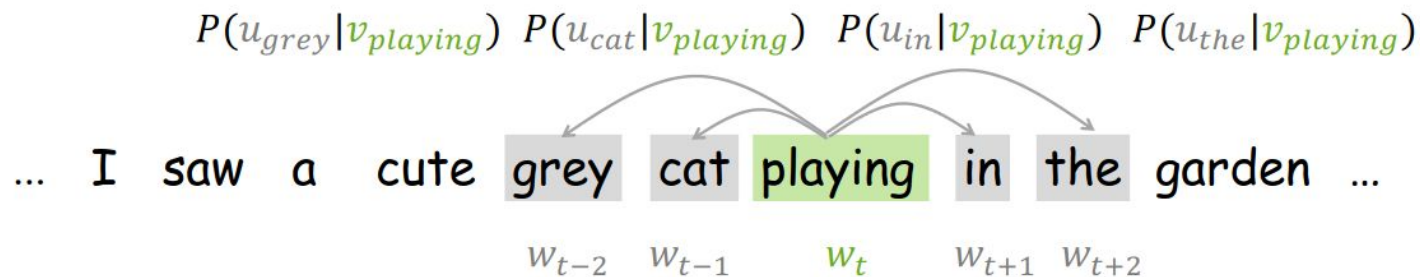
Word2Vec training

$$-\log P(cute|cat) = -\log \frac{\exp(u_{cute}^T v_{cat})}{\sum_{w \in V} \exp(u_w^T v_{cat})} = -u_{cute}^T v_{cat} + \log \sum_{w \in V} \exp(u_w^T v_{cat})$$

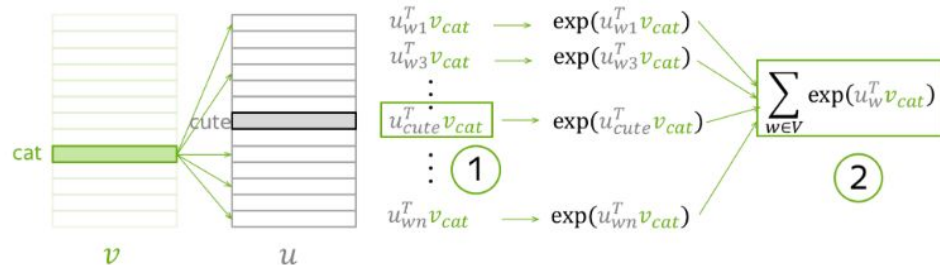**1.** Take dot product of v_{cat} with **all** u **2.** exp   **3.** sum all

Word2Vec training

$$-\log P(cute|cat)$$

$$= -u_{cute}^T v_{cat} + \log \sum_{w \in V} \exp(u_w^T v_{cat})$$

$$u_{w1}^T v_{cat} \longrightarrow \exp(u_{w1}^T v_{cat})$$
$$u_{w3}^T v_{cat} \longrightarrow \exp(u_{w3}^T v_{cat})$$
$$\vdots$$
$$u_{cute}^T v_{cat} \longrightarrow \exp(u_{cute}^T v_{cat})$$
$$\vdots \quad ①$$
$$u_{wn}^T v_{cat} \longrightarrow \exp(u_{wn}^T v_{cat})$$

$$\sum_{w \in V} \exp(u_w^T v_{cat})$$

②

cat   cute

v u

4. get loss (for this one step)

$$J_{t,j}(\theta) = \underbrace{-u_{cute}^T v_{cat}}_{①} + \underbrace{\log \sum_{w \in V} \exp(u_w^T v_{cat})}_{②}$$

5. evaluate the gradient, make an update

$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$

$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \quad \forall w \in V$$
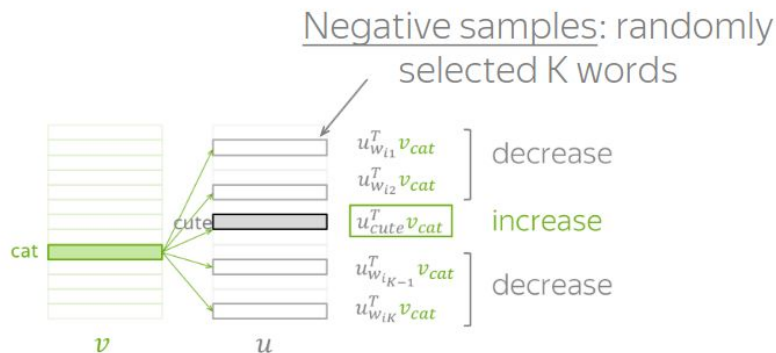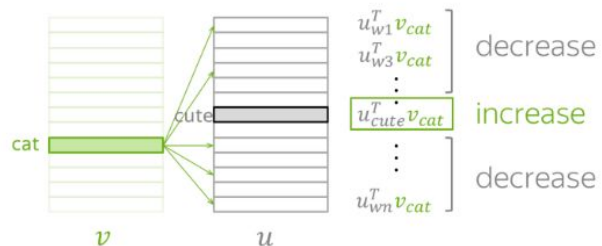
Word2Vec Negative Sampling

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **all other** u - decrease

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **a subset of other** u - decrease

Negative samples: randomly selected K words

$$u_{w_1}^T v_{cat}$$
$$u_{w_3}^T v_{cat}$$
decrease

$$\vdots$$

$$u_{cute}^T v_{cat}$$ increase

$$\vdots$$

$$u_{wn}^T v_{cat}$$
decrease

cat

v u

$$u_{w_{i1}}^T v_{cat}$$
$$u_{w_{i2}}^T v_{cat}$$
decrease

$$u_{cute}^T v_{cat}$$ increase

$$u_{w_{i_{K-1}}}^T v_{cat}$$
$$u_{w_{iK}}^T v_{cat}$$
decrease

cat

v u

Parameters to be updated: bad
- v_{cat}
- u_w for all w in the vocabulary $|V| + 1$ vectors

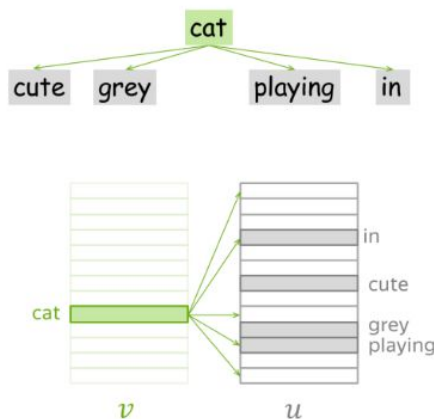Parameters to be updated: good
- v_{cat}
- u_{cute} and u_w for w in K negative examples $K + 2$ vectors
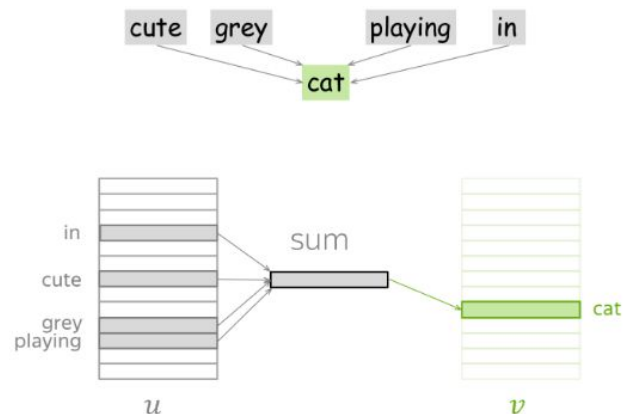
Word2Vec: Skip-Gram vs CBOW

… I saw a cute grey cat playing in the garden …

Skip-Gram: from central predict context
(one at a time)

CBOW: from sum of context predict central

(this is what we did so far)

(Continuous Bag of Words)

Word2Vec Hyperparams

- **Model**: Skip-Gram with negative sampling;

- **Number of negative examples**: for smaller datasets, 15-20; for huge datasets (which are usually used) it can be 2-5.

- **Embedding dimensionality**: frequently used value is 300, but other variants (e.g., 100 or 50) are also possible.

- **Sliding window (context) size**: 5-10.

Word2Vec: Window Size Difference
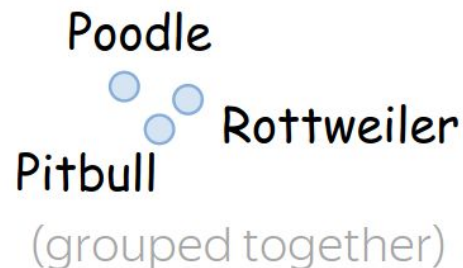
- Larger windows – more topical similarities

- Smaller windows – more functional and syntactic similarities

dog

bark leash

(grouped together)

walking

walked

run

(grouped together)

Poodle

Rottweiler

Pitbull

(grouped together)

walking

running

approaching

(grouped together)

Nearest Neighbors

Closest to **frog**:

frogs
toad
litoria
leptodactylidae
rana
lizard
eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Word Similarity Benchmark

The benchmarks that estimate the quality of embeddings:

- Contain word pairs with similarity scores from humans

- For evaluation, use correlation between the two similarity scores: from humans and from embeddings

word pair		score
vulgarism	profanity	9.62
subdividing	separate	8.67
friendships	brotherhood	7.5
exceedance	probability	5.0
assigned	allow	3.5
marginalize	interact	2.5
misleading	beat	1.25
radiators	beginning	0

Linear Structure

Many semantic and syntactic relationship between words are (almost) linear in the embedding space!

semantic: $v(\textbf{king}) - v(\textbf{man}) + v(\textbf{woman}) \approx v(\textbf{queen})$

syntactic: $v(\textbf{kings}) - v(\textbf{king}) + v(\textbf{queen}) \approx v(\textbf{queens})$