Word Embeddings

NLP

lecturer: Mollaev D. E.
Sber AI Lab

Materials

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

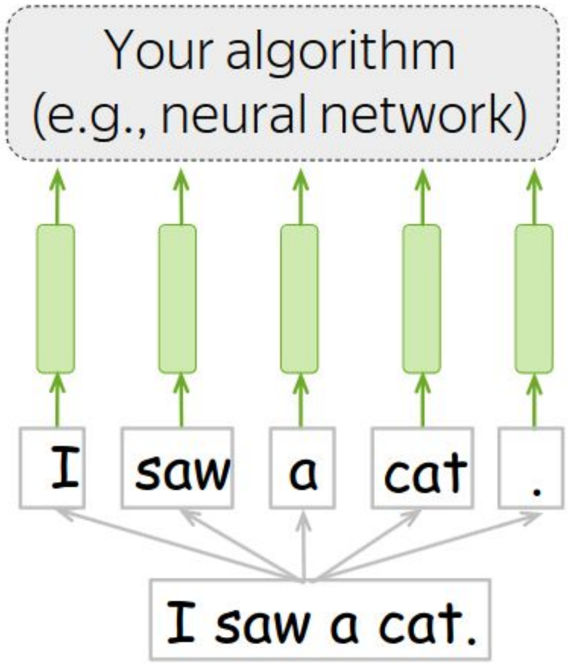
Lecture-blog and lots of additional materials are here:
https://lena-voita.github.io/nlp_course/word_embeddings.html

NLP Course For You

NLP: Word Embeddings

- $\boxed{\text{Why do we need word representations?}}$
- One-hot Vectors
- Distributional Semantics
- Count-Based Methods
- Word2Vec (Prediction-based Method)
- GloVe

Word Representations



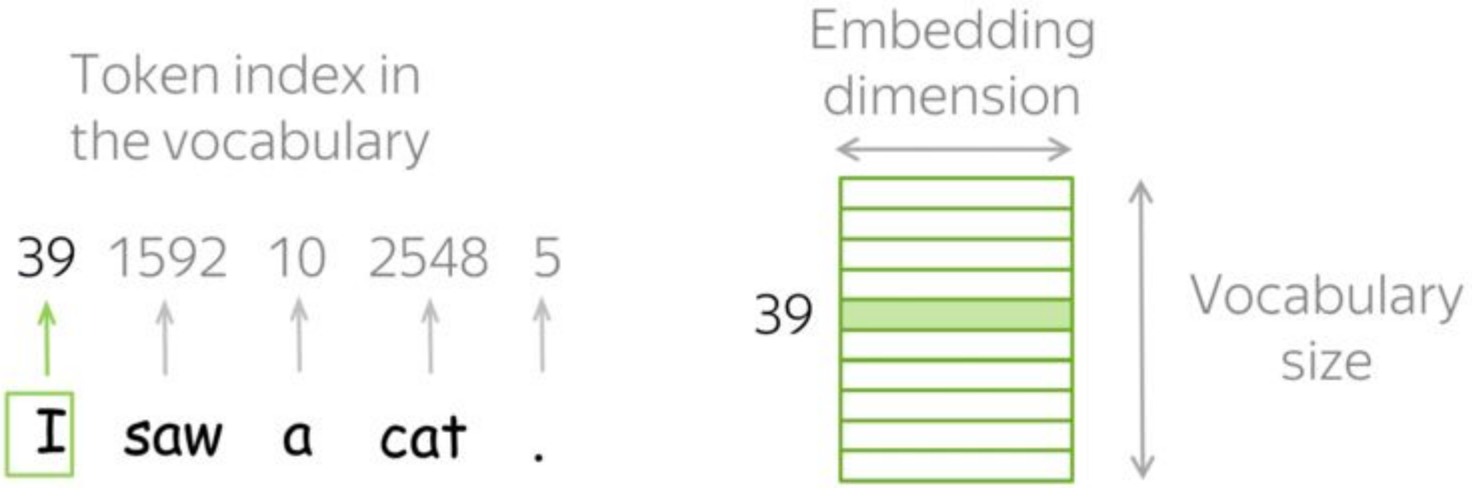
Any algorithm for solving a task

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

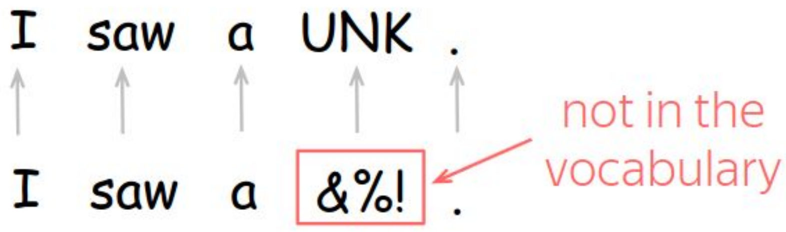
Sequence of tokens

Text (your input)

Look-up Table



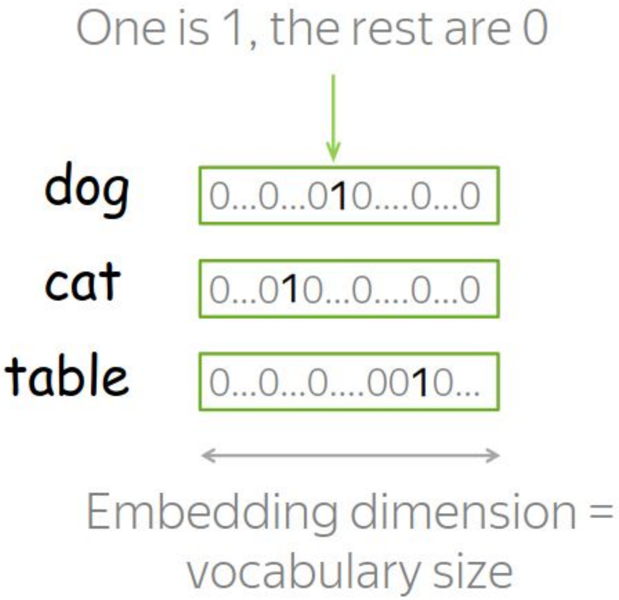
UNK tokens



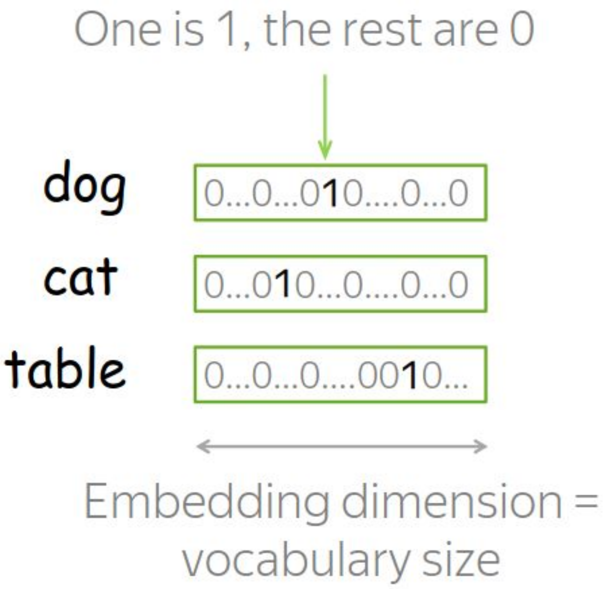
Vocabulary is chosen in advance

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

One-Hot Vectors



One-Hot Vectors



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	

rows 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

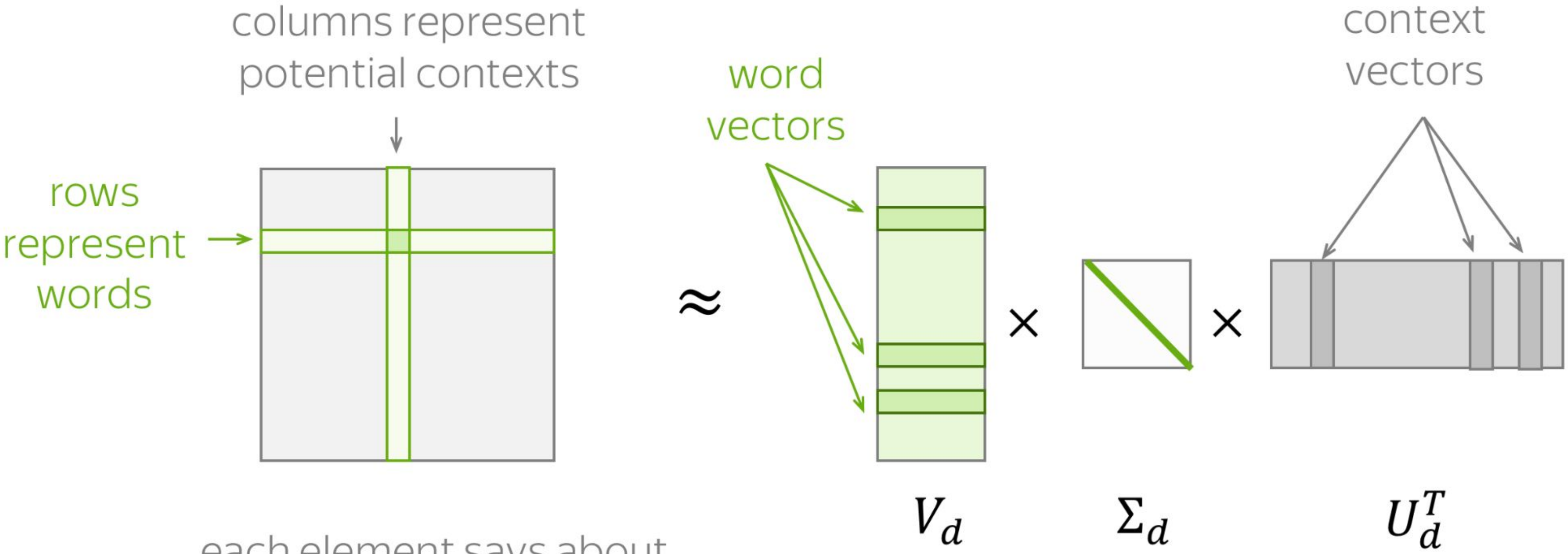
Let's remember our main idea:

> <u>Main idea</u>: We have to put information about contexts into word vectors.

Count-based methods take this idea quite literally:

> <u>How</u>: Put this information manually based on global corpus statistics.

Count-Base Methods



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:

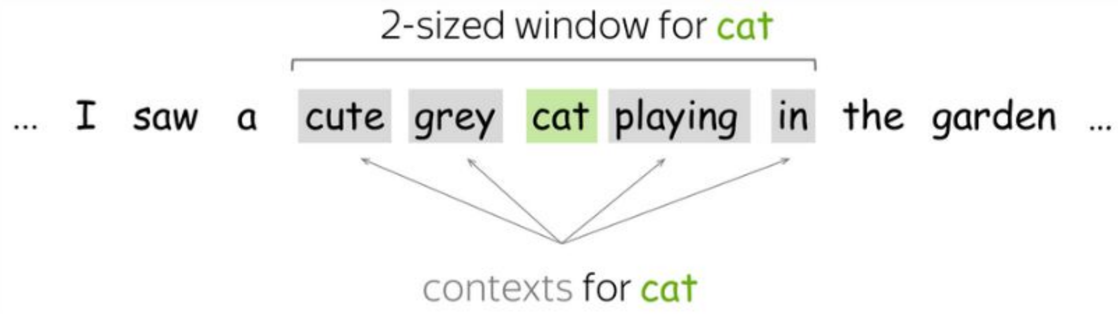
- possible contexts (including what does it mean that a word appears in a context),
- the notion of association, i.e., formulas for computing matrix elements.

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

<u>Need to define:</u>

- what is context
- how to compute matrix elements

Co-Occurrence Count



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:

- $\mathrm{PPMI}(w, c) = \max(0, \mathrm{PMI}(w, c))$,

where

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

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.

Context:

- document d (from a collection D)

Matrix element:

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

$$\text{tf}(w, d) = N(w, d) \quad \text{(term frequency)}$$

$$\text{idf}(w, D) = \log \frac{|D|}{|\{d \in D : w \in d\}|} \quad \text{(inverse document frequency)}$$

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**.

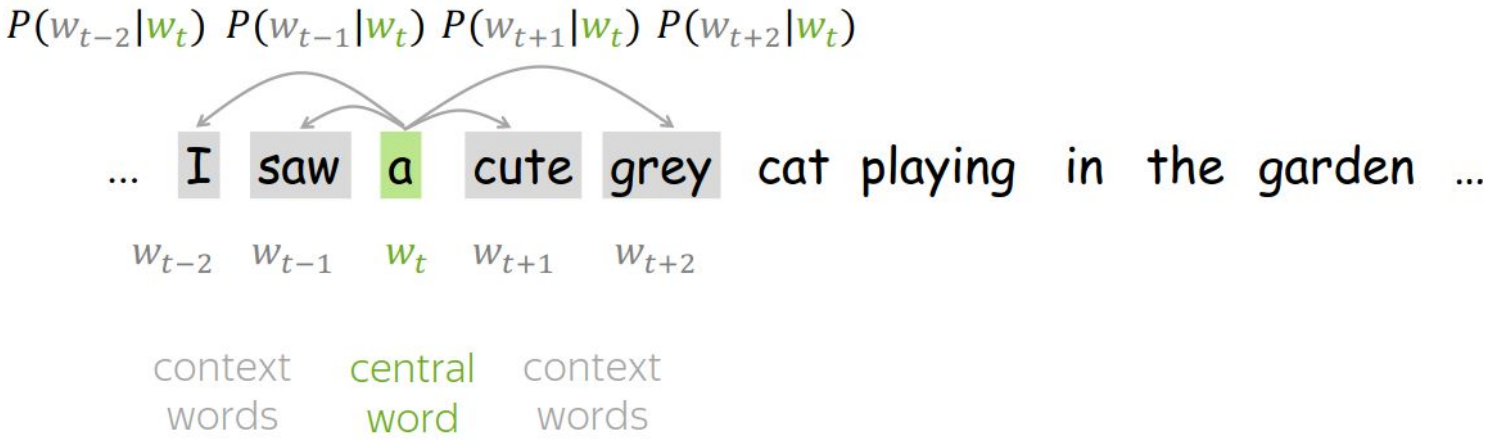
Word2Vec

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)

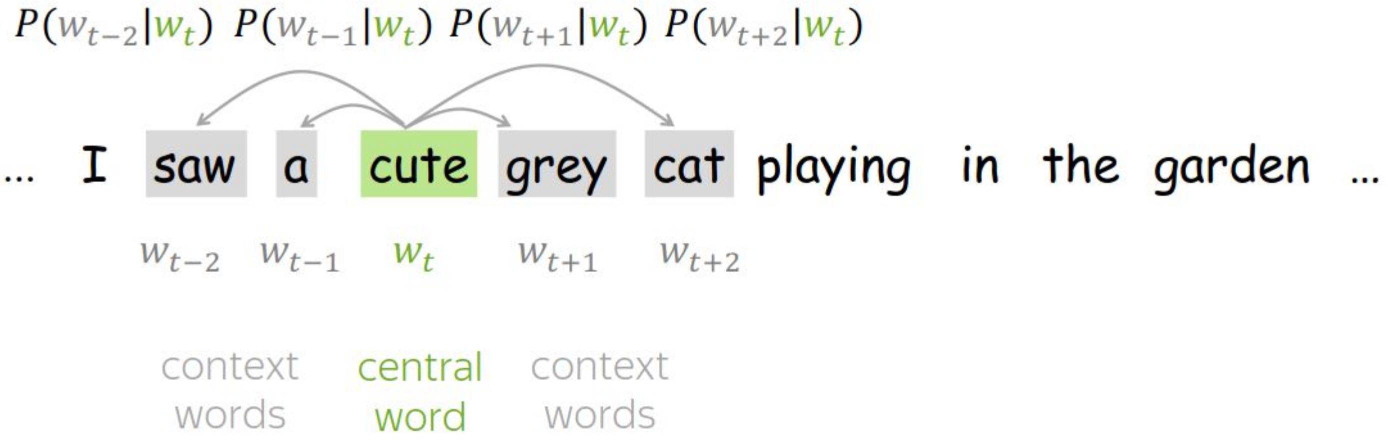
Word2Vec Pipeline

- 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.



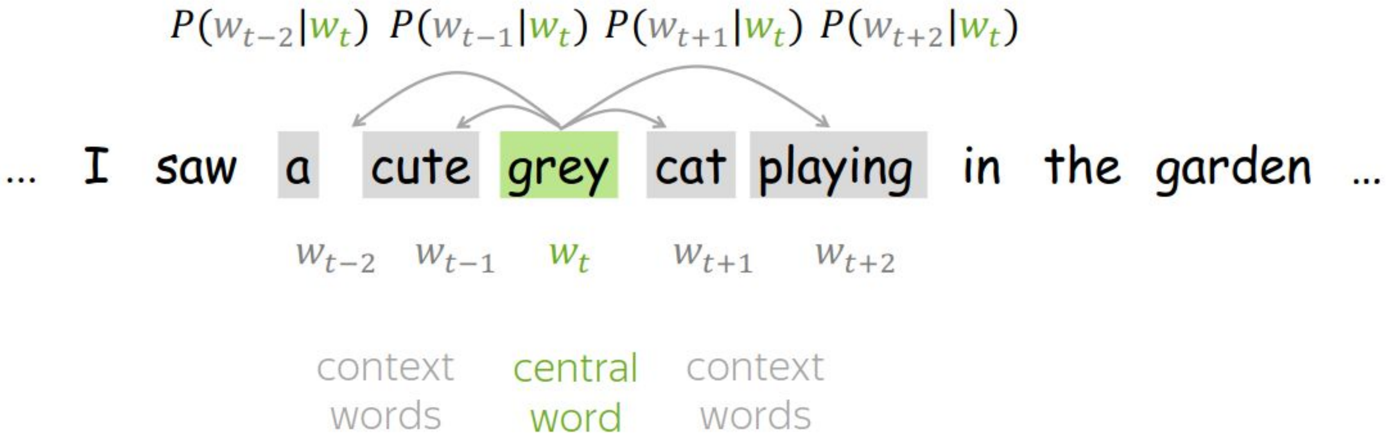
Word2Vec Pipeline

- 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.



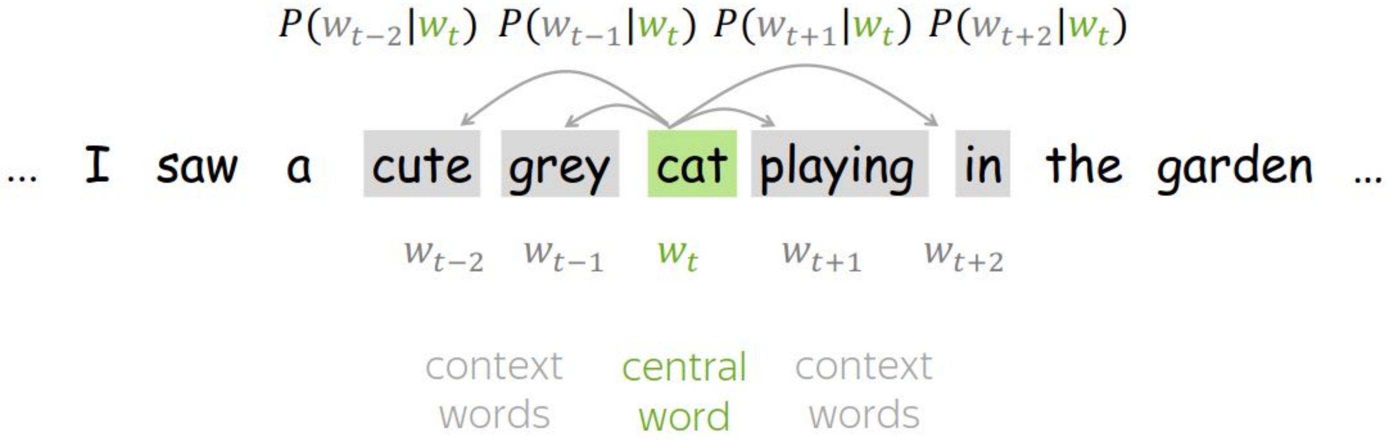
Word2Vec Pipeline

- 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.



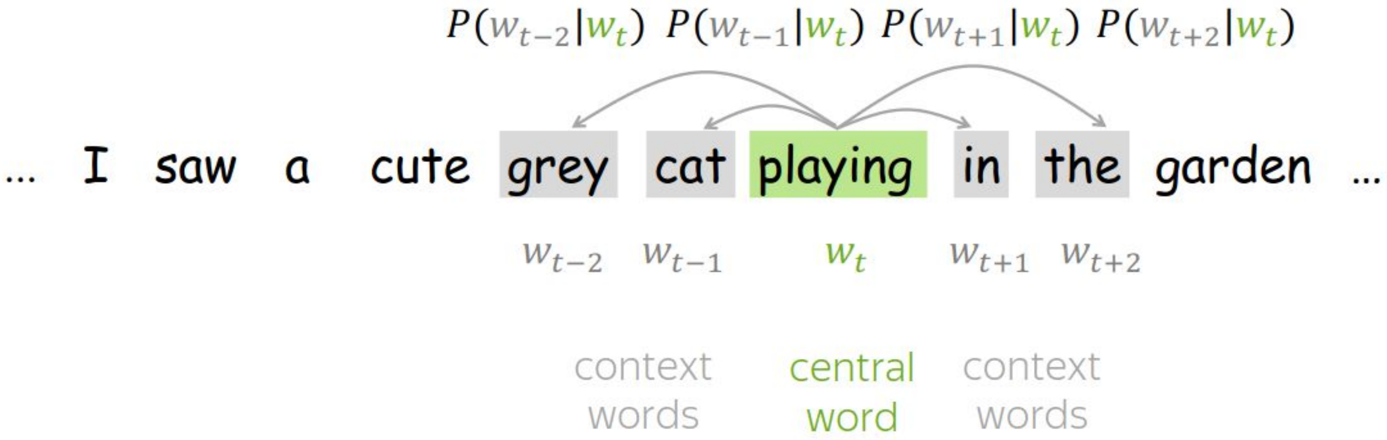
Word2Vec Pipeline

- 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.



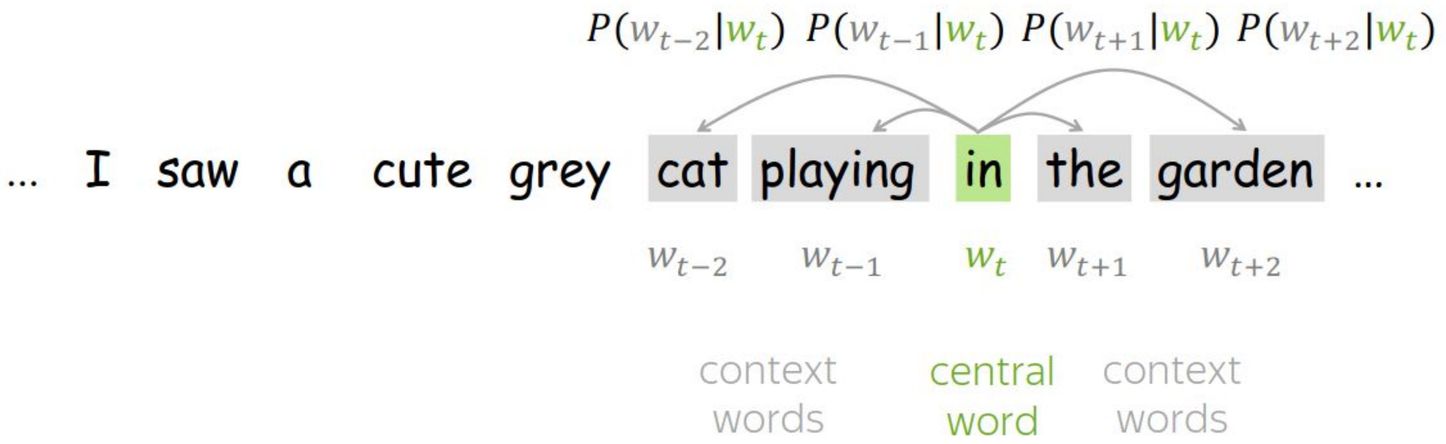
Word2Vec Pipeline

- 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.



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.



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:

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

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:

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

agrees with our plan above $\mapsto$ go over text — with a sliding window — compute probability of the context word given the central

Word2Vec loss

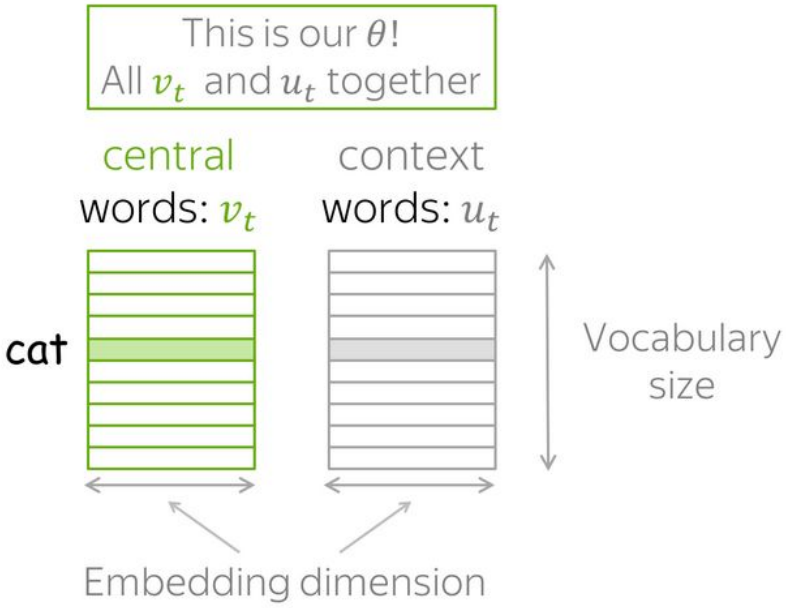
How to calculate $P(w_{t+j}|w_t, \theta)$?

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



Word2Vec loss

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

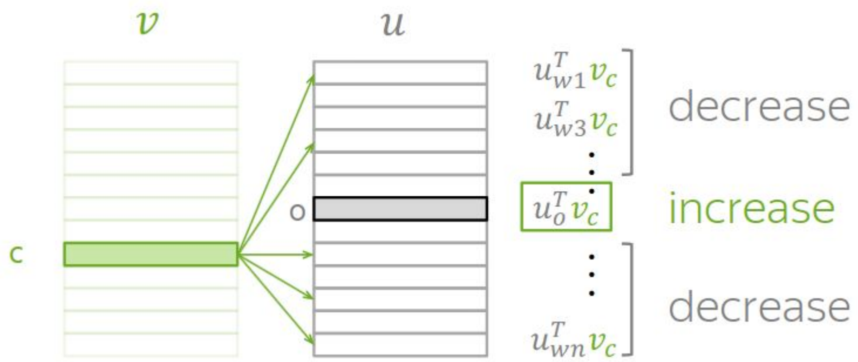
$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\sum_{w \in V} \exp(u_w^T v_c)}$$

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

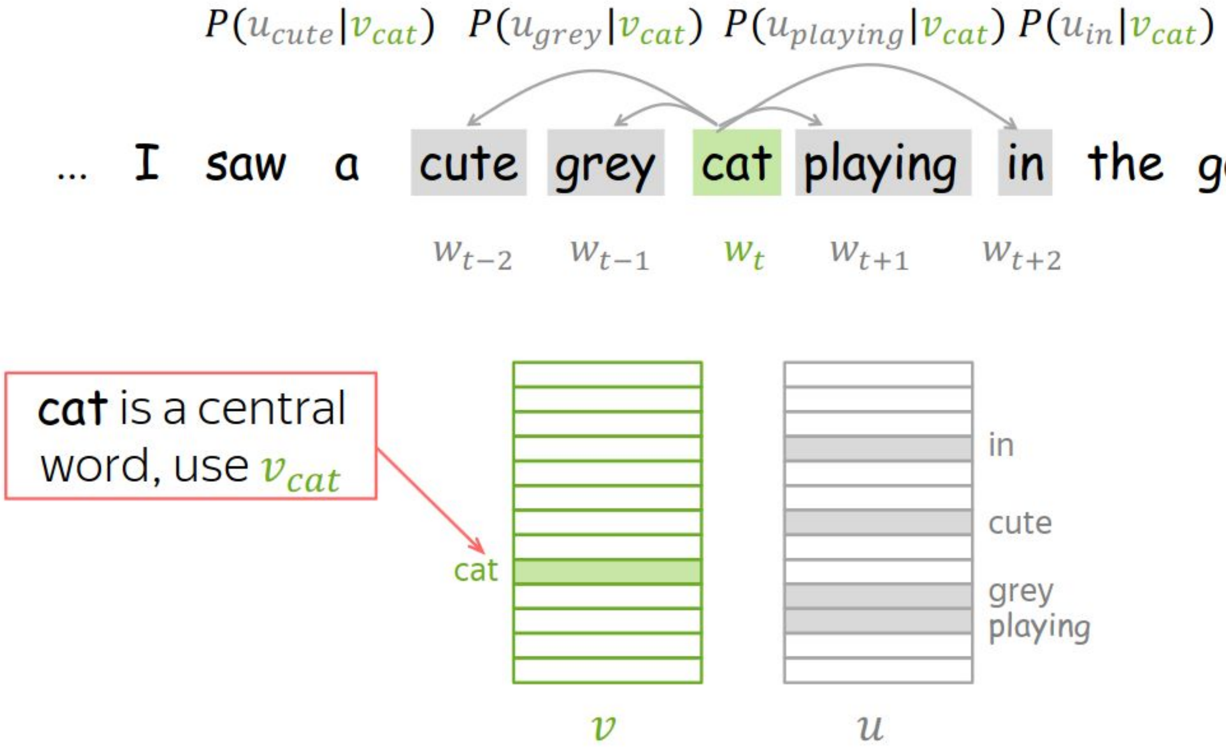
Normalize over entire vocabulary
to get probability distribution

Let us recall our plan:

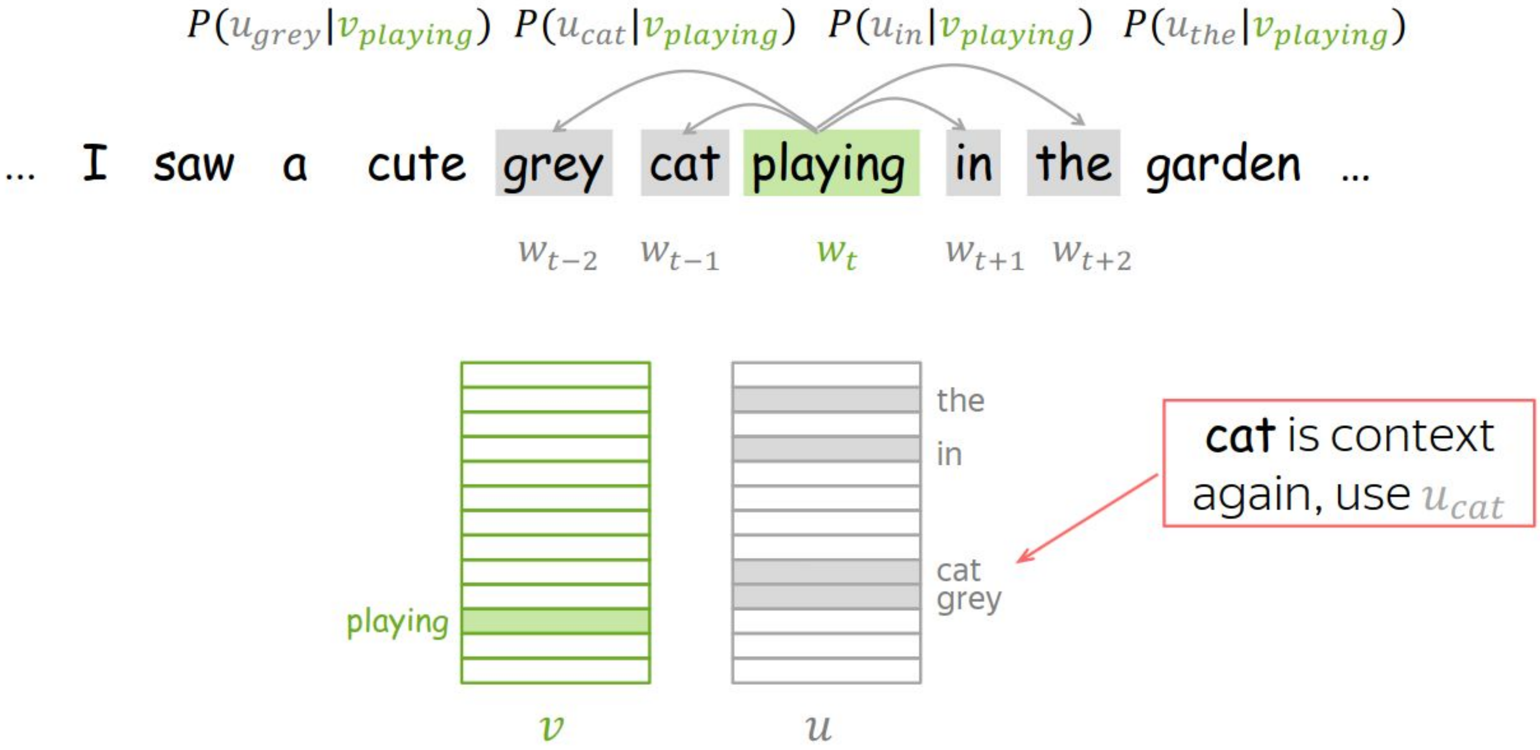
- ...
- adjust the vectors to increase these probabilities.



Word2Vec training



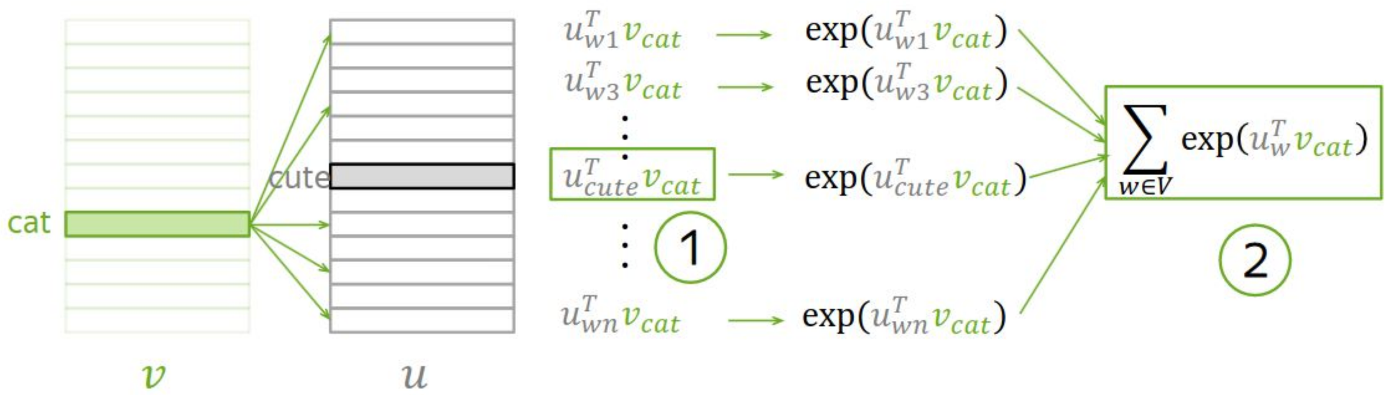
Word2Vec training



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

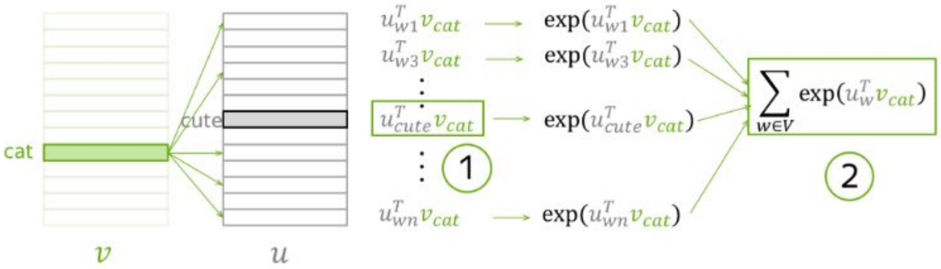


Word2Vec training

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

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

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



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

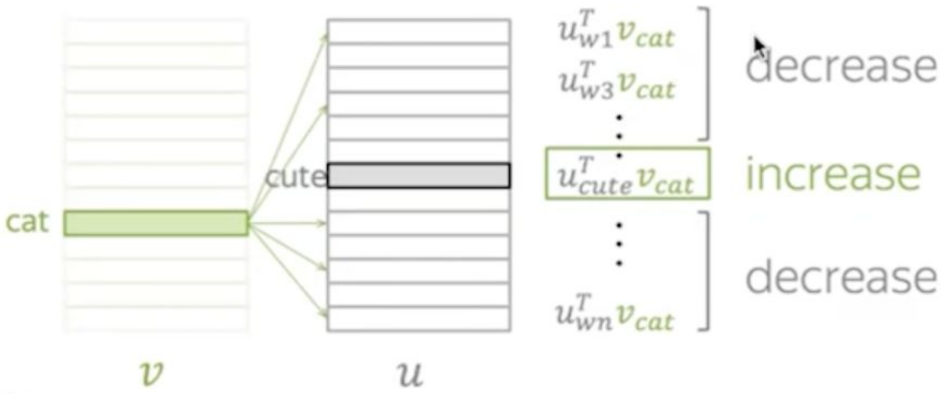
$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$

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

Recap: One Update Intuition

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

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

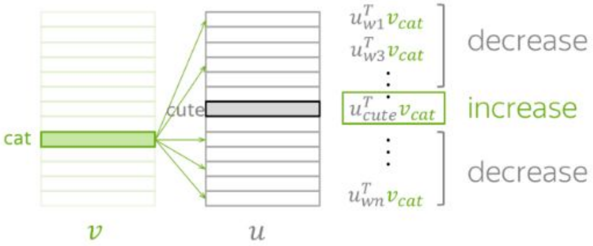


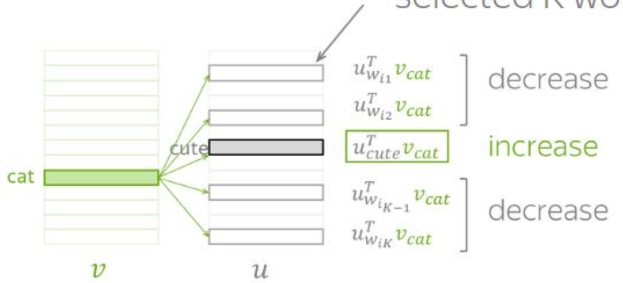
Word2Vec Negative Sampling

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

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

<u>Negative samples</u>: randomly selected K words





Parameters to be updated: bad
- v_{cat}
- u_w for all w in the vocabulary

|V| + 1 vectors

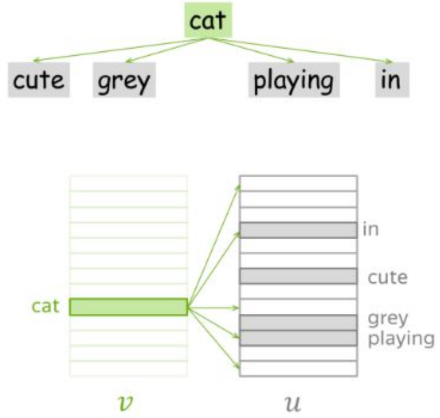
Parameters to be updated: good
- v_{cat}
- u_{cute} and u_w for w in K negative examples

K + 2 vectors

Word2Vec: Skip-Gram vs CBOW

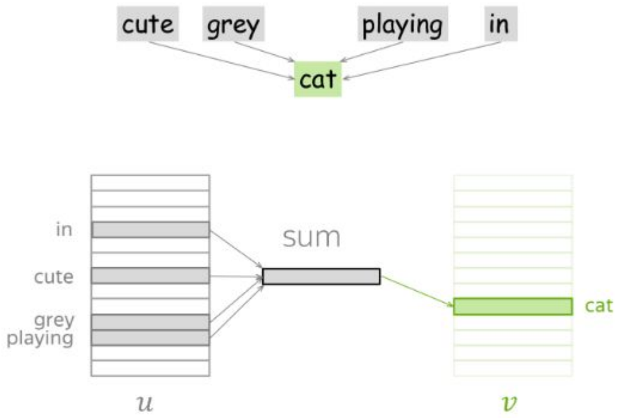
... I saw a cute grey cat playing in the garden ...

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



(this is what we did so far)

CBOW: from sum of context predict central



(Continuous Bag of Words)

Word2Vec Hyperparams

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

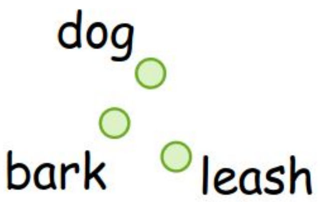
- **Number of negative examples**: for smaller datasets, 15-20; for huge datasets (which are usually used) it can be 2-5.

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

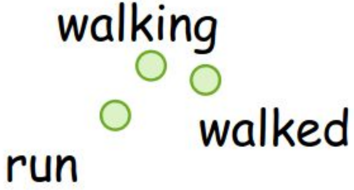
- **Sliding window (context) size**: 5-10.

Word2Vec: Window Size Difference

- Larger windows – more topical similarities

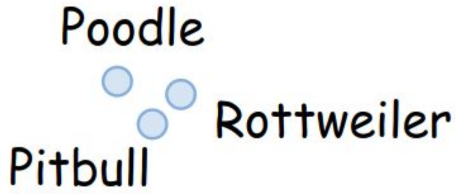


(grouped together)



(grouped together)

- Smaller windows – more functional and syntactic similarities



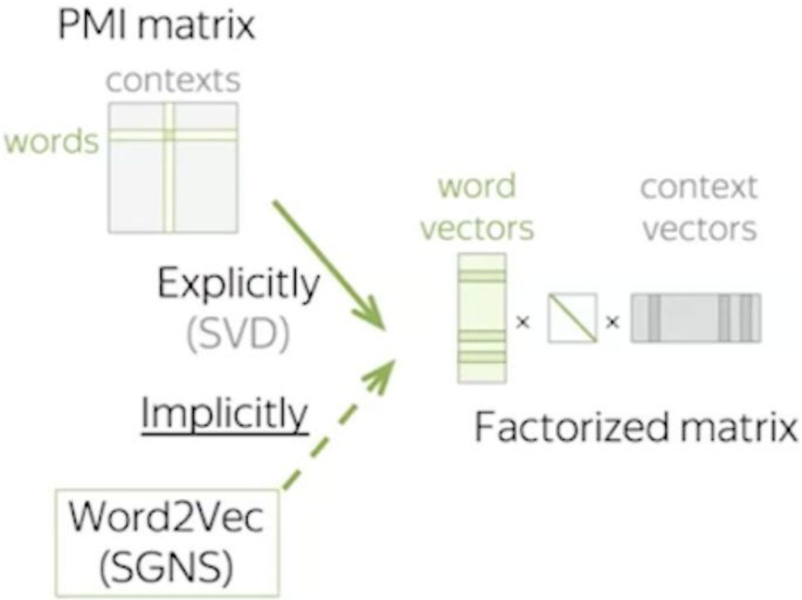
(grouped together)

(grouped together)

Relation to PMI Matrix Factorization

NLP Course For You

Word2Vec SGNS (Skip-Gram with Negative Sampling) implicitly approximates the factorization of a (shifted) PMI matrix. Learn more here.

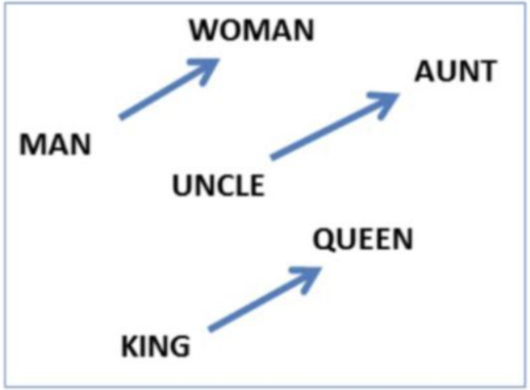


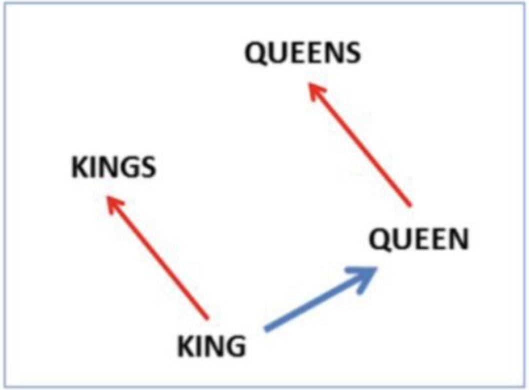
Linear Structure

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

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

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





GloVe: Global Vectors for Word Representation

	Count-based	GloVe	Prediction-based
Information comes from:	global corpus statistics	global corpus statistics	“reading” text corpora
Vectors are:	obtained via dimensionality reduction	learned by gradient descent	learned by gradient descent

context vector

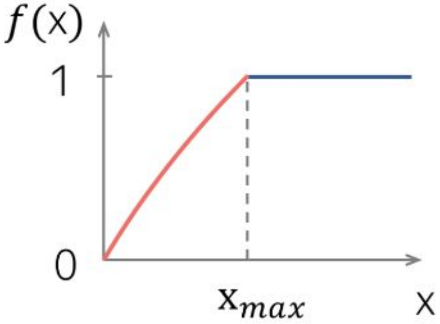
word vector

bias terms (also learned)

$$J(\theta) = \sum_{w,c \in V} f(\mathrm{N}(w, c)) \cdot (u_c^T v_w + b_c + \overline{b_w} - \log \mathrm{N}(w, c))^2$$

Weighting function to:

- penalize rare events
- not to over-weight frequent events



$$\begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max}, \\ 1 & \text{otherwise.} \end{cases}$$

$\alpha = 0.75$, $x_{max} = 100$