CMC MSU Department of Algorithmic Languages
Samsung Moscow Research Center

Neural Networks for Natural Language Processing

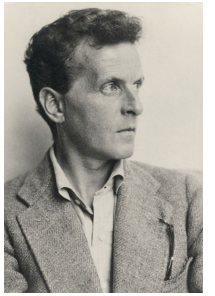
Нейронные сети в задачах автоматической обработки текстов

Block: Word Embeddings.
Lecture 1. Word Vectors: PPMI, word2vec and Glove.

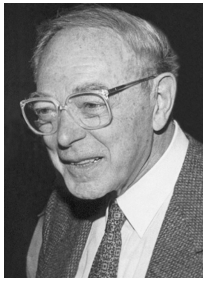
Arefyev Nikolay
CMC MSU Department of Algorithmic Languages &
Samsung Moscow Research Center

Data-driven approach to derivation of word meaning

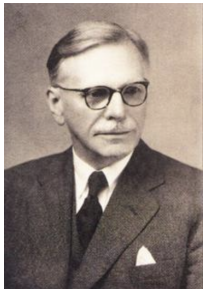
- Ludwig Wittgenstein (1945): “The meaning of a word is its use in the language”



- Zellig Harris (1954): “If A and B have almost identical environments we say that they are synonyms”



- John Firth (1957): “You shall know the word by the company it keeps.”



Source: https://web.stanford.edu/~jurafsky/slp3/

What does “ong choi” mean?

Suppose you see these sentences:

- Ong choi is delicious **sautéed with garlic**.
- Ong choi is superb **over rice**
- Ong choi **leaves** with salty sauces

- And you've also seen these:
  - …spinach **sautéed with garlic over rice**
  - Chard stems and **leaves** are **delicious**
  - Collard greens and other **salty** leafy greens

- Conclusion:
  - Ong choi is a leafy green like spinach, chard, or collard greens

Source: https://web.stanford.edu/~jurafsky/slp3/

“Water Spinach”

Source: https://web.stanford.edu/~jurafsky/slp3/

We'll build a model of meaning focusing on similarity

- Each word = a vector
  - Not just "word" or "word45".
- Similar words are "nearby in space"

Source: https://web.stanford.edu/~jurafsky/slp3/

We define a word as a vector

- Called an "embedding" because it's embedded into a space
- The standard way to represent meaning in NLP
- Fine-grained model of meaning for similarity
  - NLP tasks like sentiment analysis
    - With words, requires **same** word to be in training and test
    - With embeddings: ok if **similar** words occurred!!!
  - Question answering, conversational agents, etc

Source: https://web.stanford.edu/~jurafsky/slp3/

Two kinds of embeddings

- Sparse (e.g. TF-IDF, PPMI)
  - A common baseline model
  - Sparse vectors
  - Words are represented by a simple function of the counts of nearby words

- Dense (e.g. word2vec)
  - Dense vectors
  - Representation is created by training a classifier to distinguish nearby and far-away words

Source: https://web.stanford.edu/~jurafsky/slp3/

Representation of Documents: The Vector Space Model (VSM)

- (a.k.a. term-document matrix in Information Retrieval)
- word vectors: characterizing word with the documents they occur in
- document vectors: characterizing documents with their words

Documents

Words

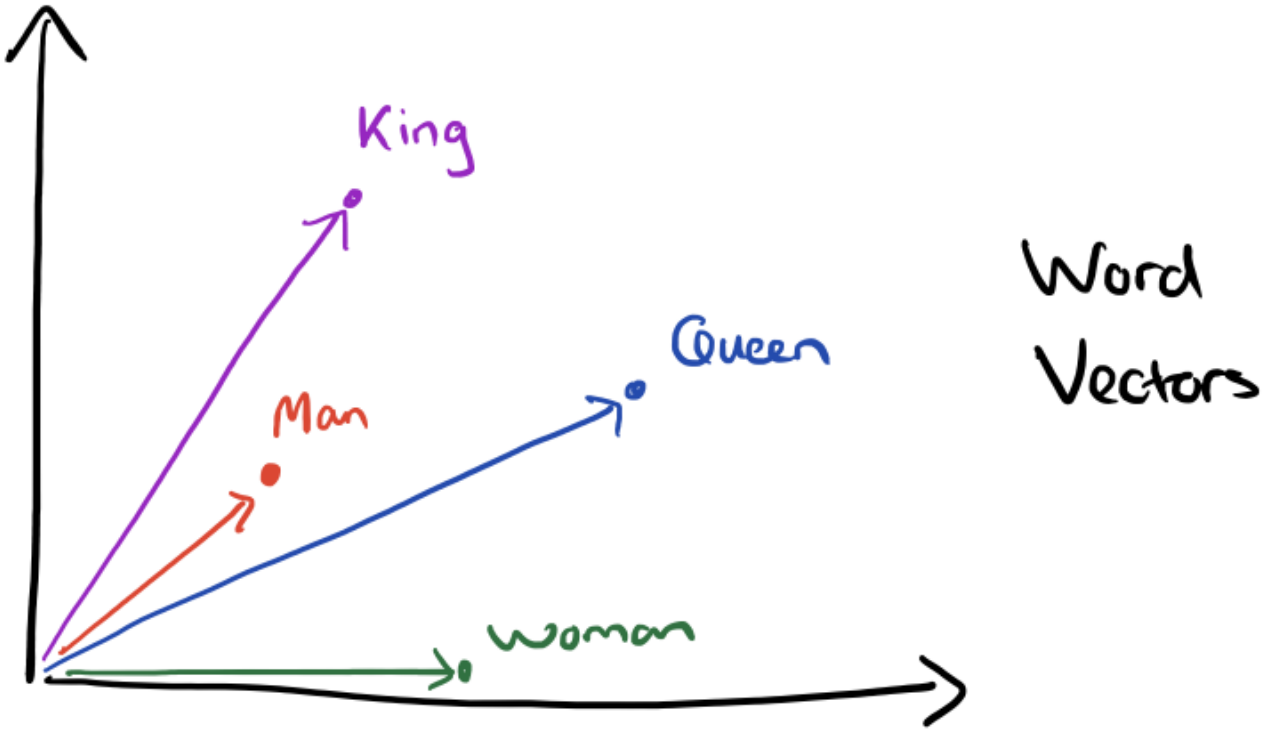
	d1	**d2**	**…**	**di**	**…**	**dn**
w1						
w2						
…						
wj				n(di,wj)		
…						
wm						

n(di, wj):= (number of words wj in document di) * term weighting

Source: https://web.stanford.edu/~jurafsky/slp3/

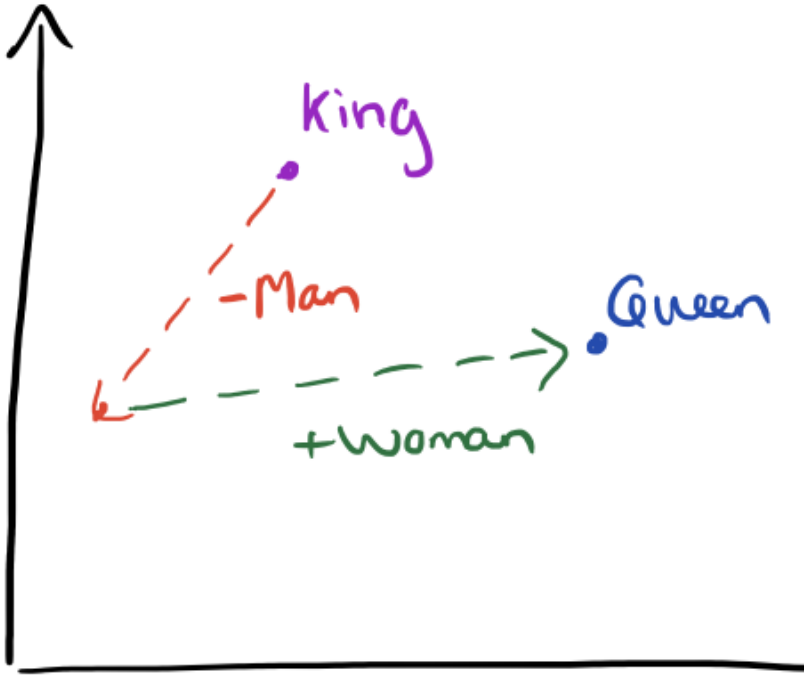
Cosine as a similarity measure

- Angle is small → cosine has a large value
- Angle is large → cosine has a small value



Source: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

The result of the vector composition King – Man + Woman = ?



Vector Composition

Relationship	Example 1	Example 2
France - Paris	Italy: Rome	Japan: Tokyo
big - bigger	small: larger	cold: colder
Miami - Florida	Baltimore: Maryland	Dallas: Texas
Einstein - scientist	Messi: midfielder	Mozart: violinist
Sarkozy - France	Berlusconi: Italy	Merkel: Germany
copper - Cu	zinc: Zn	gold: Au
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev
Microsoft - Windows	Google: Android	IBM: Linux
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy
Japan - sushi	Germany: bratwurst	France: tapas

Source: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

word2vec (Mikolov et al., 2013)

- Idea: predict rather than count

- Instead of counting how often each word w occurs near "apricot" train a classifier on a binary prediction task:
  - Is w likely to show up near "apricot"?

- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings

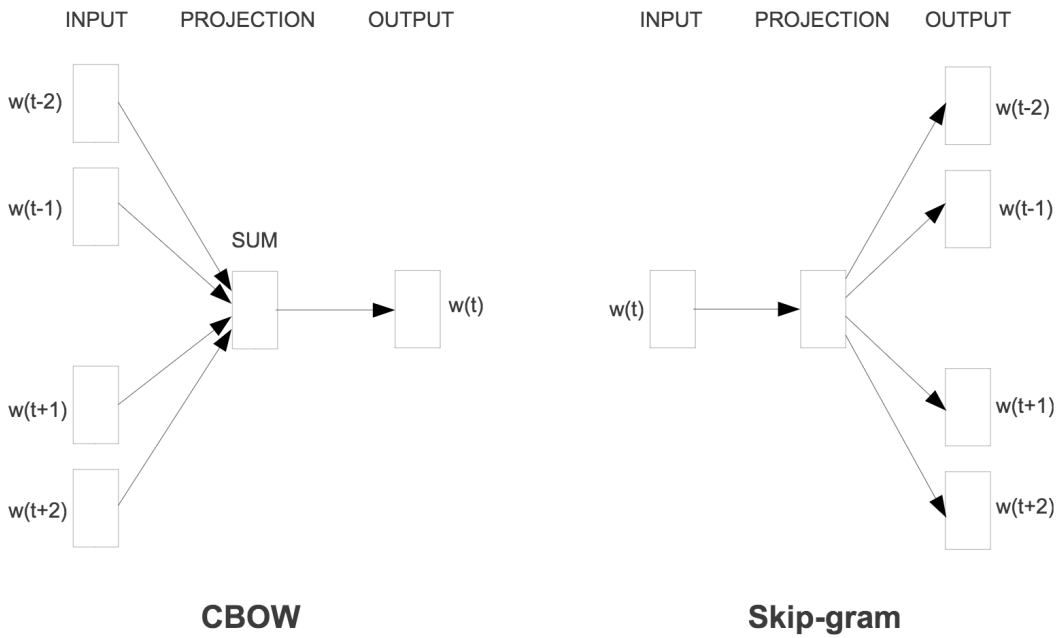
Use running text as implicitly supervised training data

- A word s near apricot
  - Acts as gold ‘correct answer’ to the question
  - “Is word w likely to show up near apricot?”

- No need for hand-labeled supervision

- The idea comes from neural language modeling
  - Bengio et al. (2003)
  - Collobert et al. (2011)

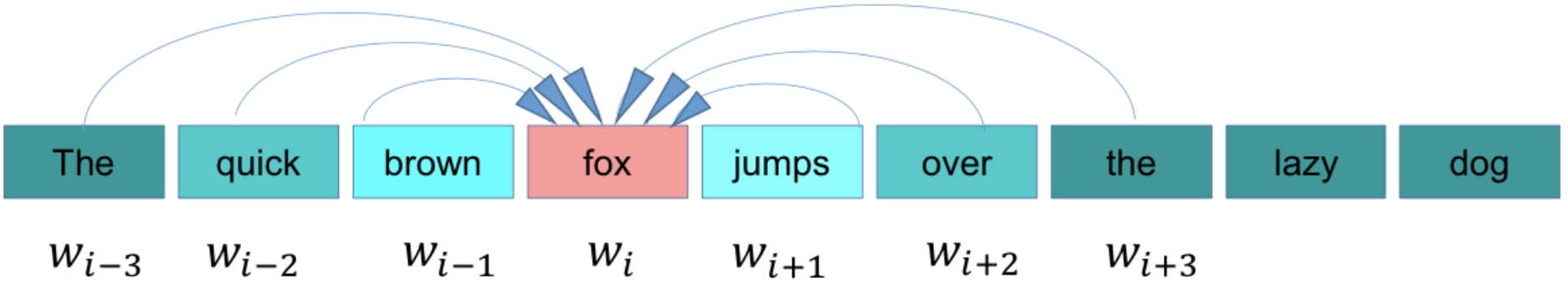
word2vec



- CBOW: predict word, given its close context. Bag-of-words within context
- Skip-gram: predict context, given a word. Takes order into account.

Source: Mikolov, T., Chen, K., Conrado, G., Dean, J. (2013) Efficient Estimation of Word Representations in Vector Space. Proceedings of the Workshop at ICLR, Scottsdale, pp. 1-12.

Continuous bag-of-word model (CBOW)



Current word: w_i
Context: $w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n}$

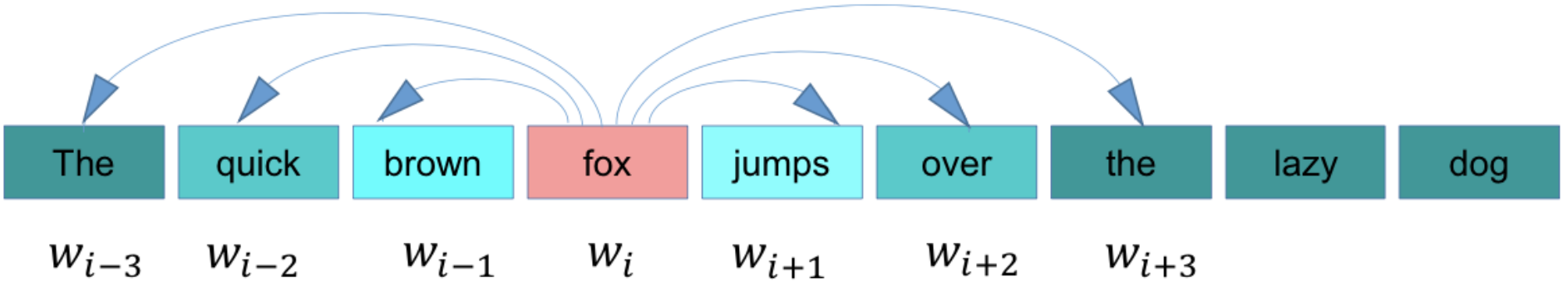
Want to estimate: $P(w_i|Context) = P(w_i|w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n}$

Loss function:

$$\mathcal{L} = -\sum_i \log P(w_i|w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n})$$

Source: Word representations in vector space. Irina Piontkovskaya iPavlov, MIPT 25.10.2018

Skip-Gram model



Current word: w_i
Context: $w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n}$

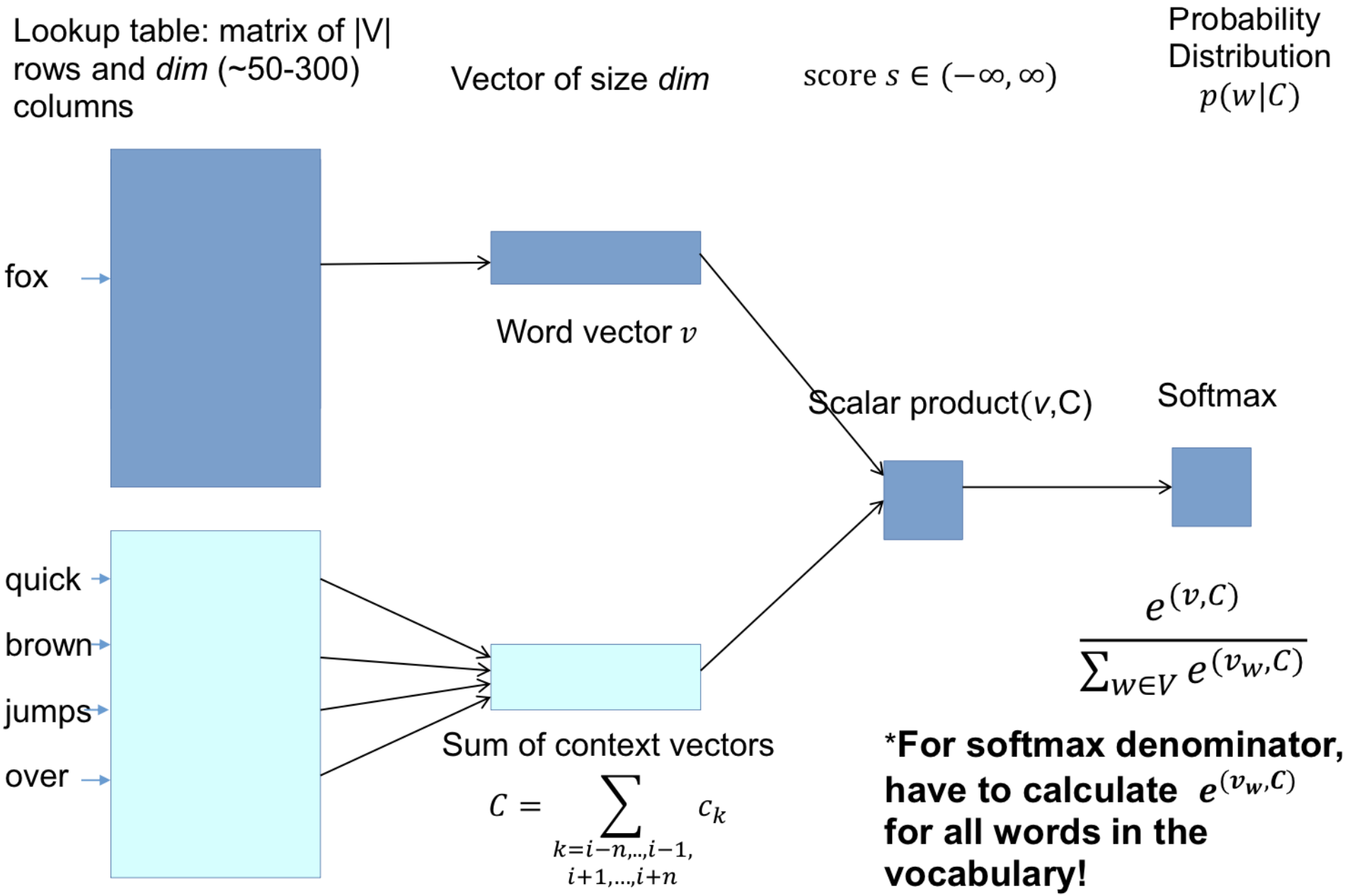
Want to estimate: $P(Context|w_i) = \prod_{w_j \in Context} P(w_j|w_i)$

Loss function:

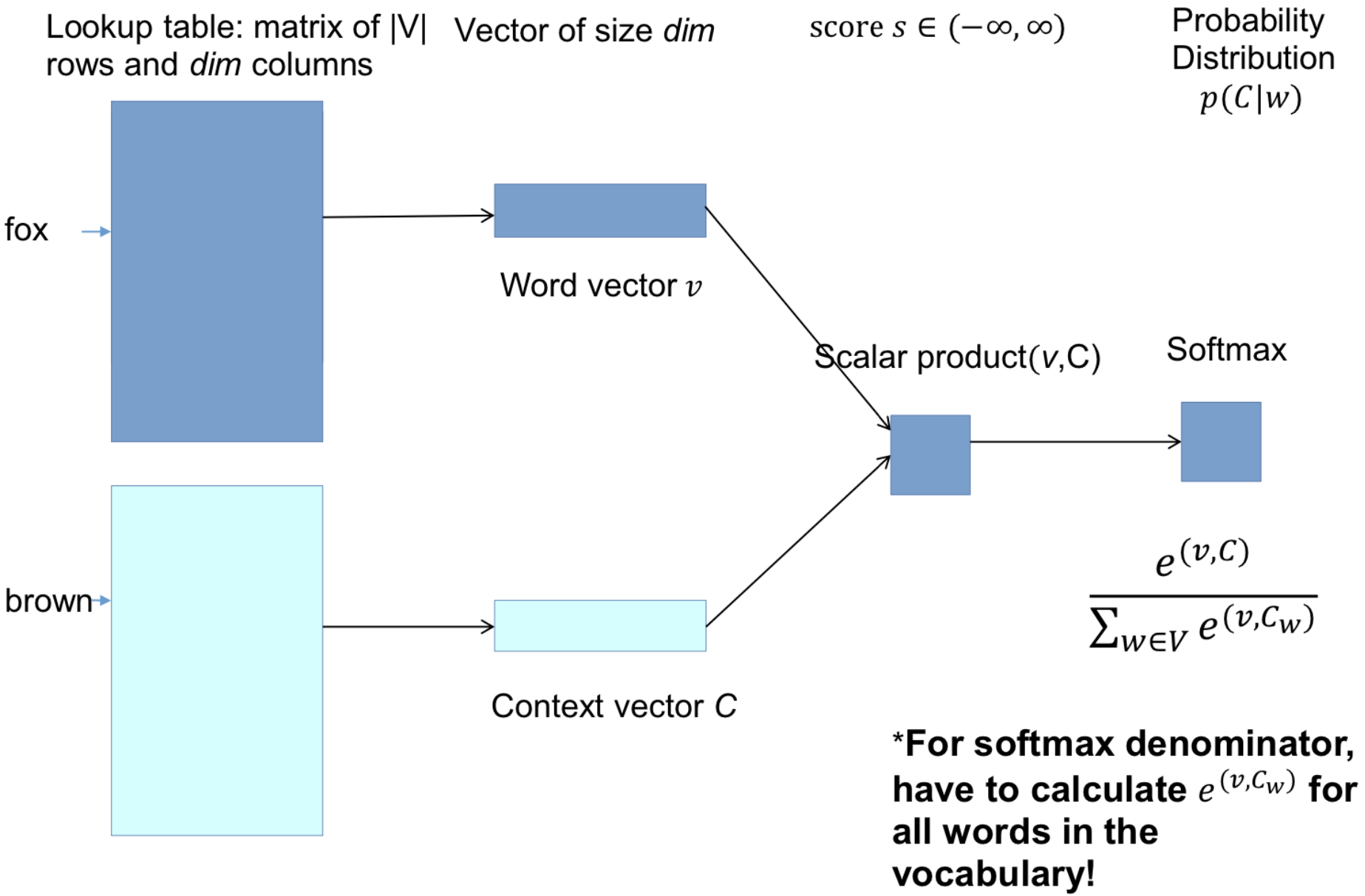
$$\mathcal{L} = - \sum_{\substack{i \\ j=i-n,\ldots,i-1,i+1,\ldots,i+n}} \log P(w_j|w_i)$$

Source: Word representations in vector space. Irina Piontkovskaya iPavlov, MIPT 25.10.2018

CBOW model


Lookup table: matrix of |V| rows and *dim* (~50-300) columns
Vector of size *dim*
score $s \in (-\infty, \infty)$
Probability Distribution $p(w|C)$
fox
quick
brown
jumps
over
Word vector v
Sum of context vectors
$C = \sum_{k=i-n,..,i-1,\ i+1,...,i+n} c_k$
Scalar product(*v*,C)
Softmax
$\frac{e^{(v,C)}}{\sum_{w \in V} e^{(v_w,C)}}$
*For softmax denominator, have to calculate $e^{(v_w,C)}$ for all words in the vocabulary!

Skip-gram model



Training tricks

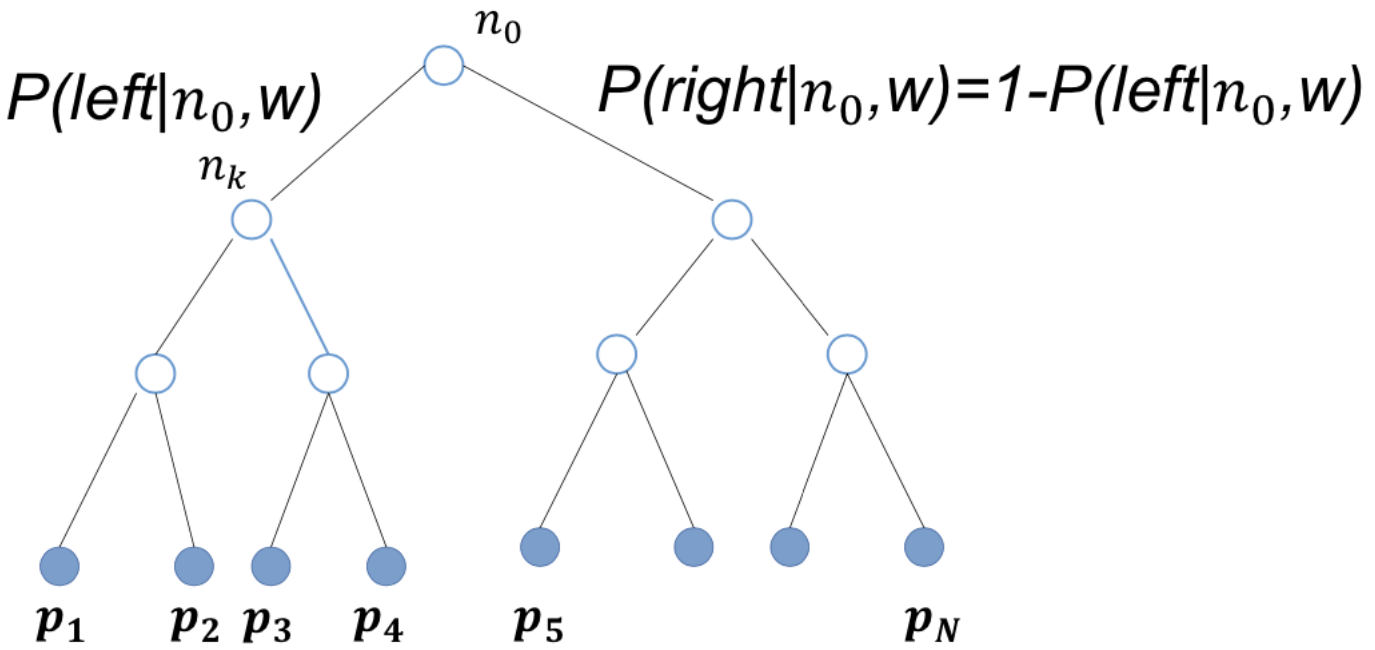
- Softmax issue:

$$\frac{e^{(v,C)}}{\sum_{w \in V} e^{(v,C_w)}}$$

- Denominator in softmax is a sum for the whole dictionary.
- Softmax calculation is required for all (word, context) pairs

Hierarchical softmax

- Idea: represent probability distribution as a tree, where leaves are classes (words in our case).
- $\boldsymbol{p}1, \ldots, \boldsymbol{p}_n$- leaves probabilities
- Mark each edge with probability of choosing this edge, moving down the tree



If $E_1, \ldots, E_k$ - path to leaf *i*, then $\boldsymbol{p}_i = P_{E_1} * \cdots * P_{E_k}$

Source: Word representations in vector space. Irina Piontkovskaya iPavlov, MIPT 25.10.2018

Hierarchical softmax

Hierarchical softmax uses a binary tree to represent all words in the vocabulary. The words themselves are leaves in the tree. For each leaf, there exists a unique path from the root to the leaf, and this path is used to estimate the probability of the word represented by the leaf. “We define this probability as the probability of a random walk starting from the root ending at the leaf in question.”

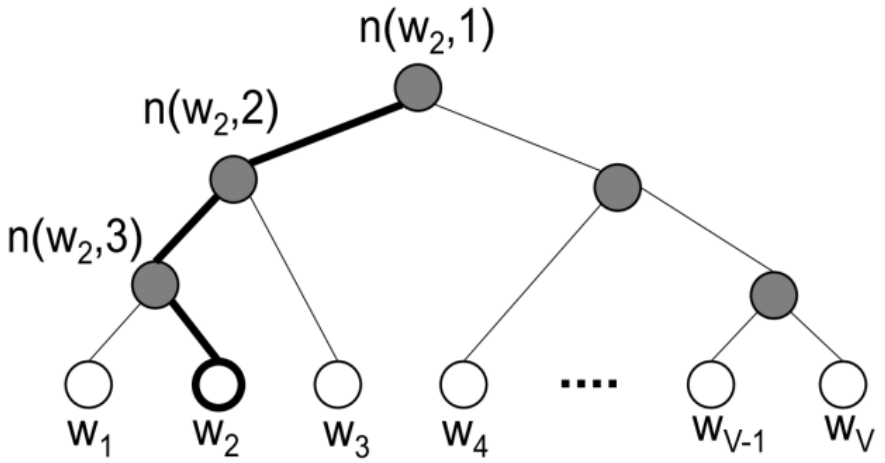


Figure 4: An example binary tree for the hierarchical softmax model. The white units are words in the vocabulary, and the dark units are inner units. An example path from root to w_2 is highlighted. In the example shown, the length of the path $L(w_2) = 4$. $n(w, j)$ means the j-th unit on the path from root to the word w.

Source: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Hierarchical softmax

- Huffman tree is used:

  1) Each word is a separate tree with corresponding prob.

  2) Merge 2 trees having the smallest probabilities. Assign the sum of their probabilities to the new tree.

- This results in:
  - Binary tree with words as leafs.
  - Paths to frequent words are shorter than to rare words; this will give on average (in expectation) the shortest possible paths.
  - Paths can be encoded with binary codes (0 – go to the left, 1 – go to the rights); this will be Huffman codes for words - optimal prefix codes for lossless data compression.

Negative sampling

- Another methods to avoid softmax calculation:
- Consider for each word w binary classifier: if given word C is good context for w, or not
- For each word, sample negative examples (negative count = 2...25)

$$Sampling\ probability(n) = P(n)^{\frac{3}{4}}$$

$P(u)$ – *word probability in the corpus*

- Loss function:

$$\mathcal{L} = -\sum_{w,C}\left(\log\sigma((v_w, u_C)) + \sum_{n\in Neg}\log\sigma(-(v_w, u_n))\right)$$

Logistic regression objective

Source: Word representations in vector space. Irina Piontkovskaya iPavlov, MIPT 25.10.2018

word2vec: Skip-Gram

- word2vec provides a variety of options (SkipGram/CBOW, hierarchical softmax/negative sampling, …). We will look more closely at:
  - "skip-gram with negative sampling" (SGNS)

- **Skip-gram training:**

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

  2) Randomly sample other words in the lexicon to get negative samples

  3) Use logistic regression to train a classifier to **distinguish those two cases**

  4) Use the weights as the embeddings

Skip-Gram Training Data

Training sentence: Assume context words are those in +/- 2 word window.

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

Given a tuple (t,c) = target, context

- (apricot , jam)
- (apricot, aadvark)

Return probability that c is a real context word:

$P(+|t,c)$

$P(-|t,c) = 1-P(+|t,c)$

Positive and negative samples

Training sentence: Asssume context words are those in +/- 2 word window.

... lemon, a tablespoon of **apricot** jam a pinch ...
c1 c2 target c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Choosing noise words

- Could pick w according to their unigram frequency P(w)
- More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

- α= ¾ works well because it gives rare noise words slightly higher probability
- To show this, imagine two events p(a)=.99 and p(b) = .01:

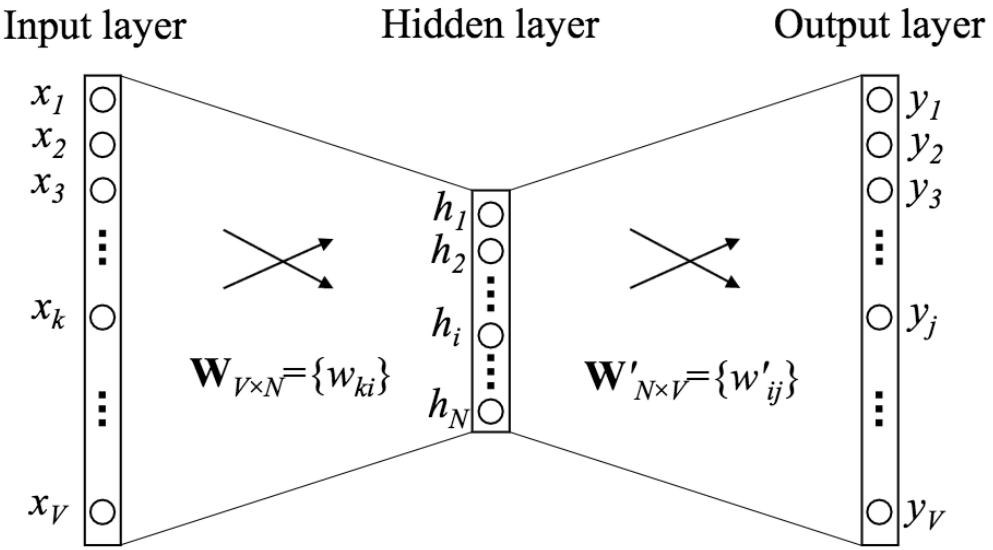
$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

Other (very) important details

- Only tokens occurring >=*min_cnt* times in the corpus are added to the vocabulary. Thus, the model doesn't see tokens occurring <*min_cnt* times in the corpus during training!
- Subsampling (downsampling) of frequent tokens: skip tokens with probability $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$
  - Words with frequency > *sample=t* (1e-5 – 1e-3) are downsampled.
  - Skipping tokens happens during reading input file. Thus, the window size is effectively increased.
- Dynamic window size: for each center word the window size is sampled uniformly from 1 to *window*. This increases the importance of the nearest tokens.

Embeddings: weights to/from projection layer



- W_{in} and W_{out}^T: V x N matrices
- every word is embedded in N dimensions, which is the size of the hidden layer
- Note: embeddings for words and contexts differ

What does the model learns

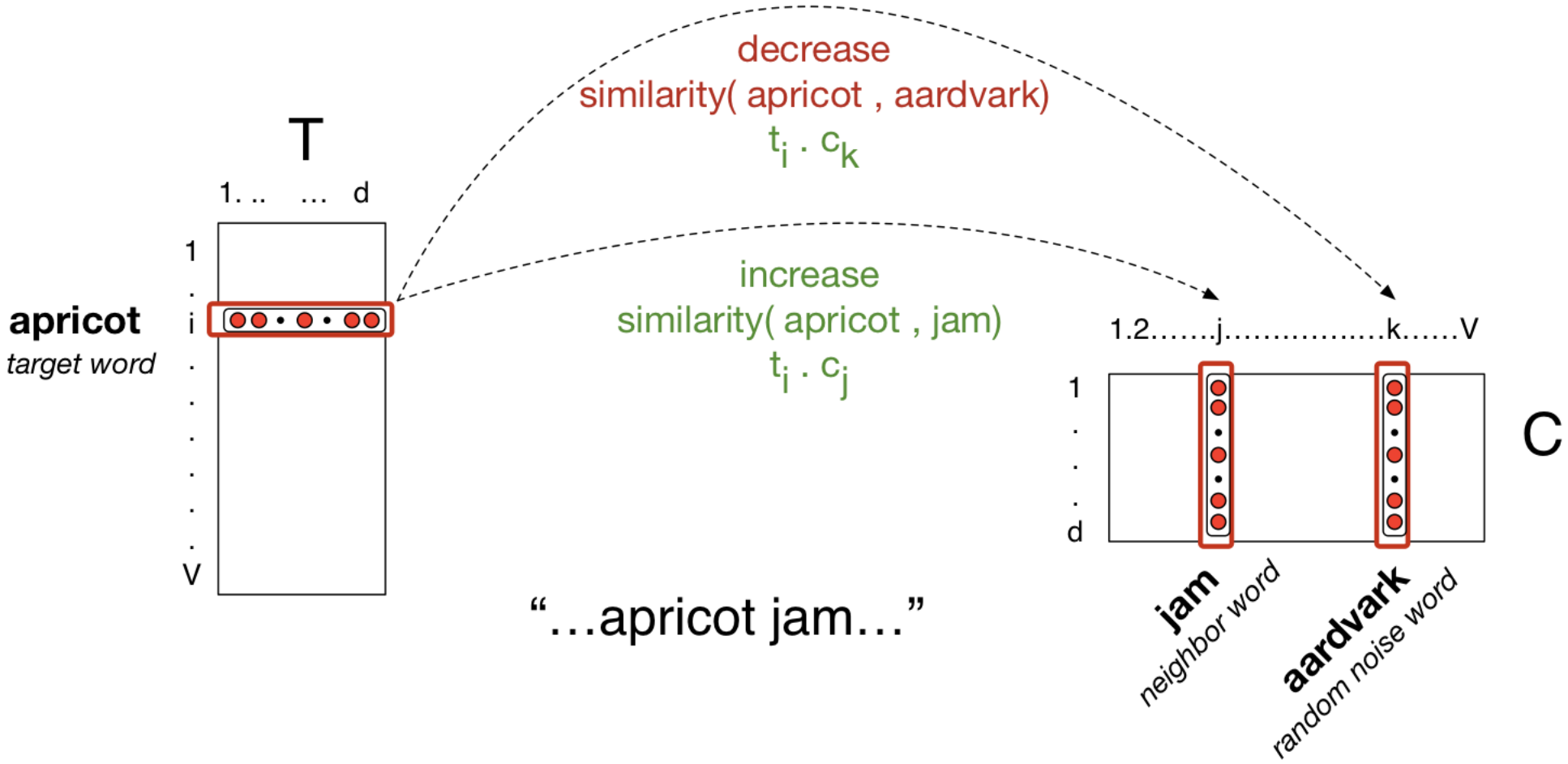
- The model tries to increase the scalar product of good (word, context) pairs and decrease for the bad ones.
- How to increase the scalar product of two vectors?
- increase lengths of one of the vectors: in that case, all scalar products of this vector are increasing

$$(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x}\| \|y\| \cos \varphi(x, y)$$

  - decrease angle between vectors
  - word vector tends to have small angle with its context vector
  - vectors which are frequently occur in the same context tend to be close to each other

What does the model learns

- The skip-gram model tries to shift embeddings so the target embeddings (here for apricot) are closer to (have a higher dot product with) context embeddings for nearby words (here jam) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (here aardvark).



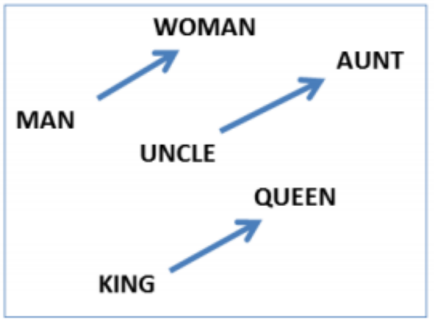
Source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

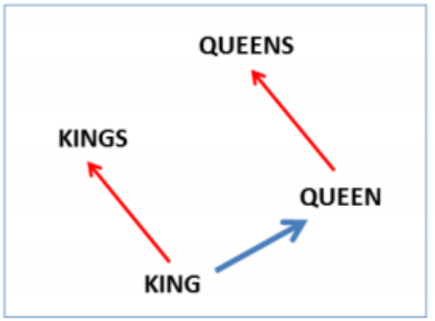
Vector Algebra for Analogy Questions

- Observation: words in the same relation have similar vector differences

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

- Syntactic analogy questions:
  “a is to b as c is to ...”
  (rough is to rougher as tough is to ...)





Source: Mikolov, T., Yih, W., Zweig, G. (2013): Linguistic Regularities in Continuous Space Word Representations. Proc. HLT-NAACL '13, pp. 746-751

word2vec vector length

L2 norm is smaller for rare words

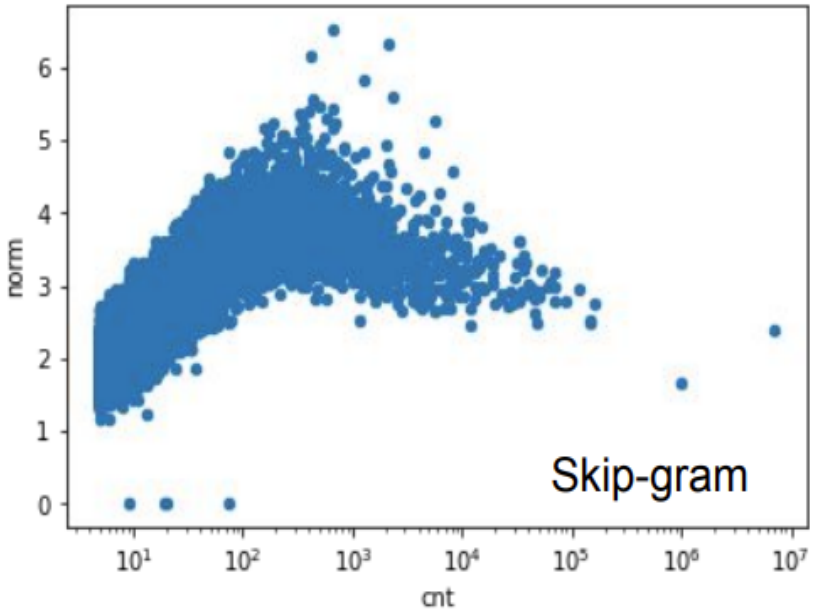
compensates for less reliable embeddings in weighted sum

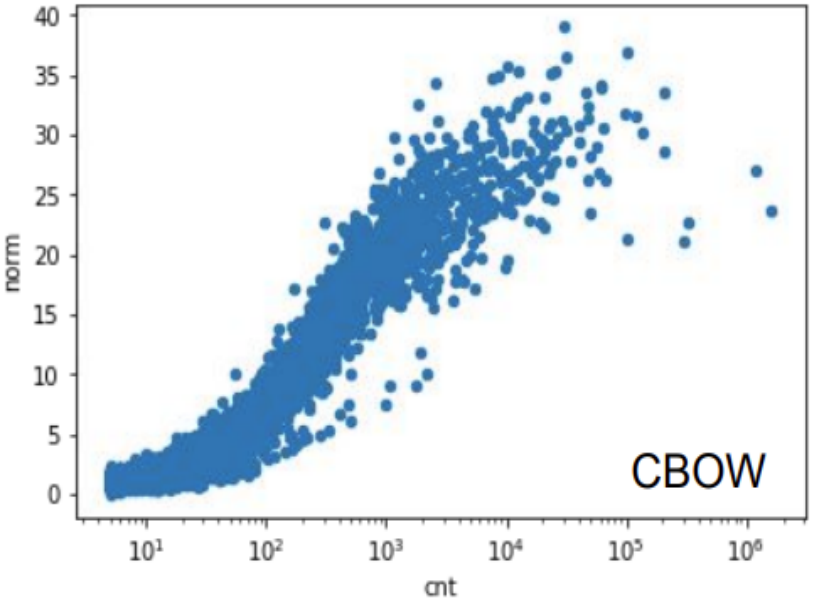
10K random words from 2 models

librusec (150G)
min count 5
=> 3M words

10 neg. samples
downsample
frequency 1e-5

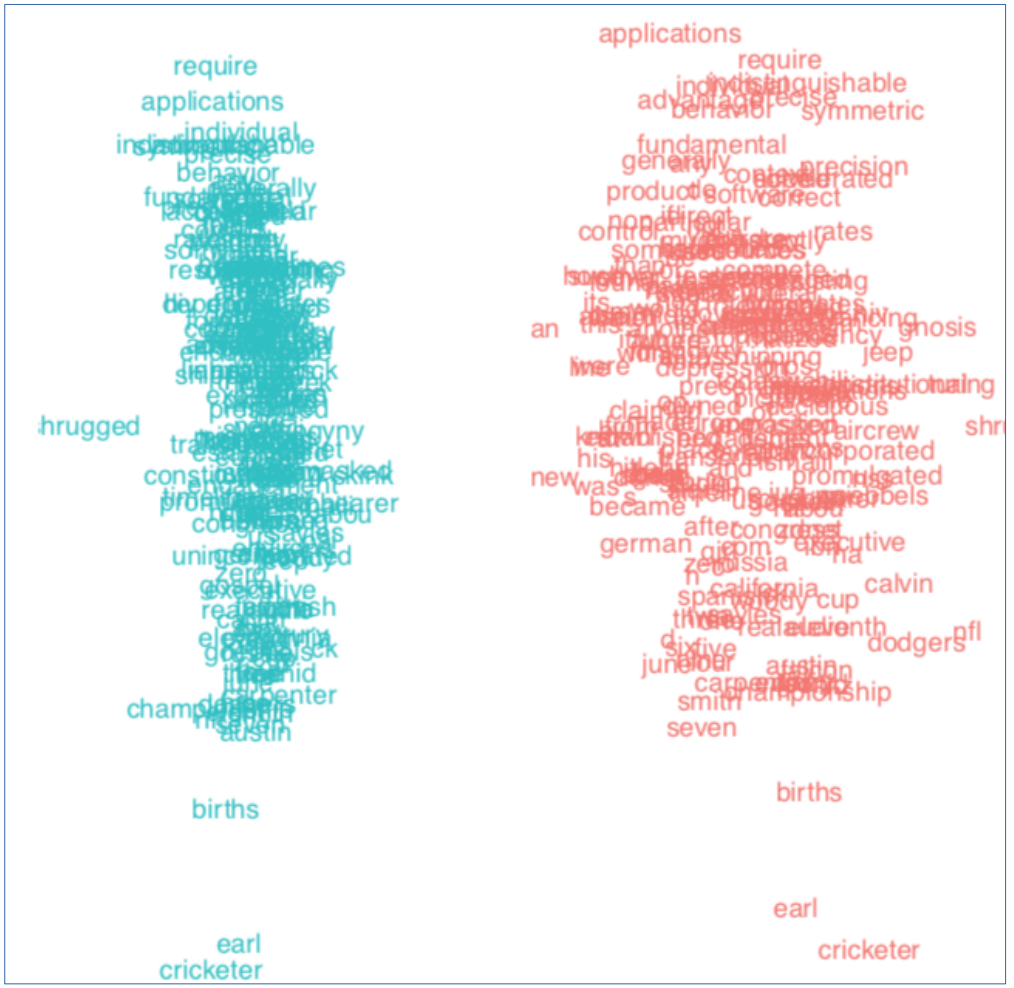
window 10
size 200
3 epochs





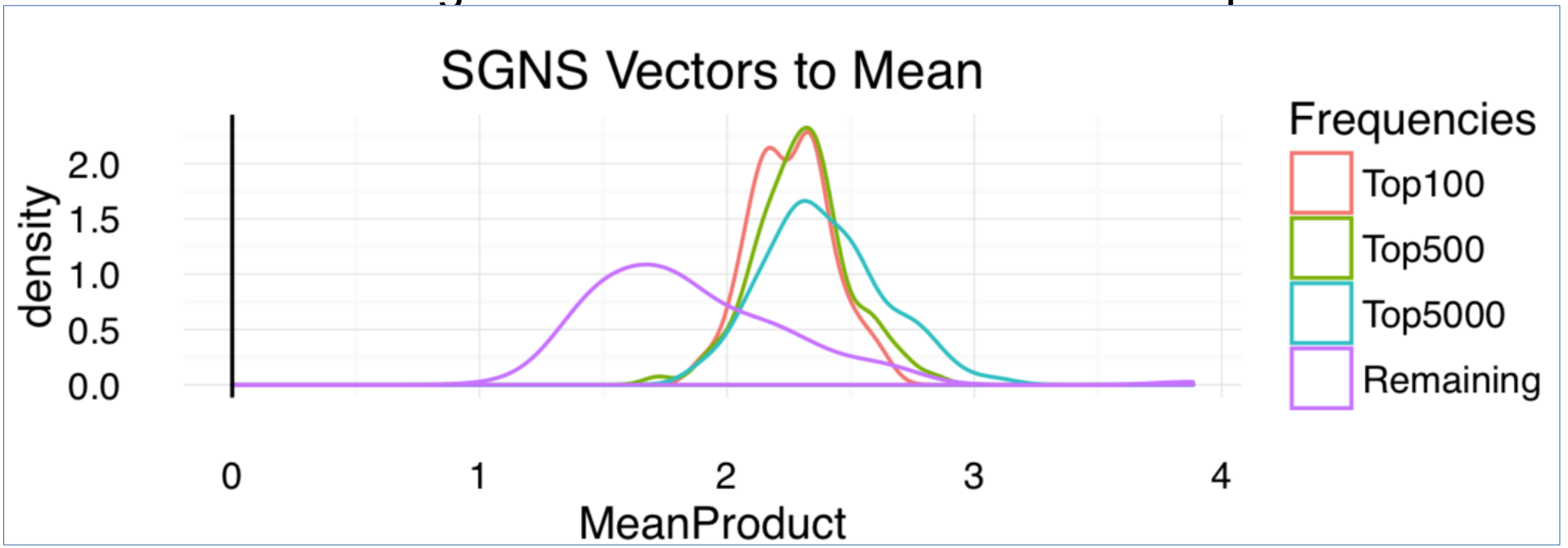
Arefyev et al. How much does a word weigh? Weighting word embeddings for word sense induction, 2018

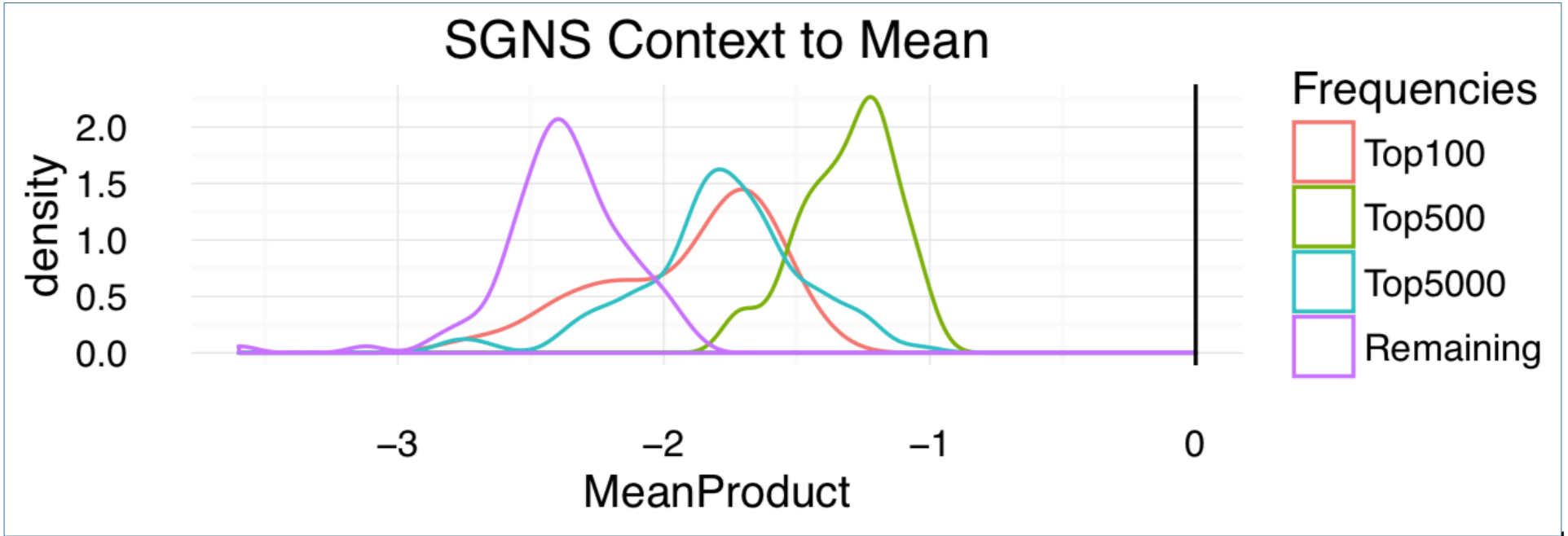
PCA (left) and t-SNE (right) for SGNS word and context vectors



Figures from David Mimno, Laure Thompson, 2017. The strange geometry of skip-gram with negative sampling

Dot-products with the average word vector: Embeddings do not cover the entire vector space!

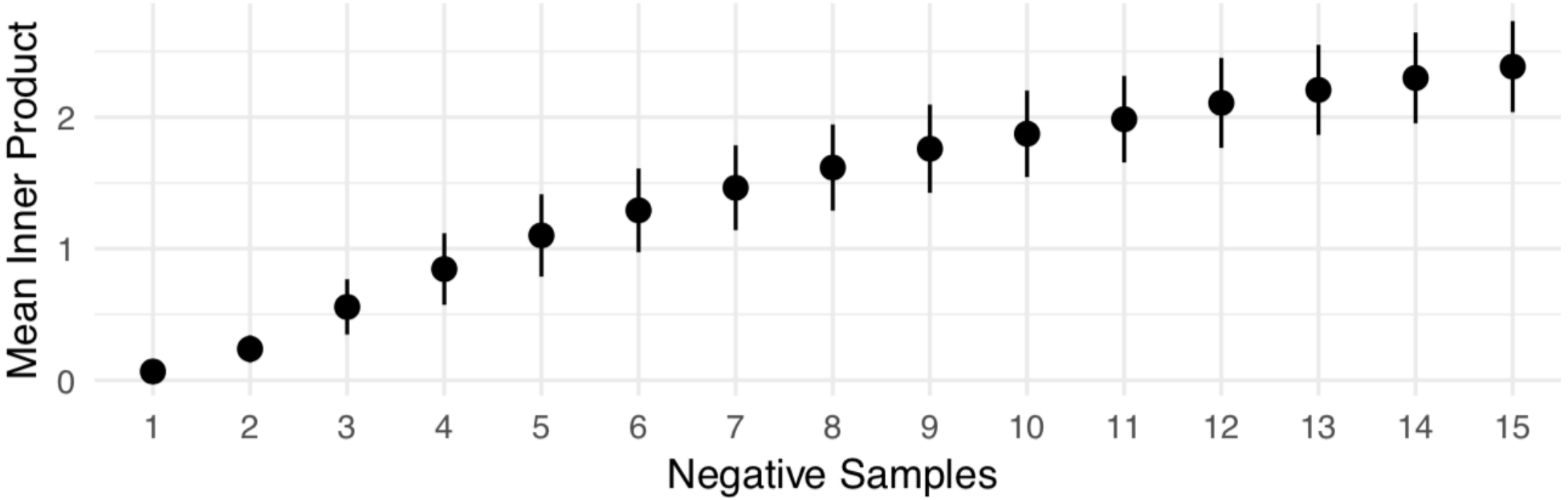




Figures from David Mimno, Laure Thompson, 2017. The strange geometry of skip-gram with negative sampling

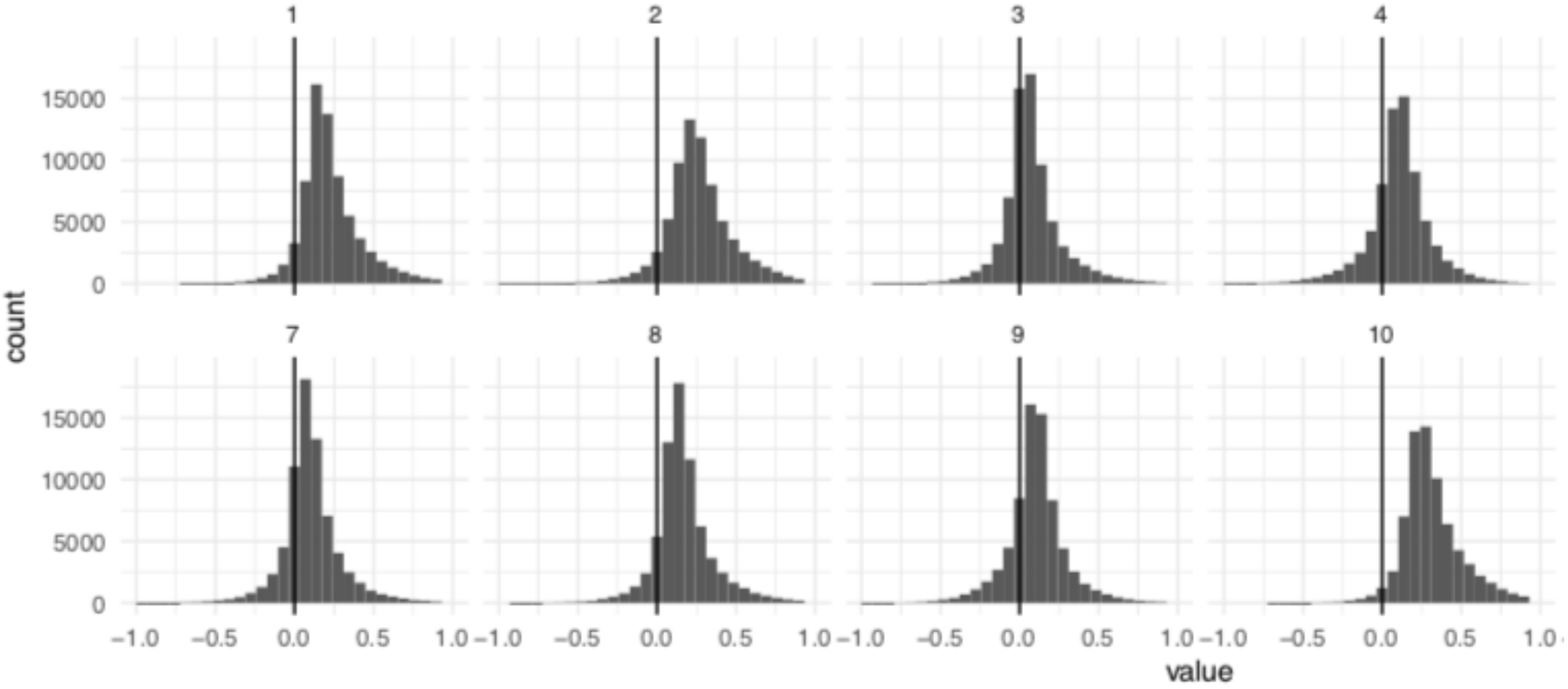
Dot-products with the average word vector: Embeddings do not cover the entire vector space!

- The more negative contexts are sampled, the more narrow cone the word embeddings are concentrated in.



Figures from David Mimno, Laure Thompson, 2017. The strange geometry of skip-gram with negative sampling

Values of some of the components (1-4, 7-10), each component is multiplied by the sign of its mean value:
each component has either mostly positive or mostly negative values



Figures from David Mimno, Laure Thompson, 2017. The strange geometry of skip-gram with negative sampling

Hidden representations of recent LMs/MLMs do not occupy the whole space either!

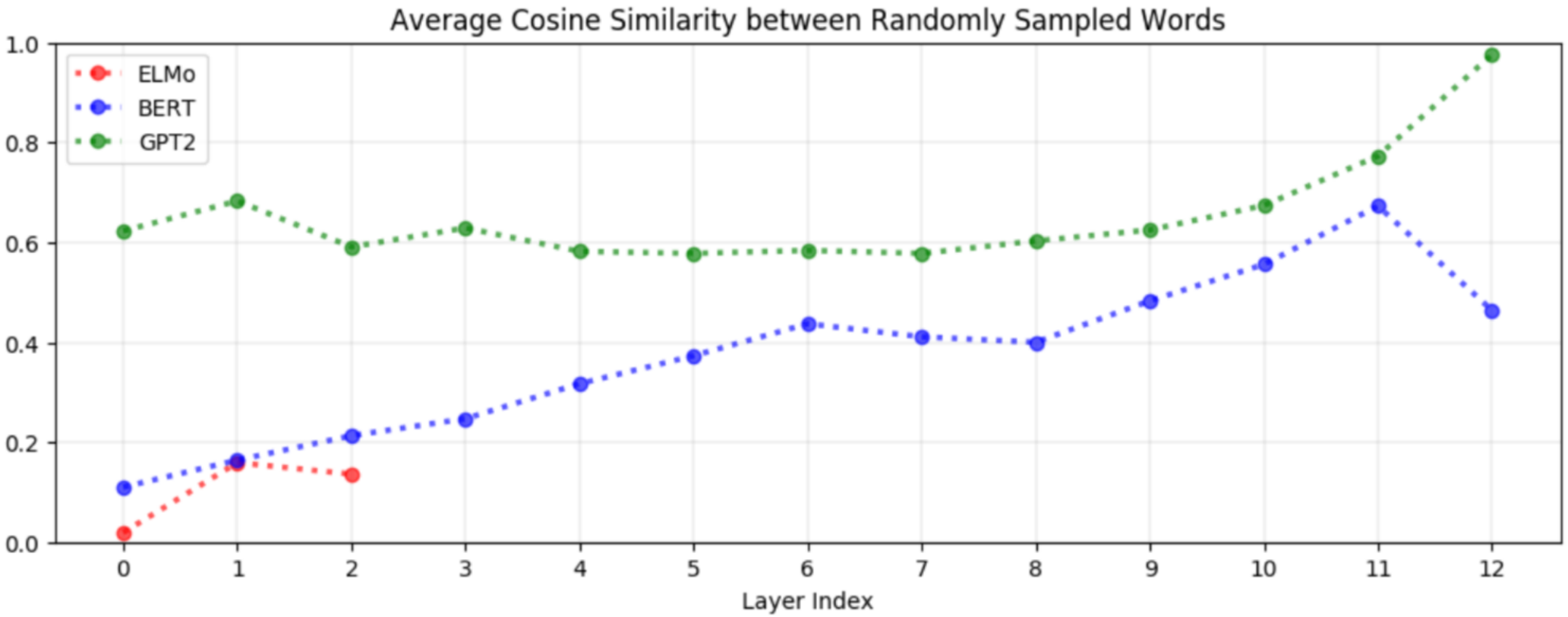


Figure from Kawin Ethayarajh. How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings, 2019.

For practical tasks adjust your thresholds according to your data!
Does cos(w1,w2)=0.95 denotes high or low similarity?

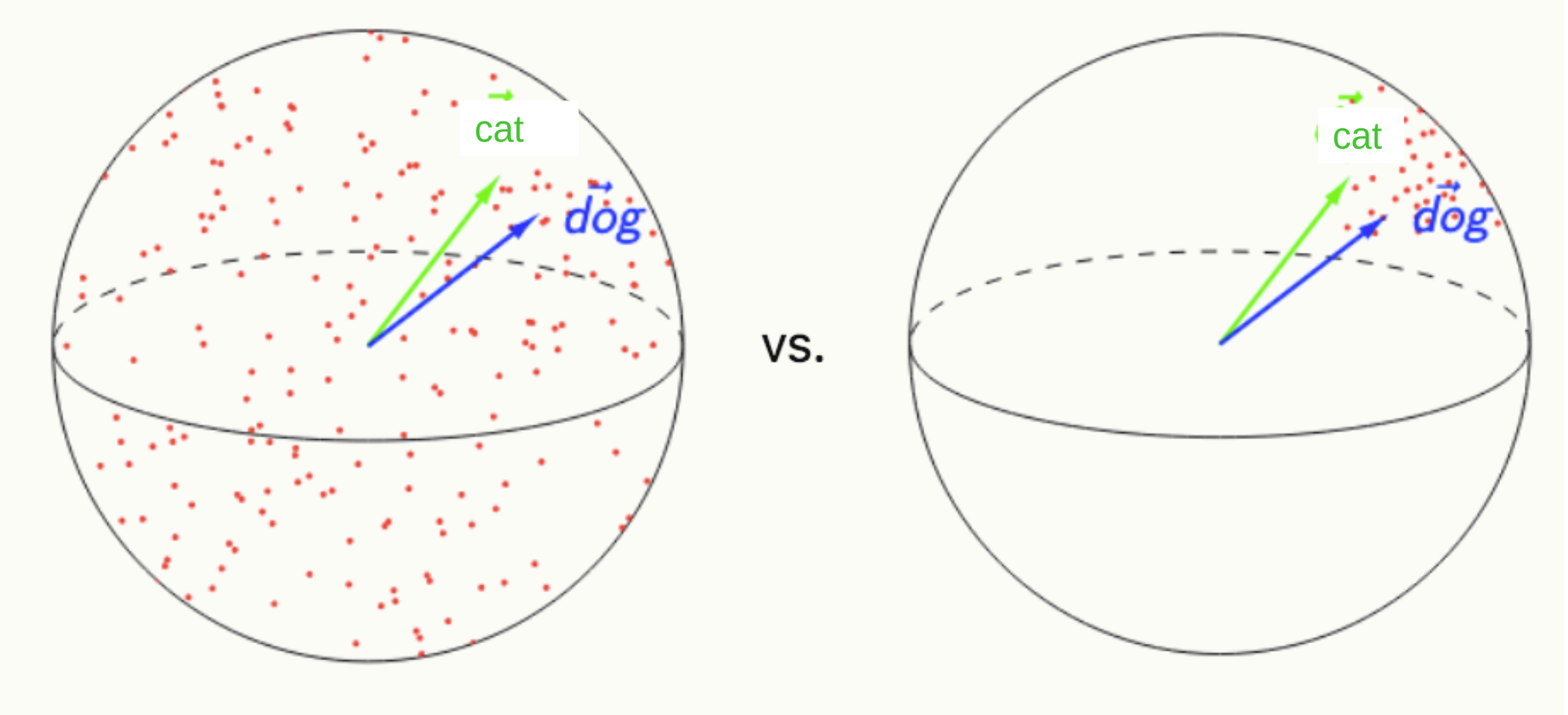


Figure from https://kawine.github.io/blog/nlp/2020/02/03/contextual.html

SNGS and word-context matrix

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = PMI(w_i, c_j) - \log k$$

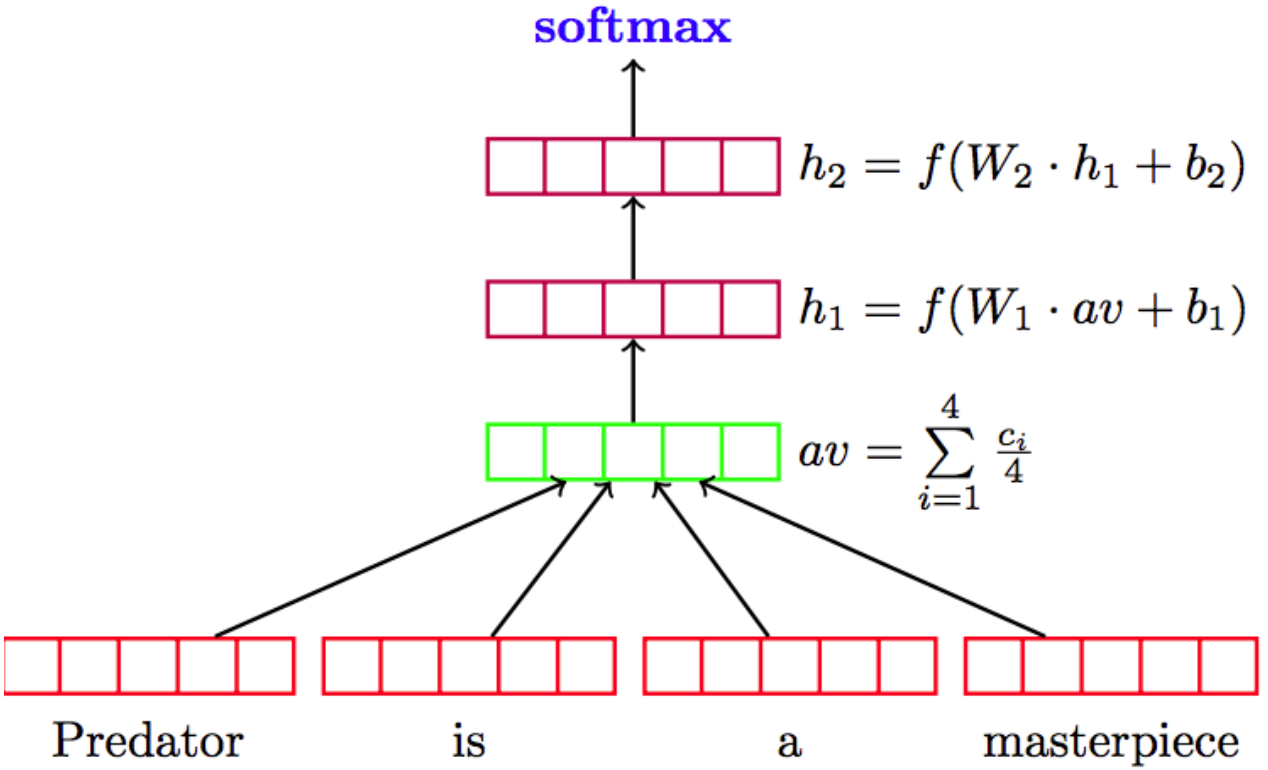
(Levy&Goldberg. Neural Word Embedding as Implicit Matrix Factorization) show that SGNG implicitly approximately factorizes the PMI matrix shifted by log(k), k – number of negative samples

- In their experiments, SVD factorization is comparable with SGNS on semantic similarity datasets, but worse on analogy datasets
- In SGNS, it is naturally more important to better approximate cells corresponding to frequent word-context pairs than the rare ones. For SVD they are equally important.
  - More on that in GLOVE

Pooling / averaging of word vectors

- The straightforward approach of averaging word embeddings of all words in a document is a crude document vector
  - May work for simple tasks like coarse-grained topic categorization (food VS politics)
  - can be improved using weighted average (TF-IDF, chi2), removing stopwords, removing or leaving only a few first principle components

DAN



- Deep Averaging Network (DAN) is a FFNN using as an input representation the average of word embeddings for all words in a document
- Word embeddings are initialized with SGNS, then fine-tuned with all other weights

Iyyer et al. Deep Unordered Composition Rivals Syntactic Methods for Text Classification, 2015

DAN

- Accuracy was comparable with NNs of 2015
- On IMDB no better than a logistic regression on BON

Model	RT	SST fine	SST bin	IMDB
DAN-ROOT	—	46.9	85.7	—
DAN-RAND	77.3	45.4	83.2	88.8
DAN	80.3	47.7	86.3	89.4
NBOW-RAND	76.2	42.3	81.4	88.9
NBOW	79.0	43.6	83.6	89.0
BiNB	—	41.9	83.1	—
NBSVM-bi	79.4	—	—	91.2
RecNN*	77.7	43.2	82.4	—
RecNTN*	—	45.7	85.4	—
DRecNN	—	49.8	86.6	—
TreeLSTM	—	**50.6**	86.9	—
DCNN*	—	48.5	86.9	89.4
PVEC*	—	48.7	87.8	**92.6**
CNN-MC	**81.1**	47.4	**88.1**	—
WRRBM*	—	—	—	89.2

Iyyer et al. Deep Unordered Composition Rivals Syntactic Methods for Text Classification, 2015

Some other popular dense word embedding methods

- **word2vec** (Mikolov et al., 2013)
  https://code.google.com/archive/p/word2vec/

- **GloVe** (Pennington et al., 2014)
  http://nlp.stanford.edu/projects/glove

- **fastText** (Bojanowski et. al., 2017)
  http://www.fasttext.cc

Global Vectors (GloVe)

- Objective function:

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

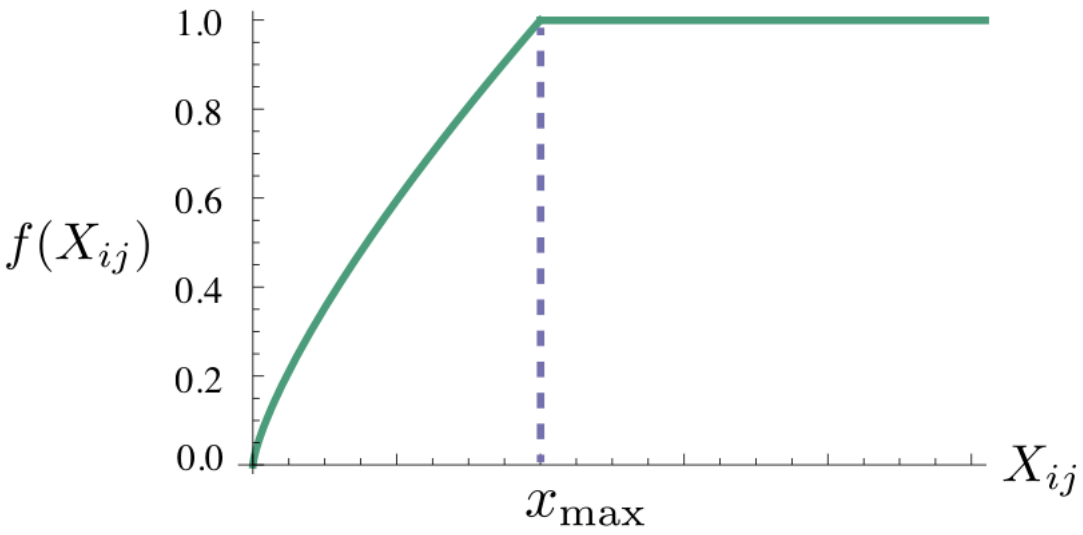
X_{ij} – matrix of word-word co-occurrence counts

$w \in \mathbb{R}^d$ – word vectors

$\tilde{w} \in \mathbb{R}^d$ – context vectors

- Weighting function:

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

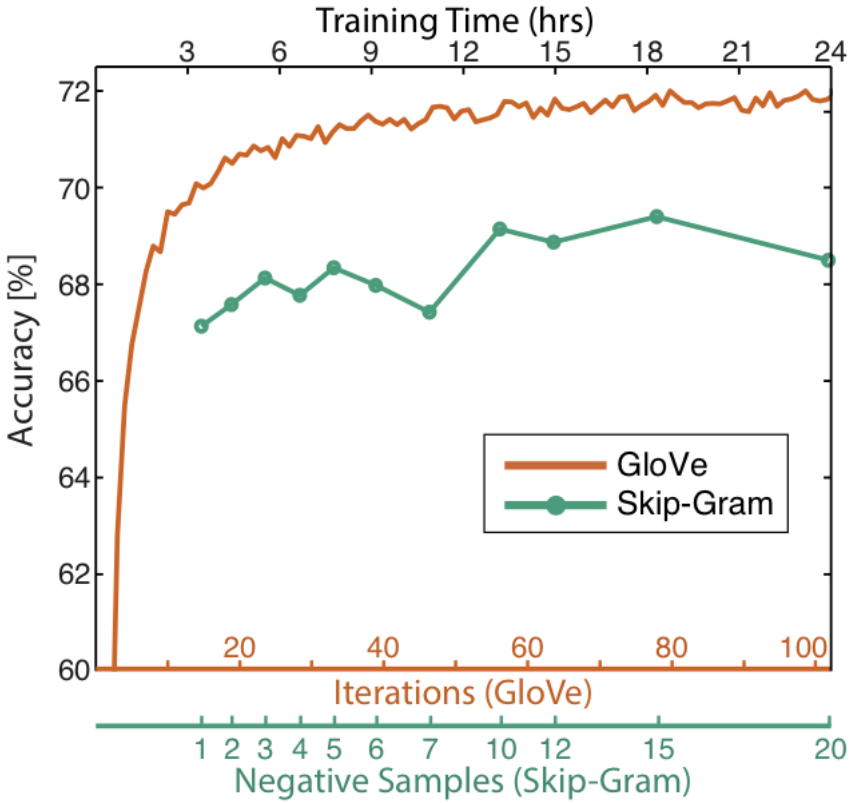


Source: Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)

Global Vectors (GloVe)

- Selling points:
  - Fast training
  - Scalable to huge corpora
  - Good performance even with small corpus, and small vectors

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**



Source: Adopted from Richard Socher, CS224n 2016 course and Pennington, J., Socher, R., & Manning, C. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

Word embeddings for the Russian language

SGNS for 3M tokens trained on 150Gb of texts (mostly) in Russian
(Arefyev et al. Evaluating Three Corpus-Based Semantic Similarity Systems for Russian, 2015):

https://zenodo.org/record/400631#.Yh97KxPP2F0

Results count: 226

1 языкознание
2 лингвистики
3 филология
4 лингвистике
5 лингвистическая
6 социолингвистика
7 языкознания
8 семиотика
9 лексикология
10 социология
11 лингвокультурология
12 семантика
13 культурология
14 лингвистику
15 компаративистика
16 фонология
17 семиотики
18 этнология
19 антропология
20 лингвистикой

Show next 20 results

Word embeddings for the Russian language

- *RusVectores* project: https://rusvectores.org/ru/
  - Lots of models trained on different corpora
  - But most of them built on lemmatized and POS-tagged texts!
    - Lemmatization and POS-tagging errors are frequent
    - Often you just want embeddings and don't want doing lemmatization and POS-tagging
- *FastText* project: https://github.com/facebookresearch/fastText/tree/master
  - provides embeddings for 157 languages
  - Provides alignments between vector spaces for different langauges