Neural Natural Language Processing

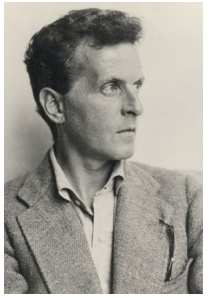
Lecture 3: Word and document embeddings

Plan of the lecture

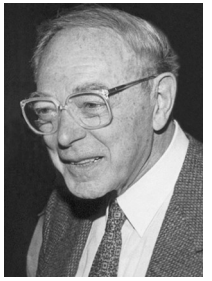
- **Part 1**: Distributional semantics and vector spaces.
- **Part 2**: word2vec and doc2vec models.
- **Part 3**: Other models for word and document embeddings.

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/

Reminders from linear algebra

$$\text{dot-product}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

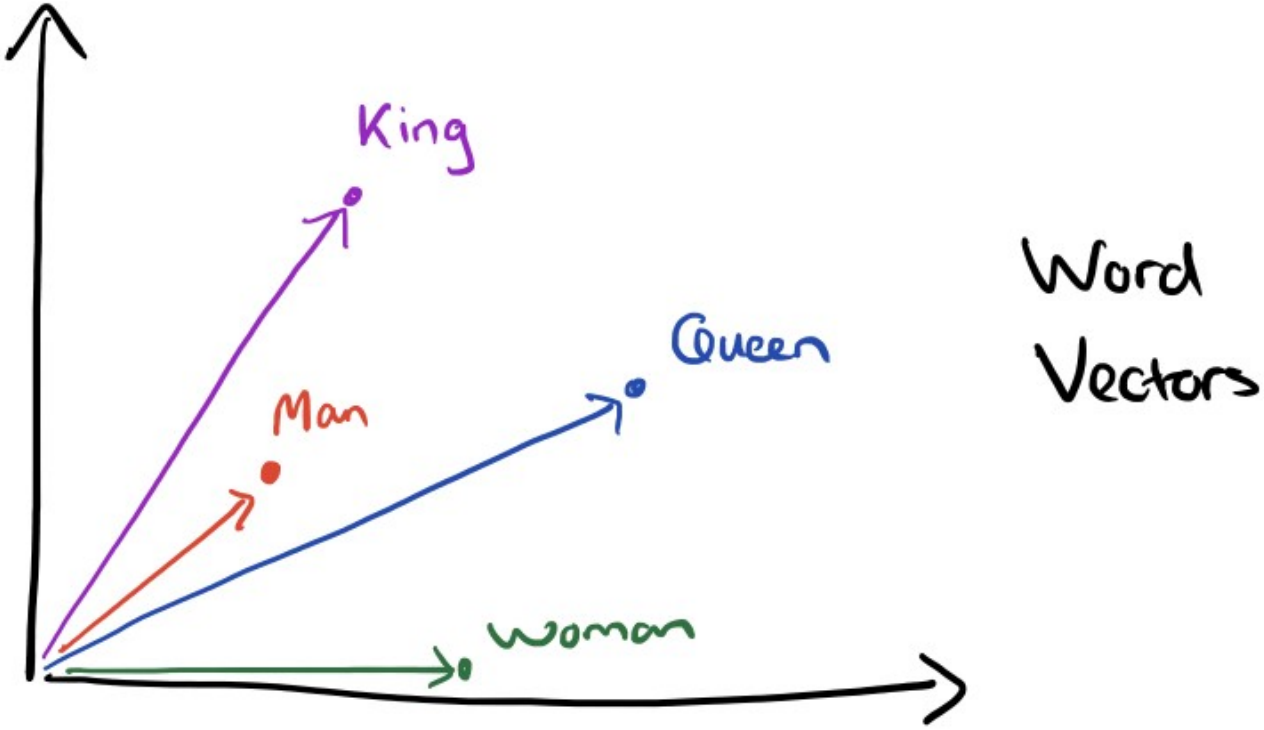
$$\text{vector length } |\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

$$\text{cosine}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

- -1: vectors point in opposite directions
- +1: vectors point in same directions
- 0: vectors are orthogonal
- If values are non-negative, cosine ranges 0-1

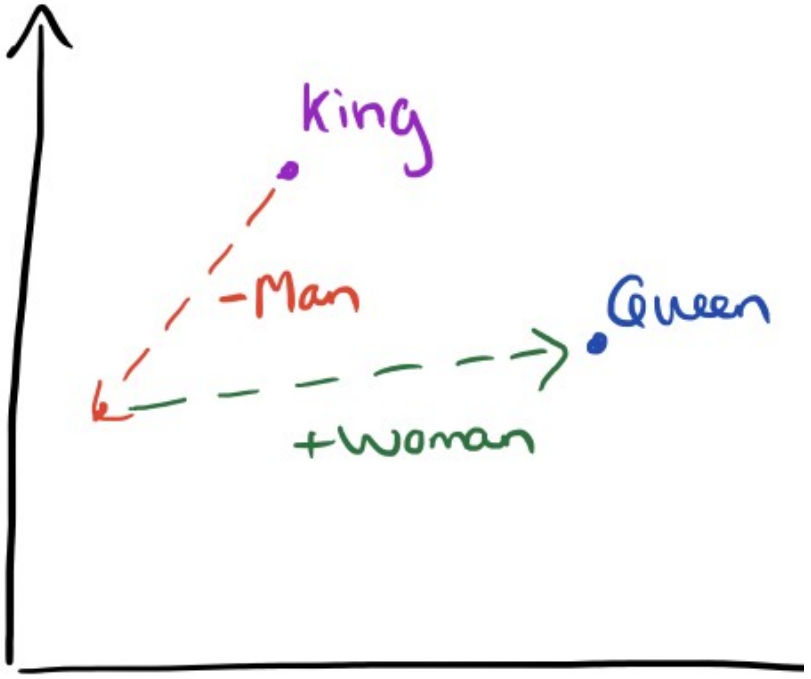
Cosine as a similarity measure

- 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/

Plan of the lecture

- **Part 1**: Distributional semantics and vector spaces.
- **Part 2**: word2vec and doc2vec models.
- **Part 3**: Other models for word and document embeddings.

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

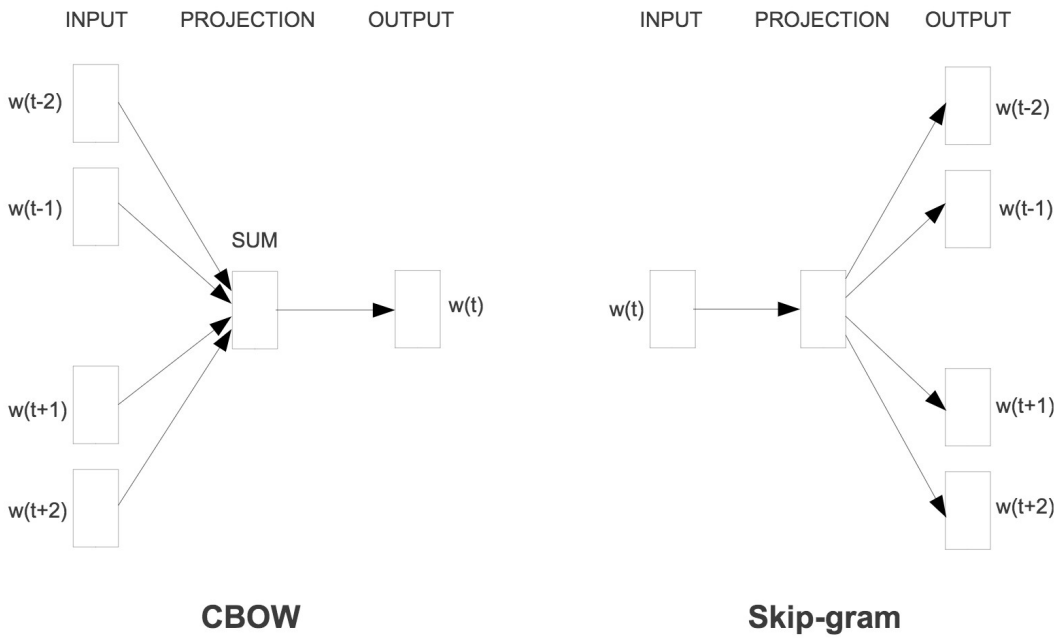
Use running text as implicitly supervised training data

- 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)

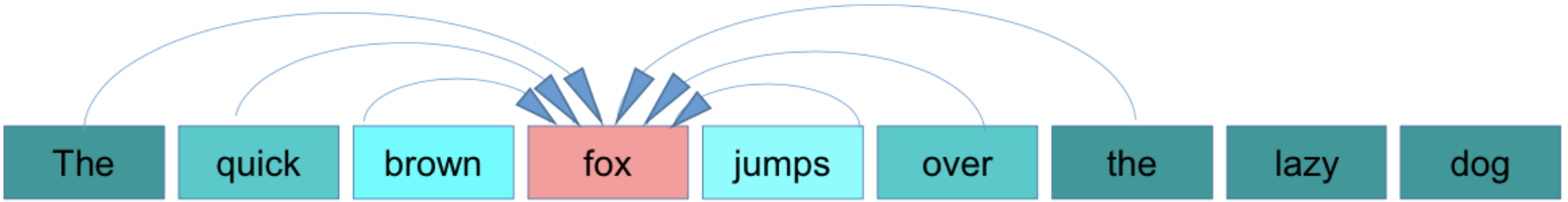
word2vec



- 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: $w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n}$

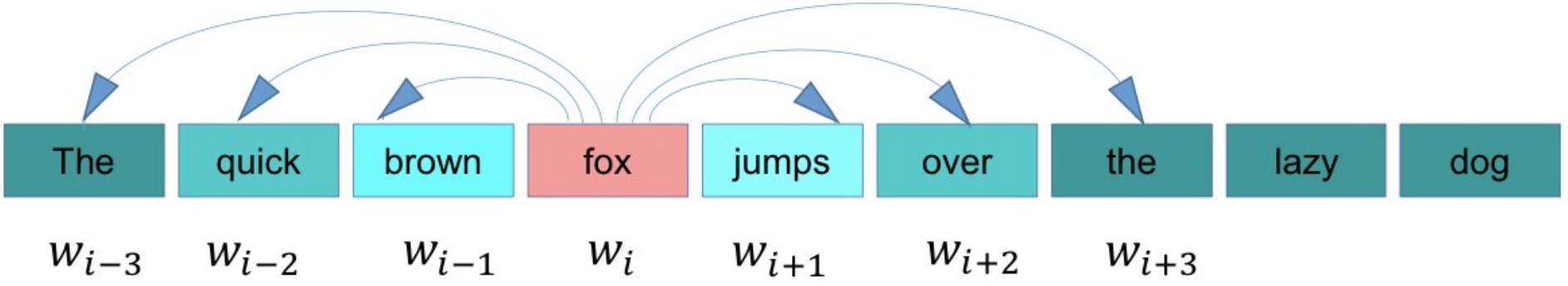
Want to estimate: $P(w_i|Context) = P(w_i|w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n}$

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: $w_{i-n}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+n}$

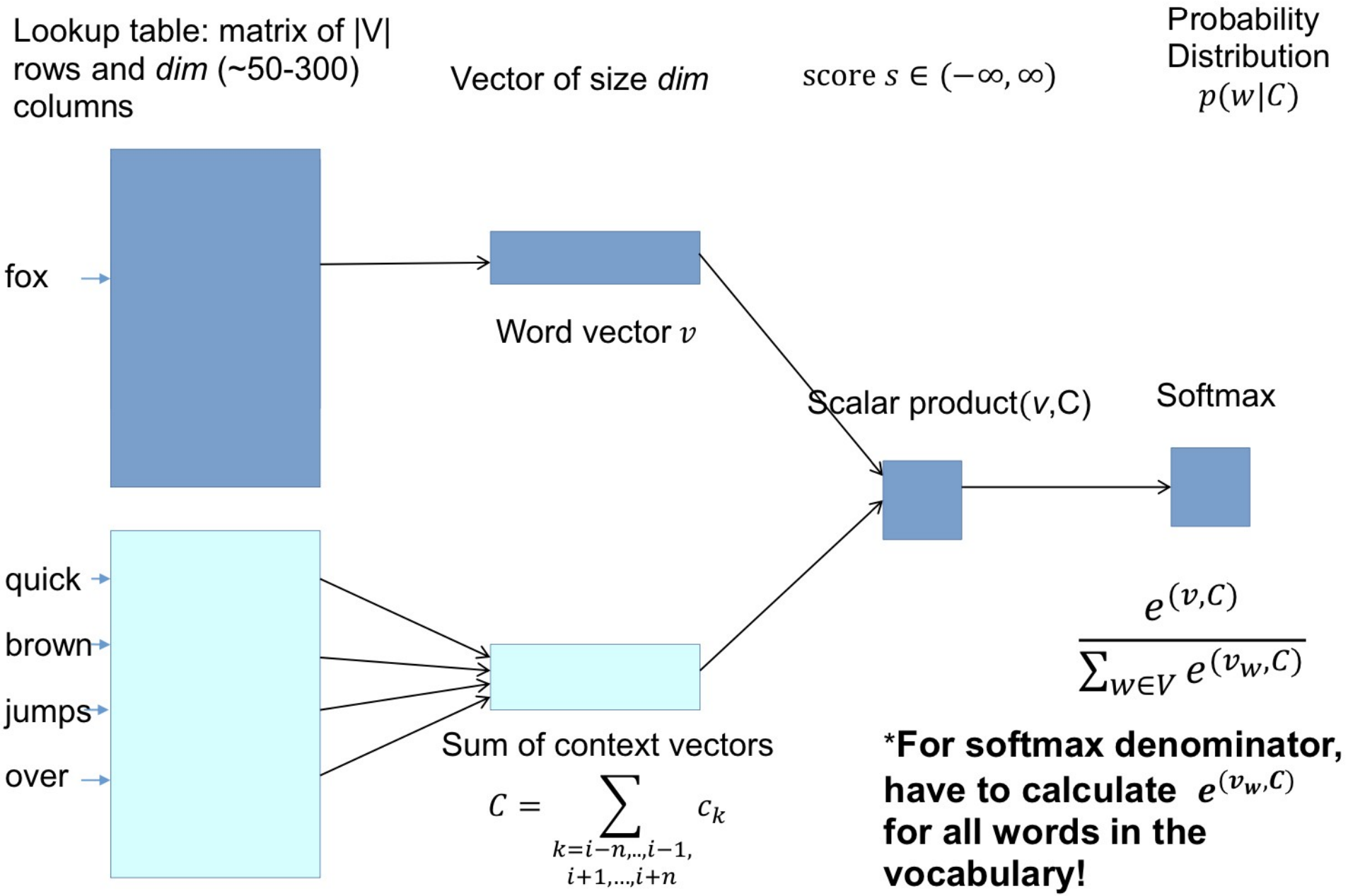
Want to estimate: $P(Context|w_i) = \prod_{w_j \in Context} P(w_j|w_i)$

Loss function:

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

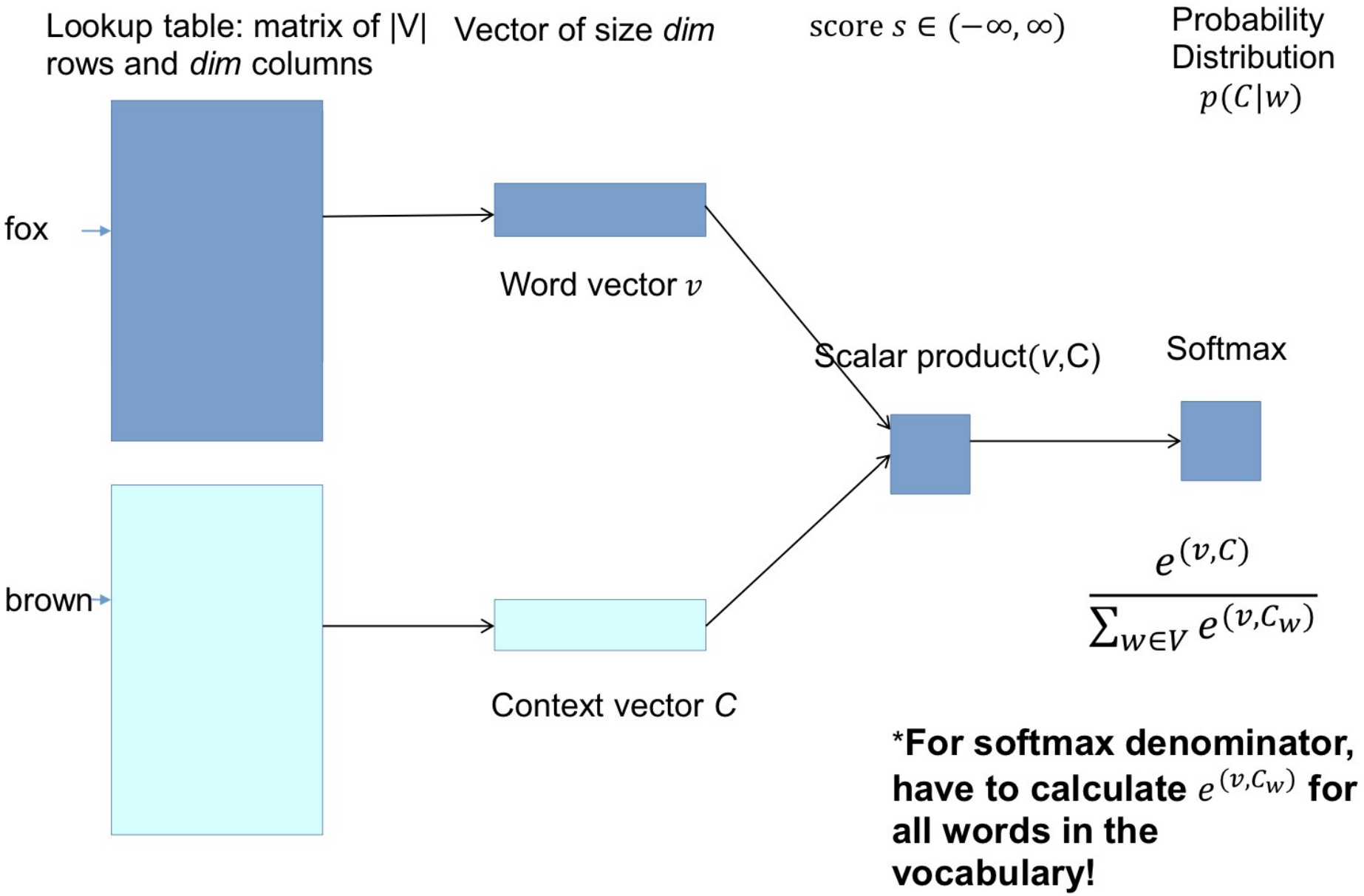
Source: Word representations in vector space. Irina Piontkovskaya iPavlov, MIPT 25.10.2018

CBOW model



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

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

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.”

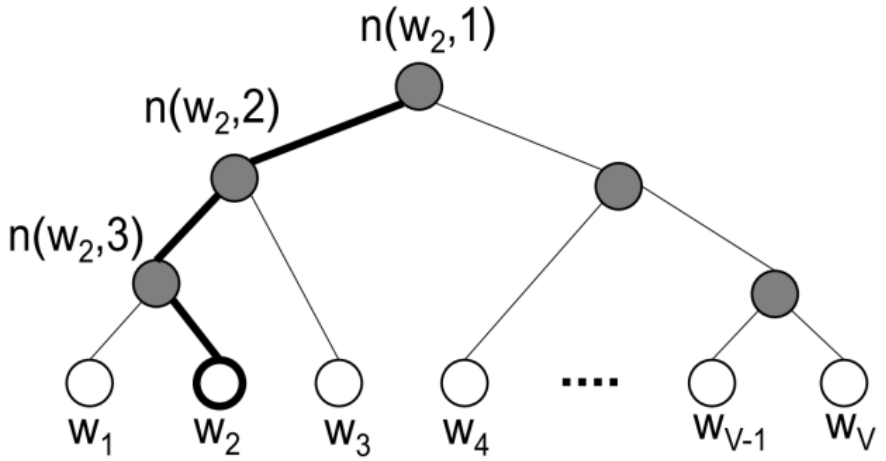
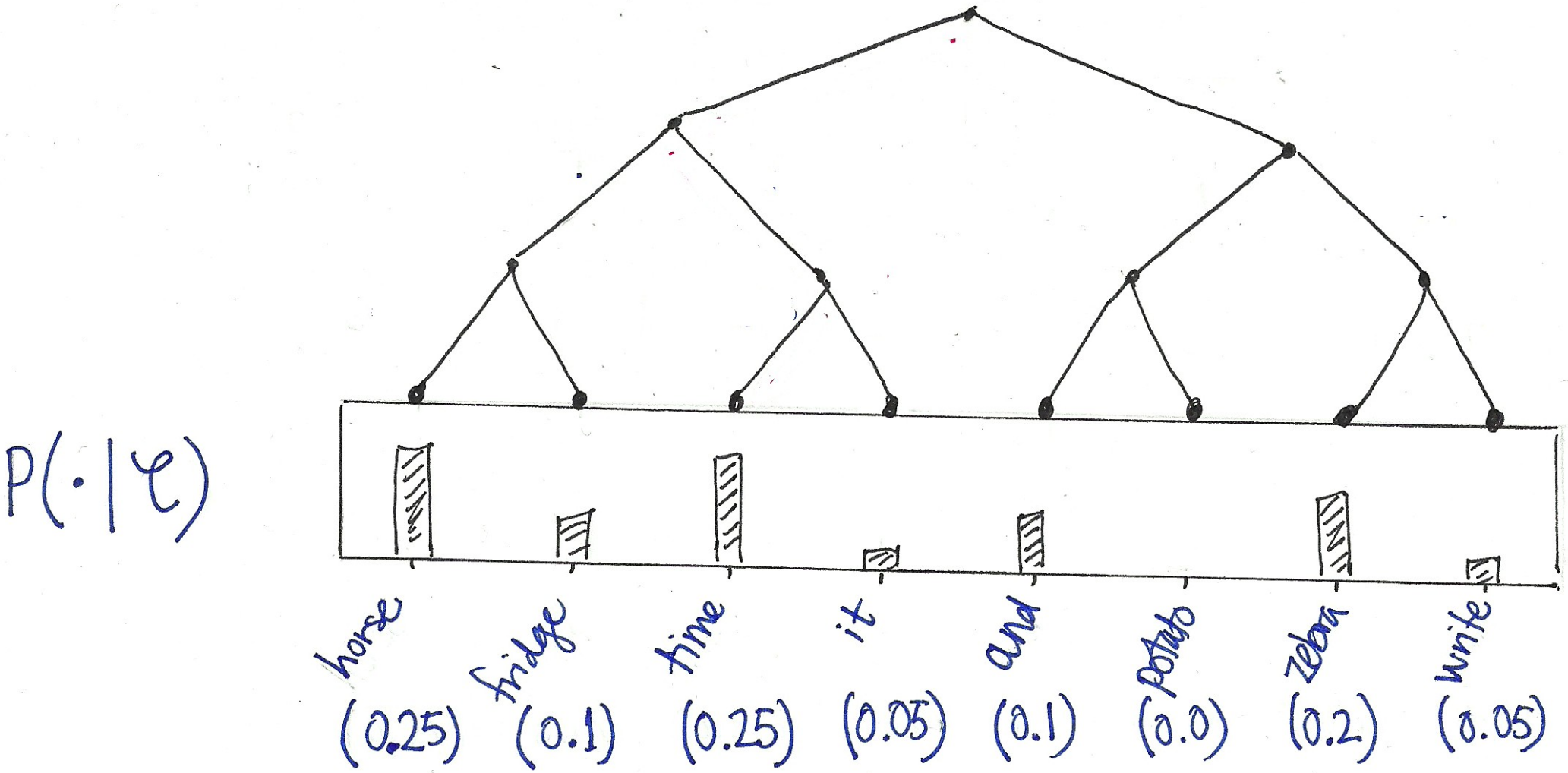


Figure 4: An example binary tree for the hierarchical softmax model. The white units are words in the vocabulary, and the dark units are inner units. An example path from root to w_2 is highlighted. In the example shown, the length of the path $L(w_2) = 4$. $n(w, j)$ means the j-th unit on the path from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

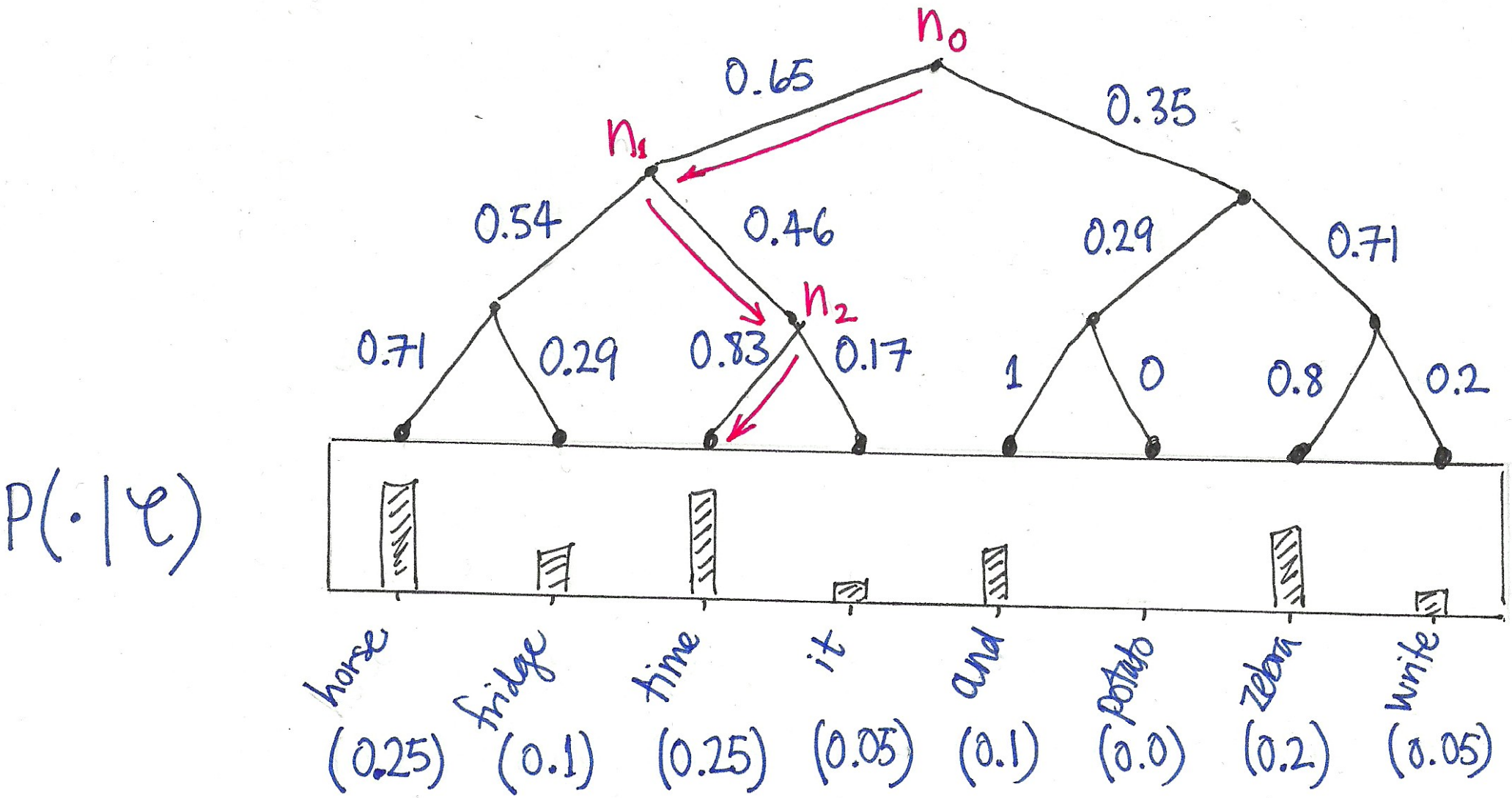
Source: https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/

Hierarchical softmax



Source: http://building-babylon.net/2017/08/01/hierarchical-softmax/

Hierarchical softmax



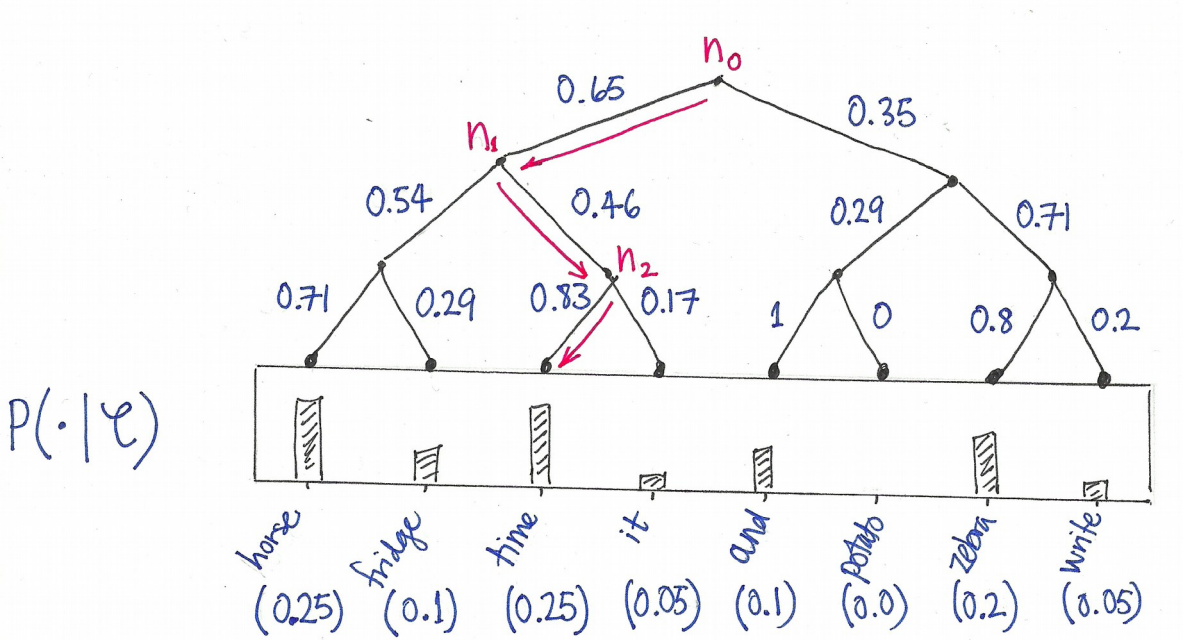
Source: http://building-babylon.net/2017/08/01/hierarchical-softmax/

Hierarchical softmax

$$P(\texttt{"time"}|C) = P_{n_0}(\text{right}|C)P_{n_1}(\text{left}|C)P_{n_2}(\text{righ}$$

$$P_n(\text{right}|C) = 1 - P_n(\text{left}|C)$$

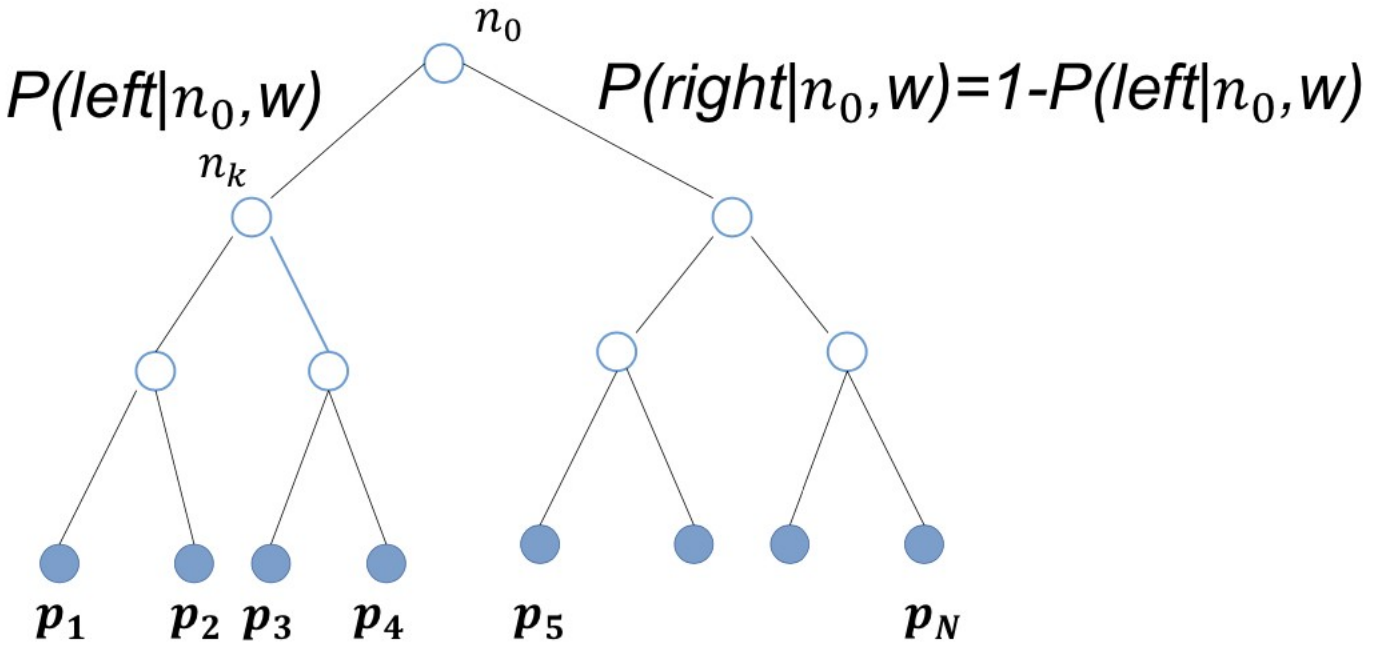
$$P_n(\text{left}|C) = \sigma({\gamma_n}^T \alpha_C)$$



Source: http://building-babylon.net/2017/08/01/hierarchical-softmax/

Hierarchical softmax

- Idea: represent probability distribution as a tree, where leaves are classes (words in our case).
- $p1, \dots, p_N$- leaves probabilities
- Mark each edge with probability of choosing this edge, moving down the tree



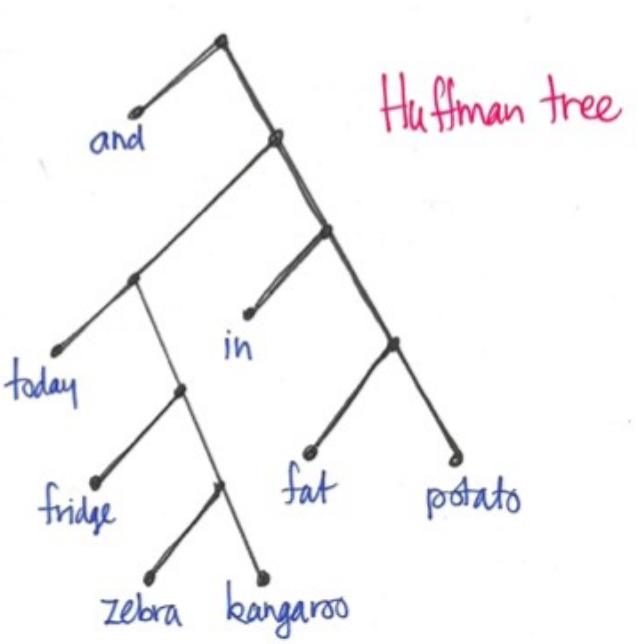
If $E_1, \dots, E_k$ - path to leaf *i*, then $p_i = P_{E_1} * \cdots * P_{E_k}$

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

Hierarchical softmax

- **Huffman tree**: minimizes the expected path length from root to leaf
- => minimizing the exp. number of updates

word	count
fat	3
fridge	2
zebra	1
potato	3
and	14
in	7
today	4
kangaroo	2



Source: http://building-babylon.net/2017/08/01/hierarchical-softmax/

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: Asssume 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)

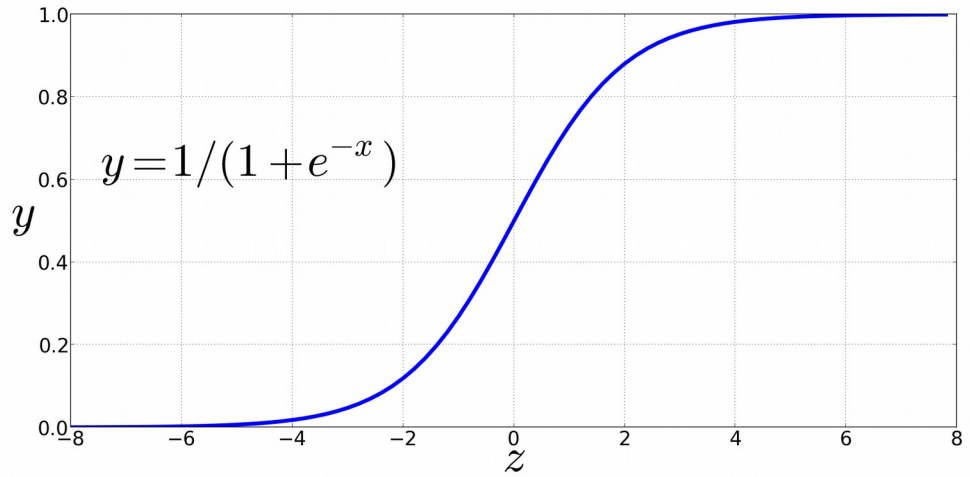
Return probability that c is a real context word:

P(+|t,c)

P(−|t,c) = 1−P(+|t,c)

How to compute p(+|t,c)?

- Intuition:
  - Words are likely to appear near similar words
  - Model similarity with dot-product!
  - Similarity(t,c) $\propto t \cdot c$
  - Turning dot product into a probability



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

Computing probabilities

Turning dot product into a probability:

$$P(+|t,c) = \frac{1}{1+e^{-t\cdot c}}$$

$$\begin{aligned} P(-|t,c) &= 1 - P(+|t,c) \\ &= \frac{e^{-t\cdot c}}{1+e^{-t\cdot c}} \end{aligned}$$

Assume all context words are independent:

$$P(+|t,c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1+e^{-t\cdot c_i}}$$

$$\log P(+|t,c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t\cdot c_i}}$$

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:

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

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

Objective function

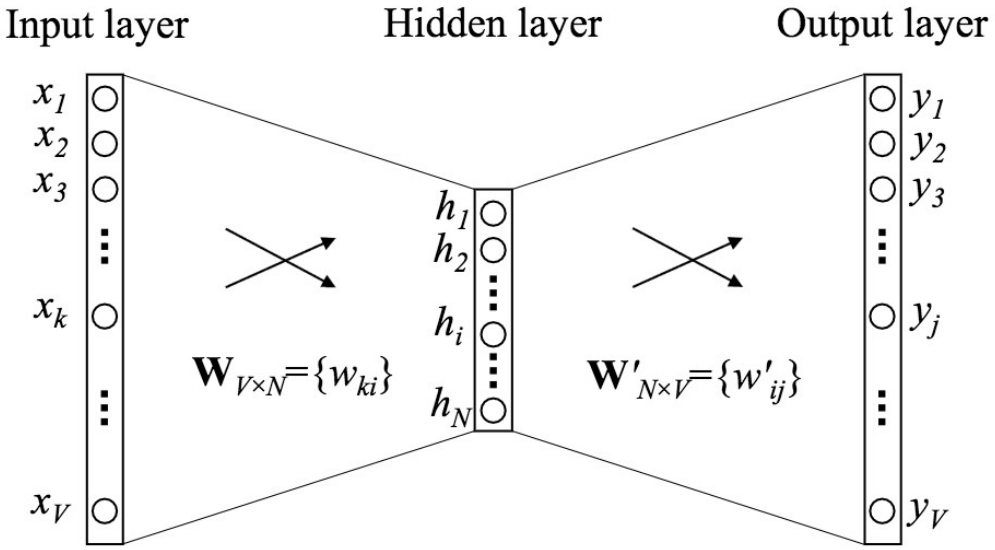
- We want to maximize…

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

- Maximize the + label for the pairs from the positive training data, and the – label for the negative samples.

$$\begin{aligned} L(\theta) &= \log P(+|t,c) + \sum_{i=1} \log P(-|t,n_i) \\ &= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\ &= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}} \end{aligned}$$

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

Training word2vec model: summary

- Start with V random 300-dimensional vectors as initial embeddings
- Use logistic regression, the second most basic classifier used in machine learning after naïve bayes
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings.

What does the model learns

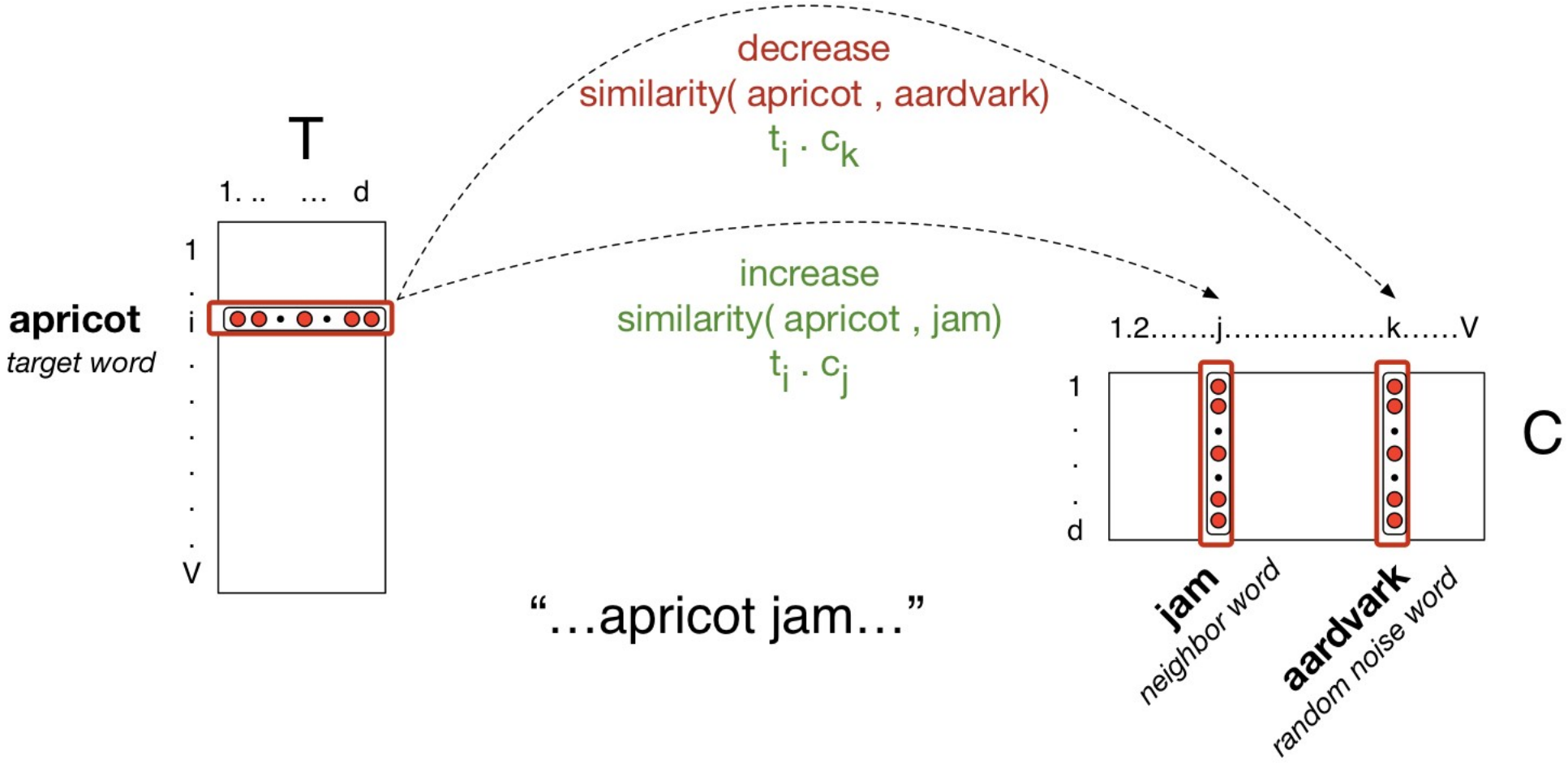
- The model tries to increase the scalar product of good (word, context) pairs and decrease for the bad ones.
- How to increase the scalar product of two vectors?
- increase lengths of one of the vectors: in that case, all scalar products of this vector are increasing

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

  - 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).



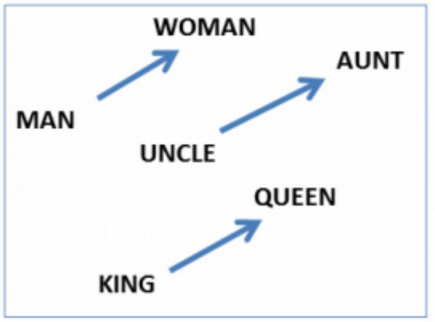
Source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

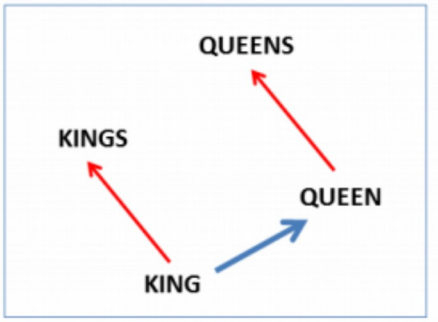
Vector Algebra for Analogy Questions

- 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

How about larger units than a word?

- Larger linguistic units:
  - Multi-word expression, noun phrase, ...
  - Sentence
  - Paragraph
  - Document
  - … corpus?

- Representing them in a low-dimensional fixed-length format is useful for feeding them into a neural network
  - Text categorization, sentiment analysis, gender detection, …
  - Clustering, analogies, arithmetics, … representing in a single space is useful

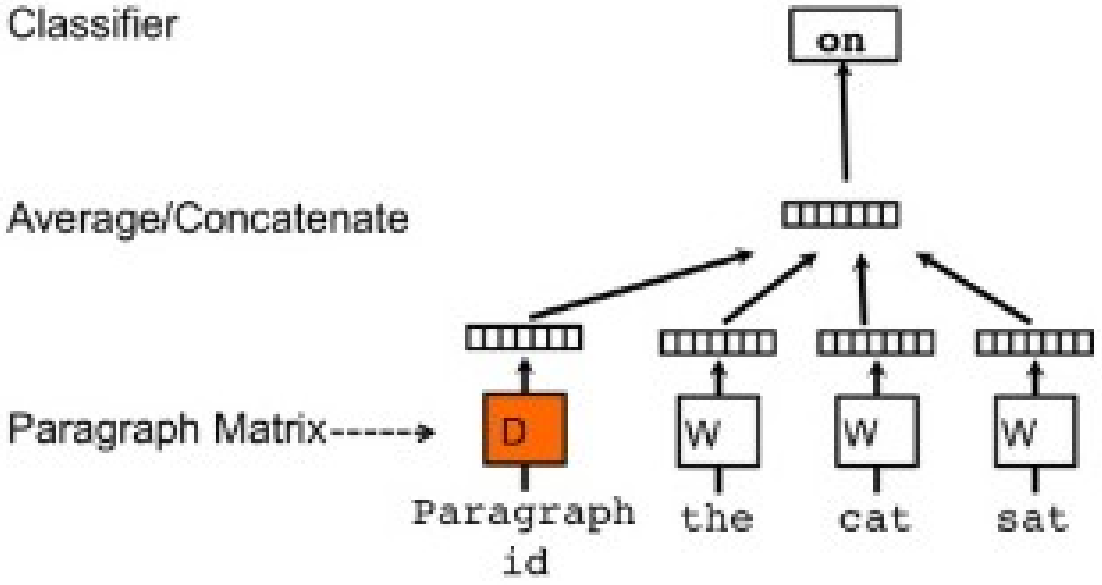
Source: .

Pooling / averaging of word vectors

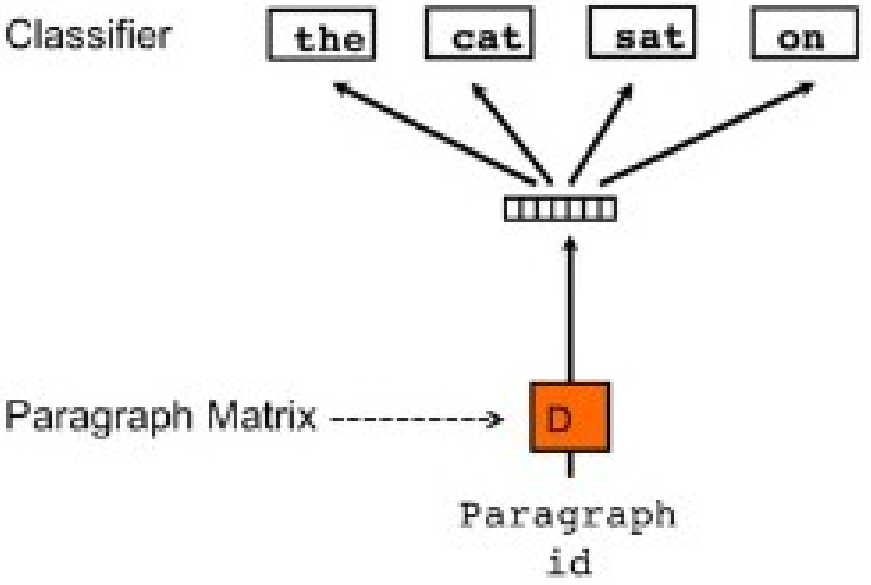
- The straightforward approach of averaging each of a text's words' word-vectors creates a quick and crude document-vector
  - often be useful
  - can be improved if weights, like TF-IDF are used and stopwords are removed
  - many models exist which outperform this baseline

Image source: https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/pooling_avg.html

Doc2vec model



Paragraph vector with distributed memory (PV-DM)

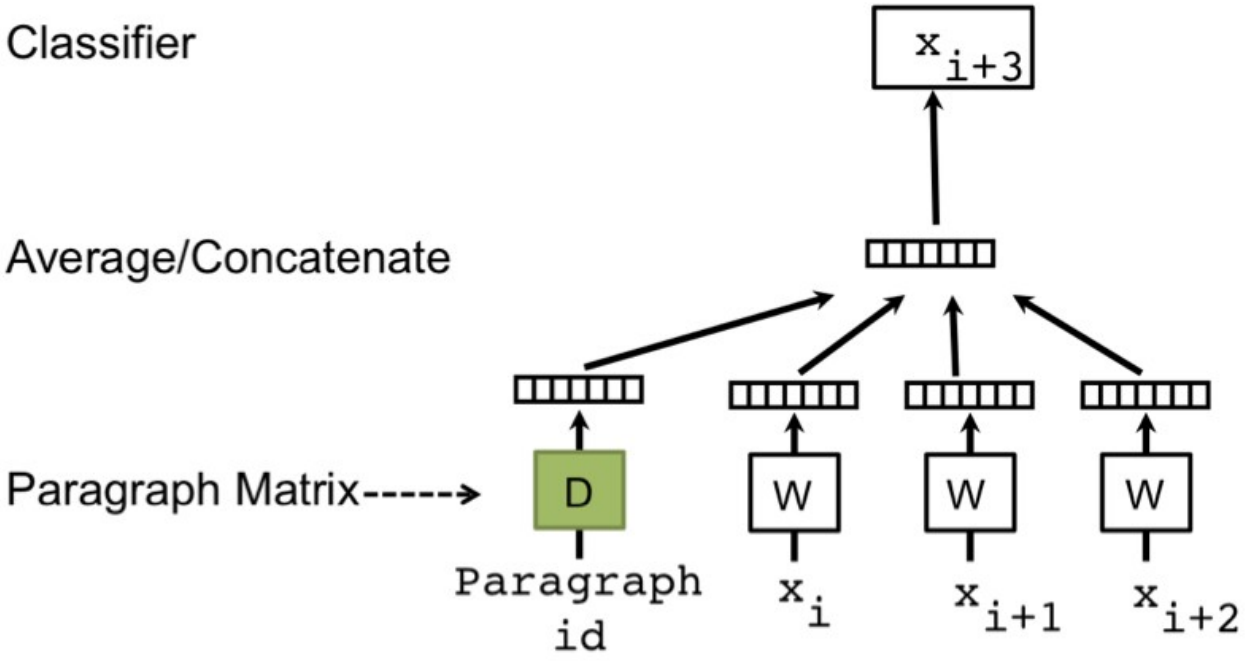


Distributed bag of words version (PVDBOW)

Source: https://arxiv.org/pdf/1507.07998.pdf

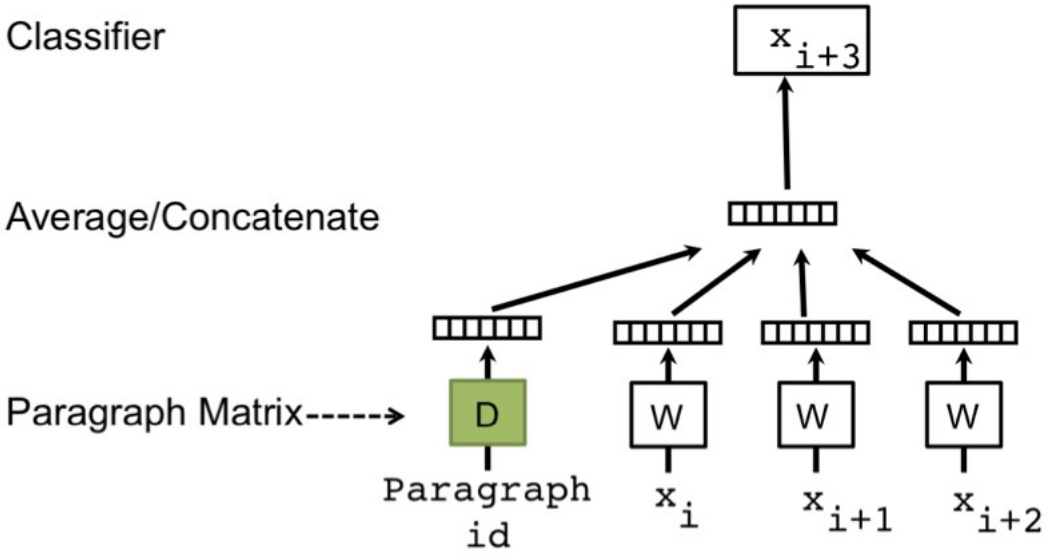
Doc2vec: Paragraph Vector - Distributed Memory (PV-DM)

- word2Vec **CBOW**
- Vectors are obtained by training a neural network on the task of predicting a center word based an **average of context word-vectors and the document vector**.



Doc2vec: Paragraph Vector - Distributed Memory (PV-DM)

- Paragraph vector is concatenated or averaged with local context word vectors to predict the next word.
- The prediction task changes the word vectors and the paragraph vector.



$$\frac{1}{M}\sum_{i=1}^{M}\frac{1}{|D_i|}\sum_{t=k}^{|D_{i-1}|-k} log(p(w_t^i|w_{t-k}^i,\ldots,w_{t+k}^i,D_i))$$

$D \in \mathbb{R}^{M\times R}$ – document matrix

$W \in \mathbb{R}^{N\times P}$ – word matrix (word2vec)

Getting a vector for an unseen document during training

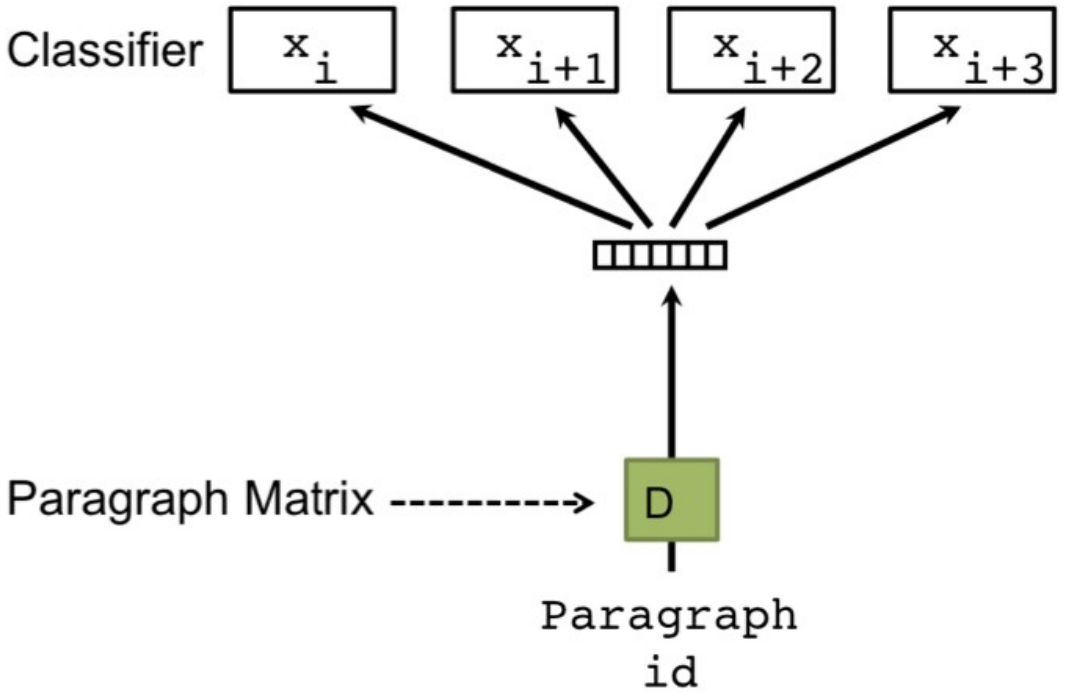
- Step 1: Fix **W** so that it is not updated
- Step 2: Augument **D** with the new randomly initialized row
- Step 3: Train for several iterations with the new row holding the embeddings for the inferred vector

- Note: This **will not give exactly the same vector** for a sentence from a training data!

Source: https://datascience.stackexchange.com/questions/10612/doc2vecgensim-how-can-i-infer-unseen-sentences-label

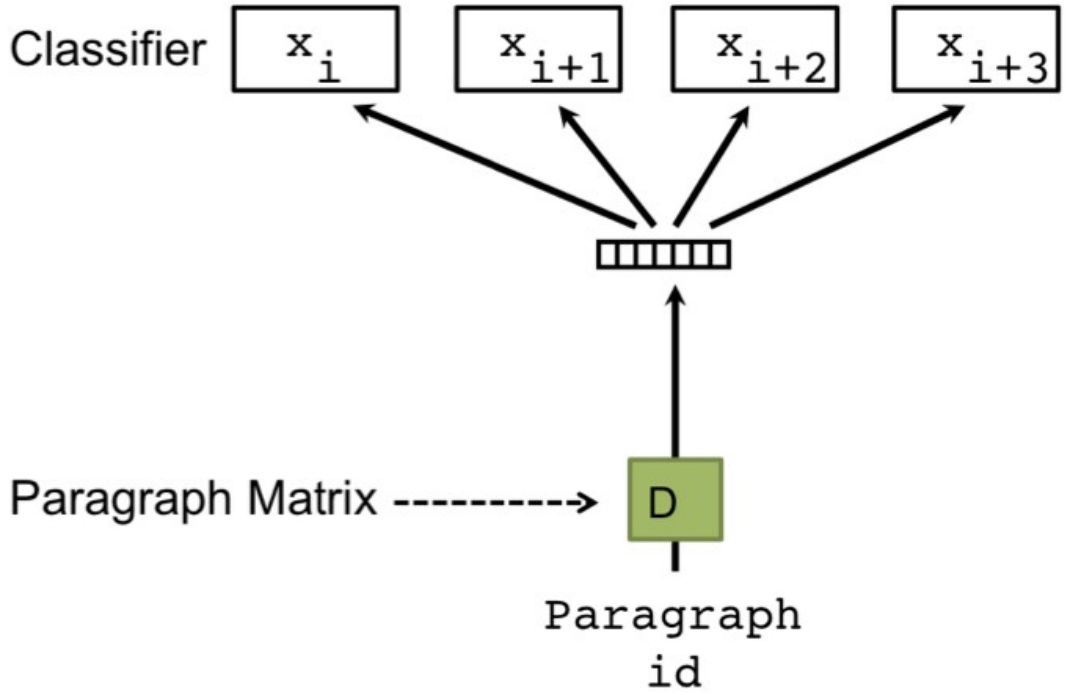
Doc2vec: Paragraph Vector - Distributed Bag of Words (PV-DBOW)

- Word2vec SkipGram model
- Vectors are obtained by training a neural network on the task of predicting a target word just from the full document's doc-vector.

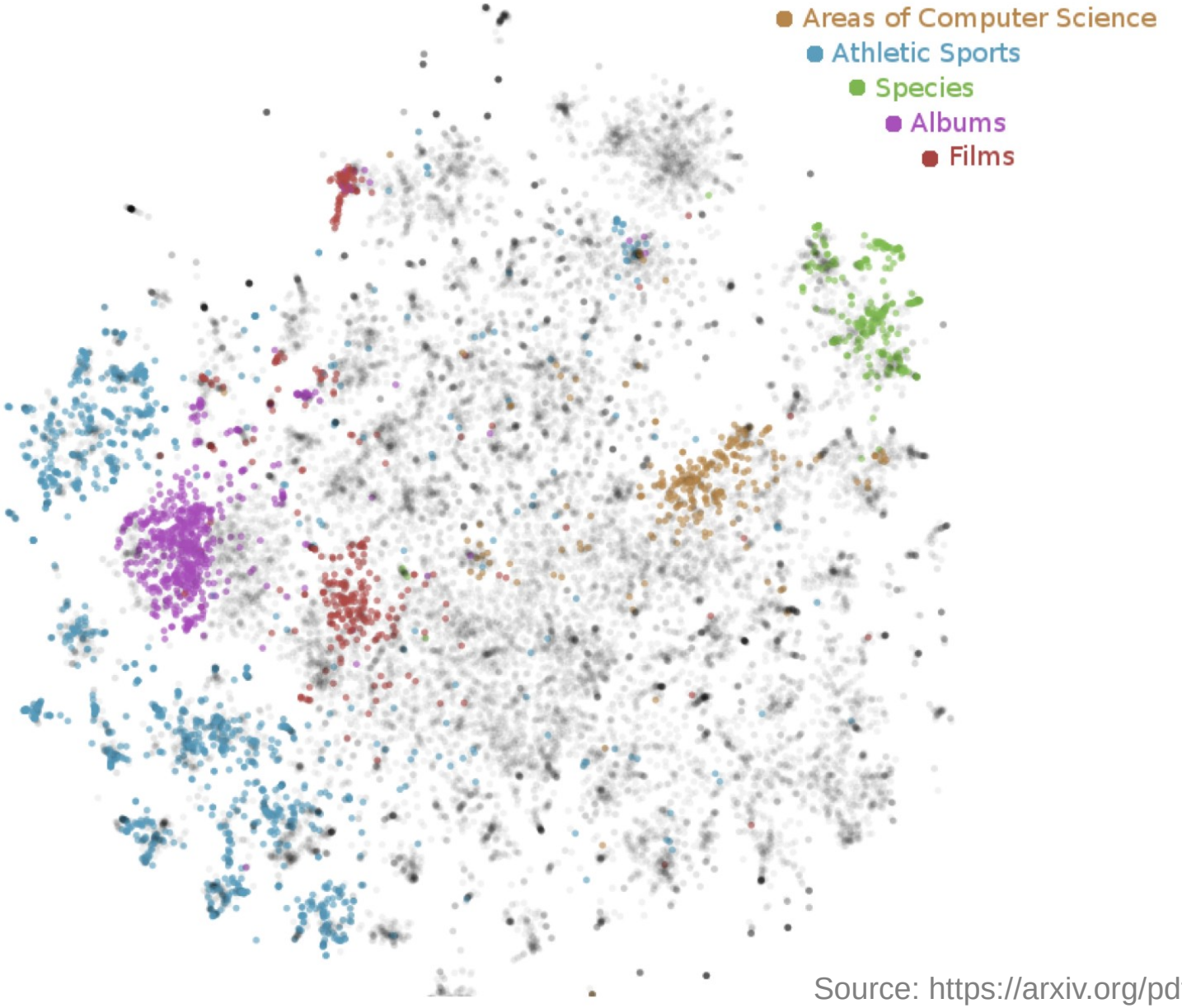


Doc2vec: Paragraph Vector - Distributed Bag of Words (PV-DBOW)

- No local context in the prediction task.
- At inference time, the parameters of the classifier and the word vectors are not needed
- backpropagation is used to tune the paragraph vectors



Visualization of Wikipedia paragraph vectors using t-SNE



Source: https://arxiv.org/pdf/1507.07998.pdf

Nearest neighbors Wikipedia articles to “Machine learning” article

LDA	Paragraph Vectors
Artificial neural network	Artificial neural network
Predictive analytics	Types of artificial neural networks
Structured prediction	Unsupervised learning
Mathematical geophysics	Feature learning
Supervised learning	Predictive analytics
Constrained conditional model	Pattern recognition
Sensitivity analysis	Statistical classification
SXML	Structured prediction
Feature scaling	Training set
Boosting (machine learning)	Meta learning (computer science)
Prior probability	Kernel method
Curse of dimensionality	Supervised learning
Scientific evidence	Generalization error
Online machine learning	Overfitting
N-gram	Multi-task learning
Cluster analysis	Generative model
Dimensionality reduction	Computational learning theory
Functional decomposition	Inductive bias
Bayesian network	Semi-supervised learning

Source: https://arxiv.org/pdf/1507.07998.pdf

Wikipedia nearest neighbours to “Lady Gaga” (Paragraph Vectors)

Article	Cosine Similarity
Christina Aguilera	0.674
Beyonce	0.645
Madonna (entertainer)	0.643
Artpop	0.640
Britney Spears	0.640
Cyndi Lauper	0.632
Rihanna	0.631
Pink (singer)	0.628
Born This Way	0.627
The Monster Ball Tour	0.620

Source: https://arxiv.org/pdf/1507.07998.pdf

Wikipedia nearest neighbours to "Lady Gaga" - "American" + "Japanese"

Article	Cosine Similarity
Ayumi Hamasaki	0.539
Shoko Nakagawa	0.531
Izumi Sakai	0.512
Urbangarde	0.505
Ringo Sheena	0.503
Toshiaki Kasuga	0.492
Chihiro Onitsuka	0.487
Namie Amuro	0.485
Yakuza (video game)	0.485
Nozomi Sasaki (model)	0.485

Ayumi Hamasaki

Japanese singer

Source: https://arxiv.org/pdf/1507.07998.pdf

Nearest Neighbours to “Distributed Representations of Sentences and Documents” using Paragraph Vectors

Title	Cosine Similarity
Evaluating Neural Word Representations in Tensor-Based Compositional Settings	0.771
Polyglot: Distributed Word Representations for Multilingual NLP	0.764
Lexicon Infused Phrase Embeddings for Named Entity Resolution	0.757
A Convolutional Neural Network for Modelling Sentences	0.747
Distributed Representations of Words and Phrases and their Compositionality	0.740
Convolutional Neural Networks for Sentence Classification	0.735
SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation	0.735
Exploiting Similarities among Languages for Machine Translation	0.731
Efficient Estimation of Word Representations in Vector Space	0.727
Multilingual Distributed Representations without Word Alignment	0.721

Source: https://arxiv.org/pdf/1507.07998.pdf

Performance evaluation

- Performances of different methods at the best dimensionality on the arXiv article triplets

Model	Embedding dimensions/topics	Accuracy
Paragraph vectors	100	85.0%
LDA	100	85.0%
Averaged word embeddings	300	81.1%
Bag of words		80.4%

Source: https://arxiv.org/pdf/1507.07998.pdf

Plan of the lecture

- **Part 1**: Distributional semantics and vector spaces.
- **Part 2**: word2vec and doc2vec models.
- **Part 3**: Other models for word document embeddings.

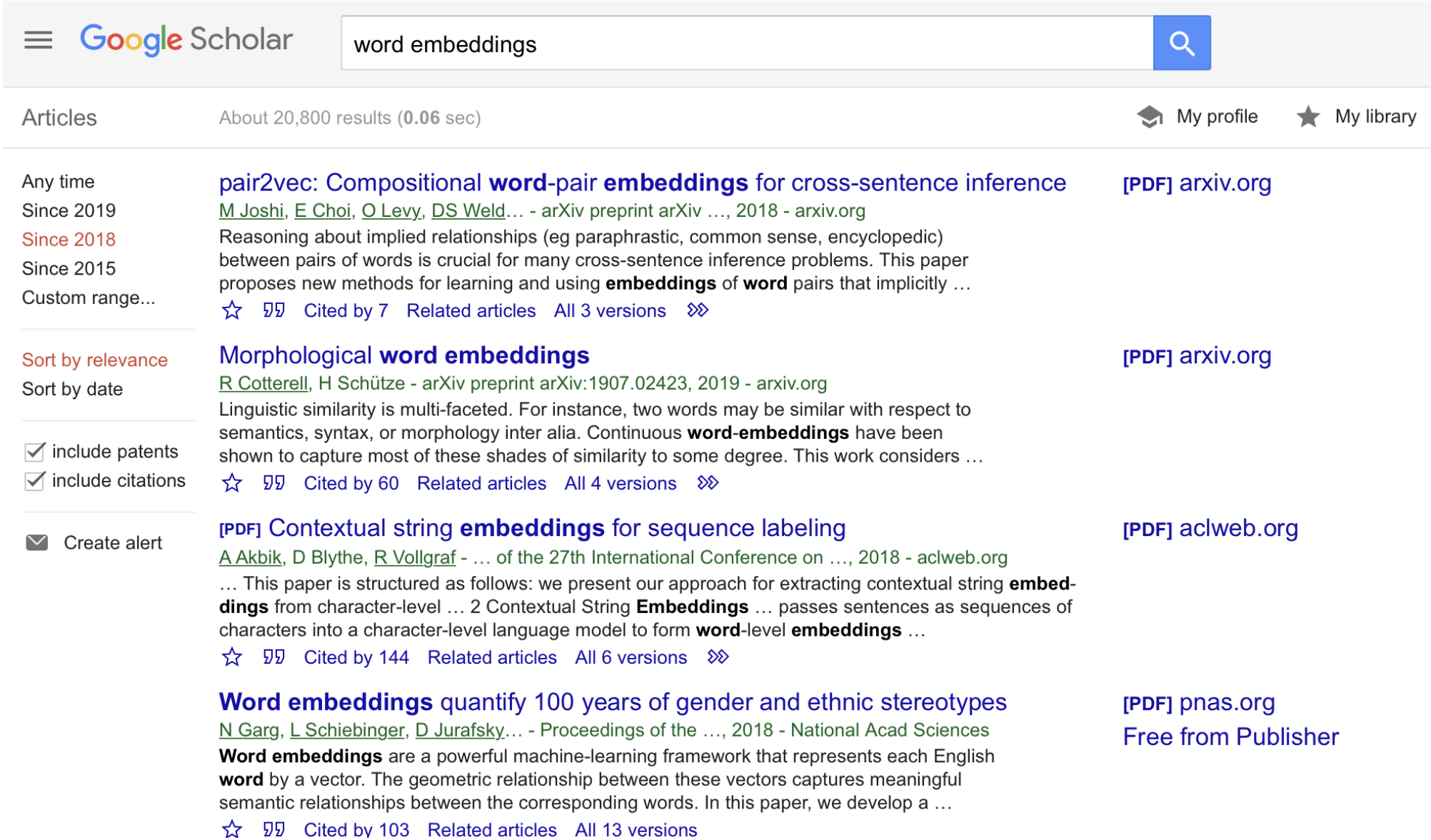
Some other popular dense word embedding methods

- **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

Tons of other models and applications of word embeddings



Global Vectors (GloVe)

- Objective function:

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

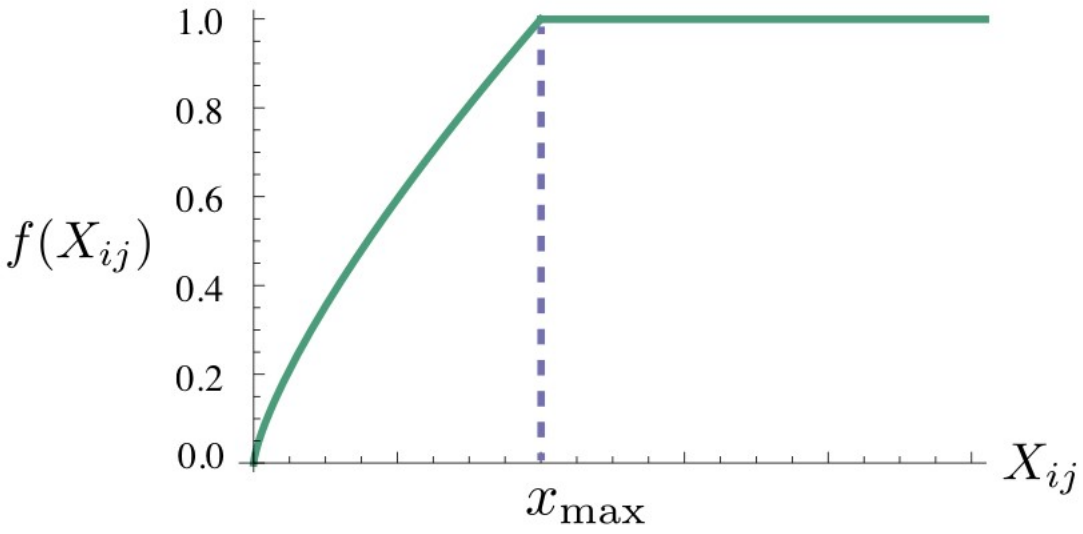
X_{ij} – matrix of word-word co-occurrence counts

$w \in \mathbb{R}^d$ – word vectors

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

- Weighting function:

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

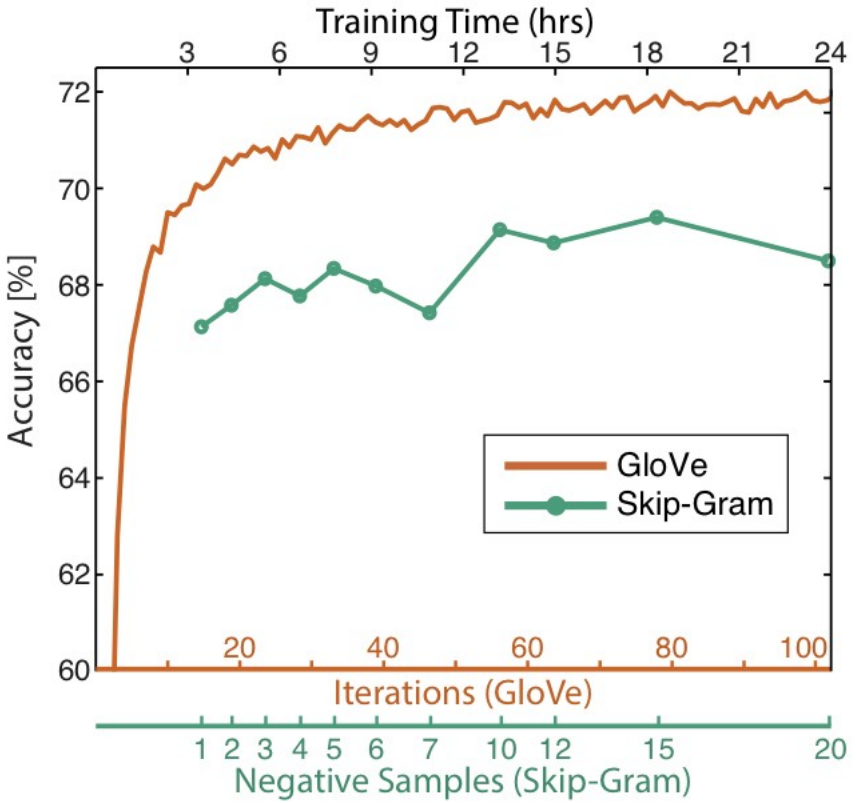


Source: Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)

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).

fastText

- From the developer of word2vec model
  - and is based on the SGNS model
- fastText is developed for text classification, but it can also be used to learn word embeddings.

  - For **text classification** read: https://arxiv.org/pdf/1607.01759.pdf

  - For **word embeddings** read: https://www.mitpressjournals.org/doi/pdfplus/10.1162/tacl_a_00051

fastText

- Uses character n-grams and word n-grams:
  - **morphological** information, not only context;
  - considering Subword units, and representing words by a **sum of its character n-grams**.
- The original SGNS loss:

$$\sum_{t=1}^{T}\left[\sum_{c\in\mathcal{C}_t}\ell(s(w_t,\,w_c))+\sum_{n\in\mathcal{N}_{t,c}}\ell(-s(w_t,\,n))\right]$$

s – scoring function: maps pairs of (word, context) to scores in *R*

$$s(w_t, w_c) = \mathbf{u}_{w_t}^{\top}\mathbf{v}_{w_c}$$

$\ell : x \mapsto \log(1 + e^{-x})$ – logistic loss

Source: https://www.mitpressjournals.org/doi/pdfplus/10.1162/tacl_a_00051

fastText

- By using a distinct vector representation for each word, the SGNS model ignores the internal structure of words.
- A different scoring function *s* which takes into account this information!
- Each word *w* is represented as a bag of character *n*-grams.
  - Add special boundary symbols < and > at the beginning and end of words.
  - Include the word *w* itself in the set of *n*-grams.

Source: https://www.mitpressjournals.org/doi/pdfplus/10.1162/tacl_a_00051

fastText

- Word “where” and *n* = 3:

<wh, whe, her, ere, re>

<where>.

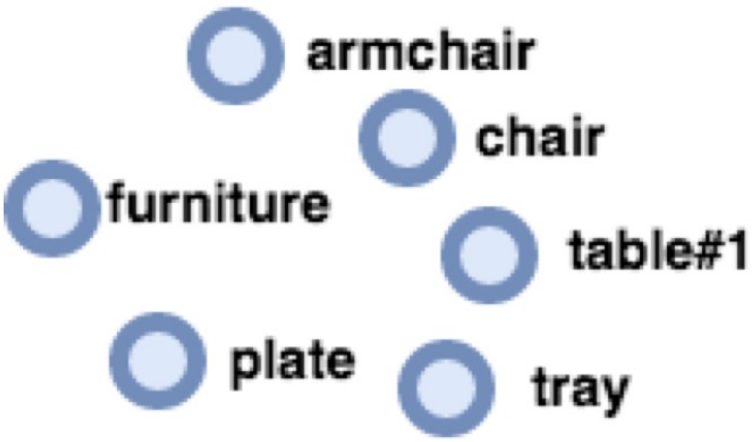
- Suppose a dictionary of *n*-grams of size *G*:
  - $\mathcal{G}_w \subset \{1, \ldots, G\}$ - set of n-grams appearing in *w*
  - Associate a vector $\mathbf{z}_g$ to each *n*-gram *g;*
  - The scoring function is:

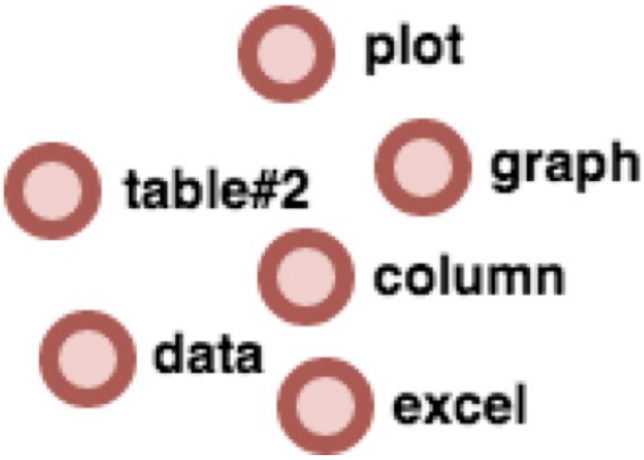
$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c$$

Source: https://www.mitpressjournals.org/doi/pdfplus/10.1162/tacl_a_00051

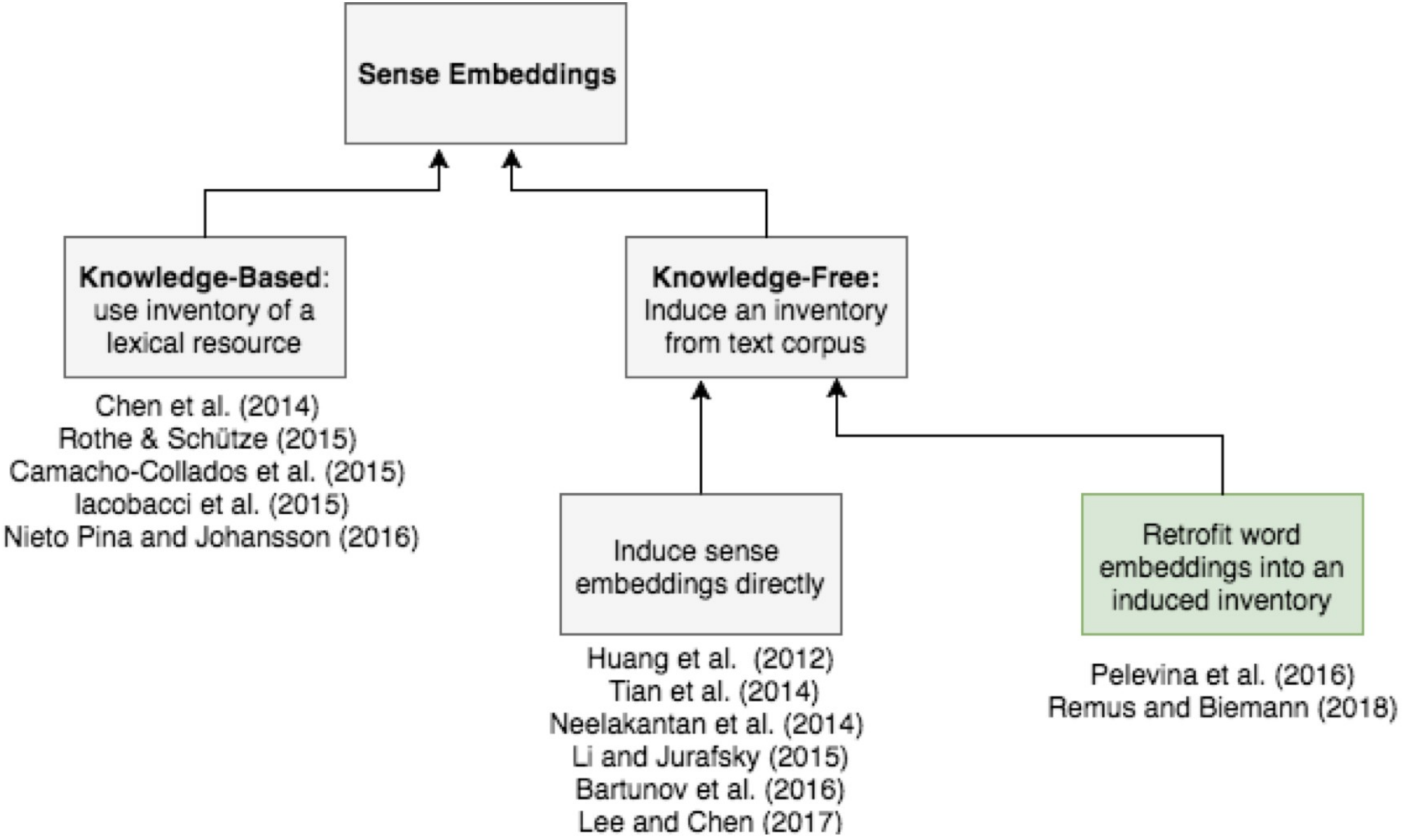
Word vs Sense Embeddings

Word vs Sense Embeddings





Sense embedding: various methods were proposed



Knowledge-based sense inventories: dictionaries, etc.

• bn:01713224n • NOUN • Named Entity • Categories: High-level programming languages, Dutch inventions, Class-based programming languages, Cros platform free software...

EN **Python (programming language)** · **/usr/bin/python** · **/usr/local/bin/python** · **Python language** · **Python programming language**

Python is a widely used general-purpose, high-level programming language. *Wikipedia*

More definitions

IS A	programming language • free software • scripting language
HAS PART	pandas
HAS KIND	Stackless Python
DESIGNER	Guido van Rossum
DEVELOPER	Python Software Foundation • Guido van Rossum
DIALECTS	Cython • Stackless Python
INFLUENCED BY	ALGOL 68 • alphabet • ruby
LICENSE	Python Software Foundation License

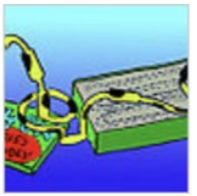
More relations

EXPLORE NETWORK

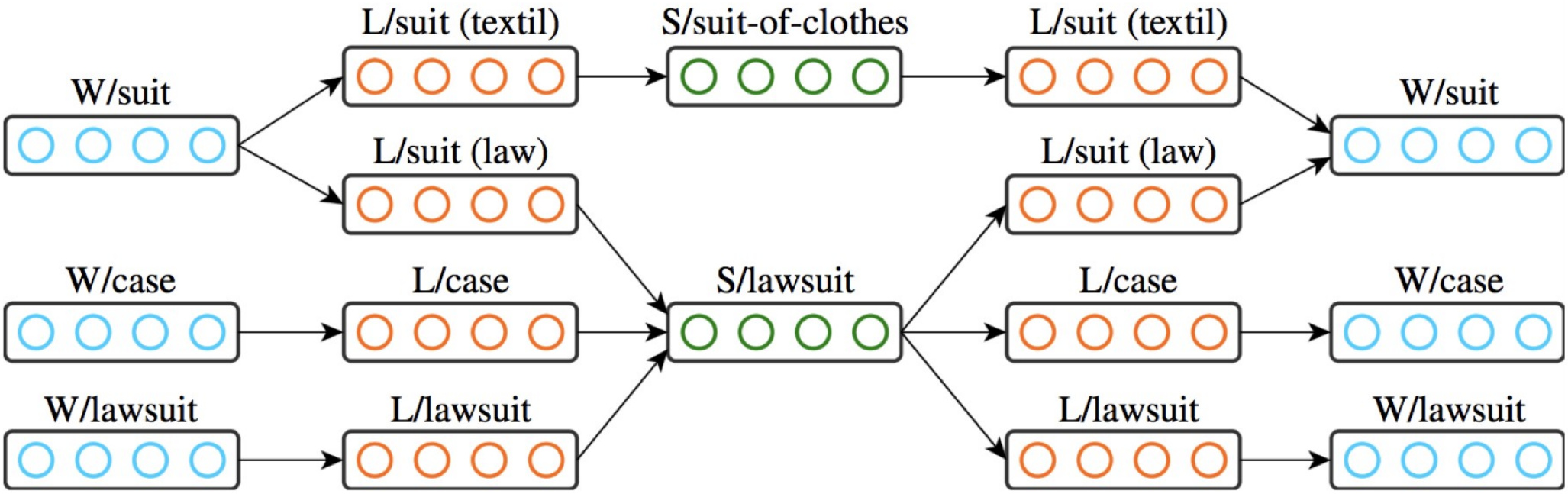








AutoExtend: a knowledge-based model using WordNet



Source: Rothe, S., & Schuetze, H. (2015). Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In EMNLP.

Multi-Sense Skip-gram: Neelakantan et al. (2015) model

- Step 1: The vector representation of the context is the **average of its context words**' vectors.
- Step 2: For every word type, **maintain clusters of its contexts**.
- Step 3: The sense of a word token is predicted as the cluster that is **closest to its context representation**.
- Step 4: After predicting the sense of a word token, perform a **gradient update on the embedding of that sense**.

- Note: Sense discrimination and learning embeddings are performed **jointly.**

Source: https://arxiv.org/pdf/1504.06654.pdf

Multi-Sense Skip-gram: Neelakantan et al. (2015) model

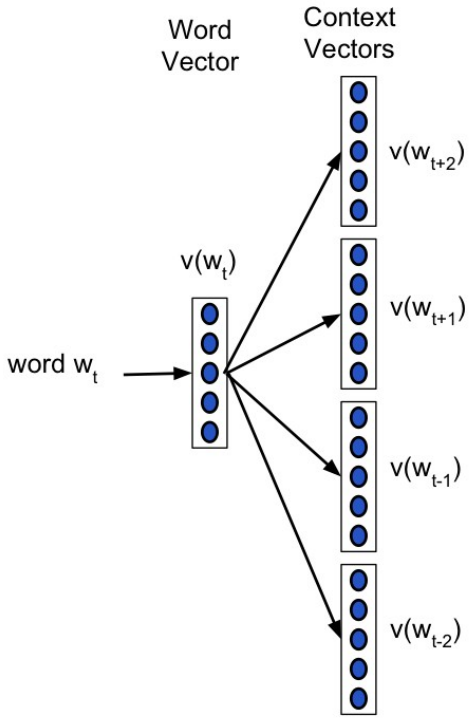


Figure 1: Architecture of the Skip-gram model with window size $R_t = 2$. Context c_t of word w_t consists of $w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2}$.

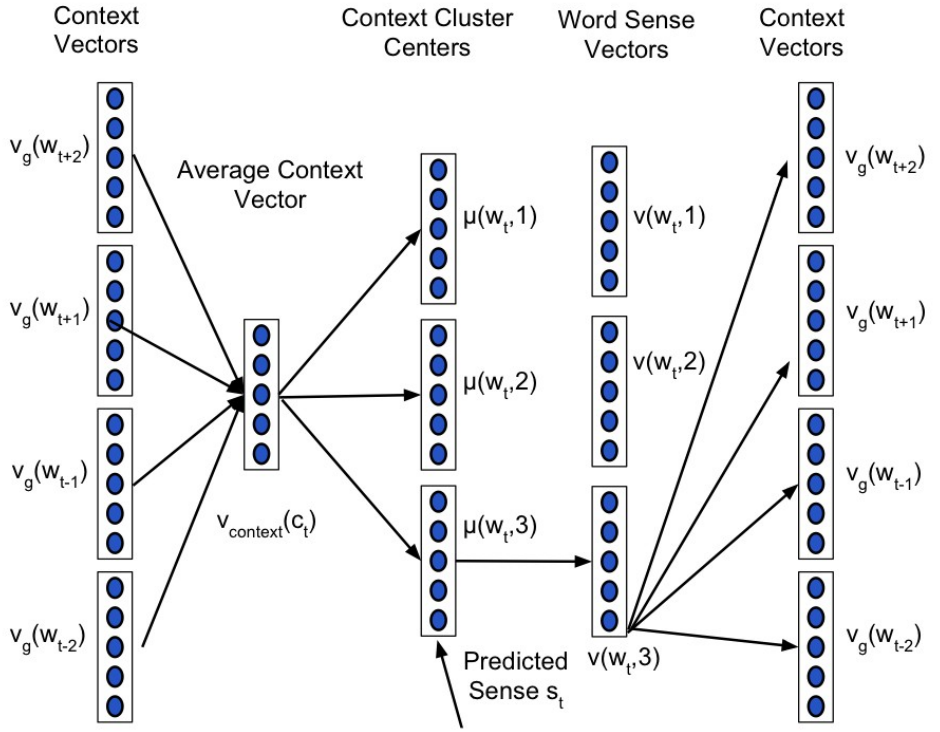


Figure 2: Architecture of Multi-Sense Skip-gram (MSSG) model with window size $R_t = 2$ and $K = 3$. Context c_t of word w_t consists of $w_{t-1}, w_{t-2}, w_{t+1}, w_{t+2}$. The sense is predicted by finding the cluster center of the context that is closest to the average of the context vectors.

Source: https://arxiv.org/pdf/1504.06654.pdf

Non-Parametric Multi-Sense Skip-gram: Neelakantan et al. (2015)

- Create a new cluster (sense) for a word type with probability proportional to the distance of its context to the nearest cluster (sense).
- The number of senses for a word is unknown and is learned during training.
- New context cluster and a sense vector are created online during training
  - when the word is observed with a context were the similarity between the vector representation of the context with every existing cluster center of the word is less than λ
  - λ is a hyperparameter

Source: https://arxiv.org/pdf/1504.06654.pdf

Non-Parametric Multi-Sense Skip-gram: Neelakantan et al. (2015)

- Nearest Neighbors of the word plant for different models:

Skip-gram	plants, flowering, weed, fungus, biomass
MS -SG	plants, tubers, soil, seed, biomass refinery, reactor, coal-fired, factory, smelter asteraceae, fabaceae, arecaceae, lamiaceae, ericaceae
NP MS -SG	plants, seeds, pollen, fungal, fungus factory, manufacturing, refinery, bottling, steel fabaceae, legume, asteraceae, apiaceae, flowering power, coal-fired, hydro-power, hydroelectric, refinery

Source: https://arxiv.org/pdf/1504.06654.pdf

Nearest neighbors of each sense of each word by cosine similarity

APPLE

Skip-gram	blackberry, macintosh, acorn, pear, plum
MSSG	pear, honey, pumpkin, potato, nut microsoft, activision, sony, retail, gamestop macintosh, pc, ibm, iigs, chipsets
NP-MSSG	apricot, blackberry, cabbage, blackberries, pear microsoft, ibm, wordperfect, amiga, trs-80

FOX

Skip-gram	abc, nbc, soapnet, espn, kttv
MSSG	beaver, wolf, moose, otter, swan nbc, espn, cbs, ctv, pbs dexter, myers, sawyer, kelly, griffith
NP-MSSG	rabbit, squirrel, wolf, badger, stoat cbs,abc, nbc, wnyw, abc-tv

NET

Skip-gram	profit, dividends, pegged, profits, nets
MSSG	snap, sideline, ball, game-trying, scoring negative, offset, constant, hence, potential pre-tax, billion, revenue, annualized, us$
NP-MSSG	negative, total, transfer, minimizes, loop pre-tax, taxable, per, billion, us$, income ball, yard, fouled, bounced, 50-yard wnet, tvontorio, cable, tv, tv-5

ROCK

Skip-gram	glam, indie, punk, band, pop
MSSG	rocks, basalt, boulders, sand, quartzite alternative, progressive, roll, indie, blues-rock rocks, pine, rocky, butte, deer
NP-MSSG	granite, basalt, outcropping, rocks, quartzite alternative, indie, pop/rock, rock/metal, blues-rock

RUN

Skip-gram	running, ran, runs, afoul, amok
MSSG	running, stretch, ran, pinch-hit, runs operated , running, runs, operate, managed running, runs, operate, drivers, configure
NP-MSSG	two-run, walk-off, runs, three-runs, starts operated, runs, serviced, links, walk running, operating, ran, go, configure re-election, reelection, re-elect, unseat, term-limited helmed, longest-running, mtv, promoted, produced

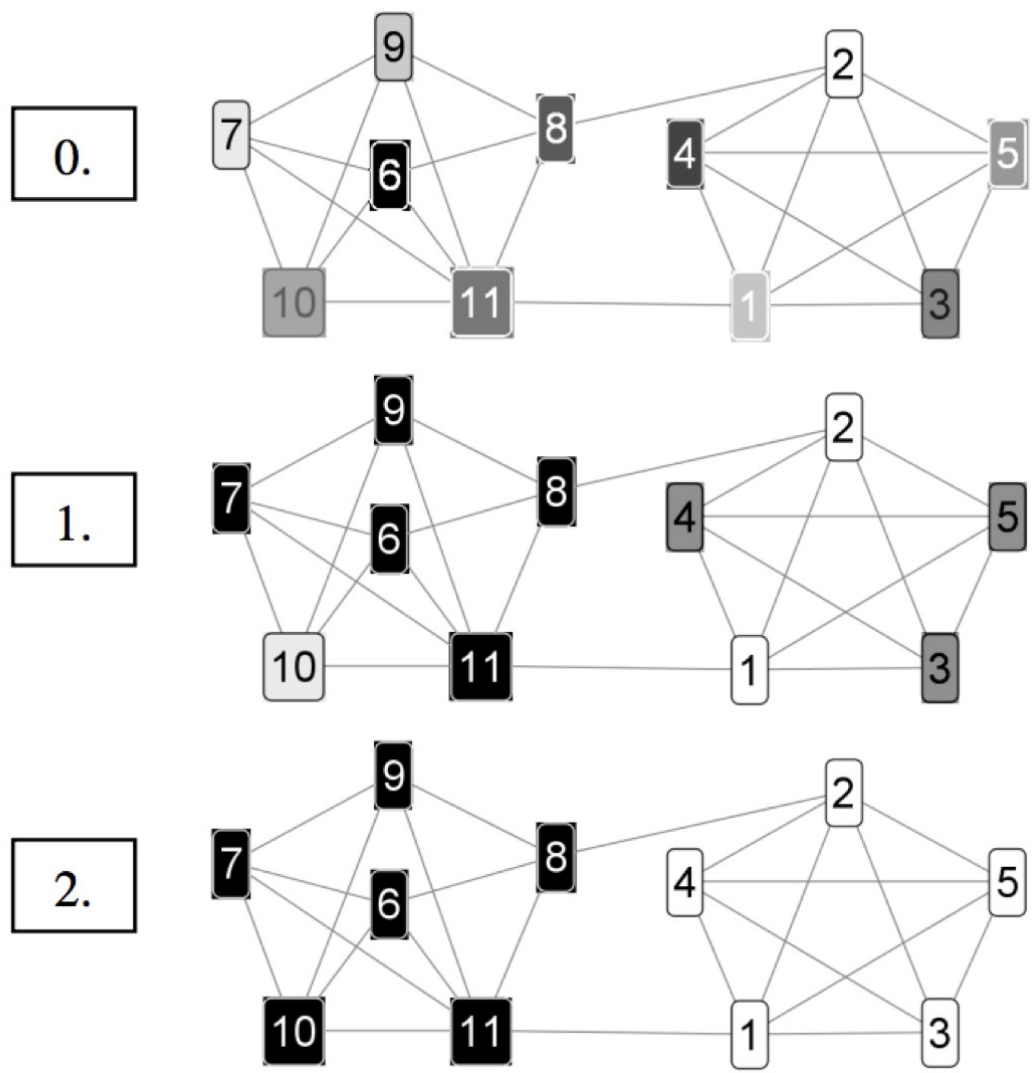
Source: https://arxiv.org/pdf/1504.06654.pdf

SenseGram: from pre-trained word embeddings to sense embeddings

- Graph clustering
  - Chinese Whispers
  - (Biemann, 2006)

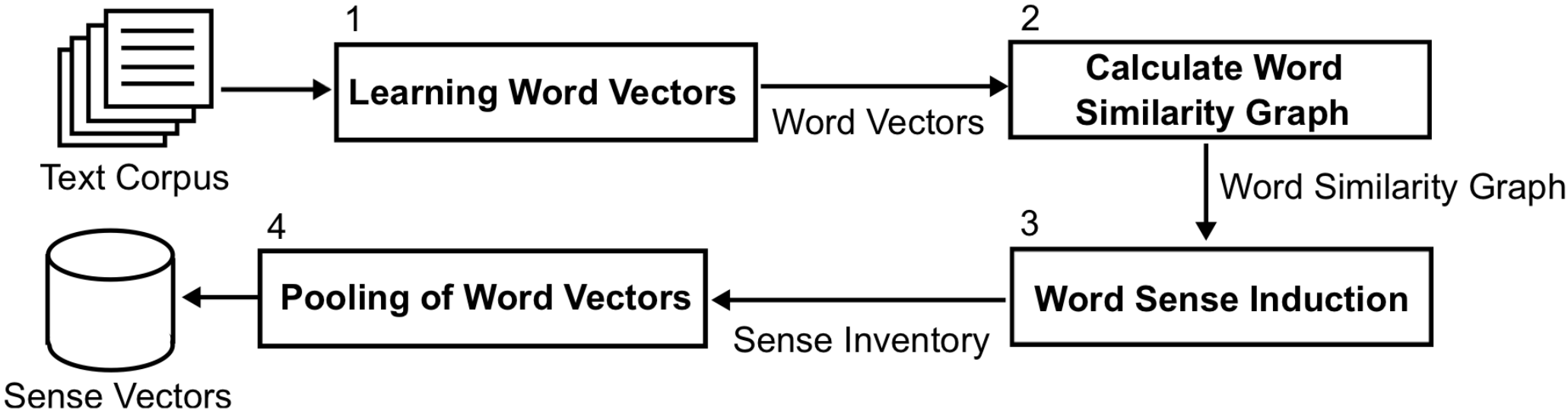
```
initialize:
 forall v_i in V: class(v_i)=i;

while changes:
 forall v in V, randomized order:
   class(v)=highest  ranked  class
            in neighborhood of v;
```



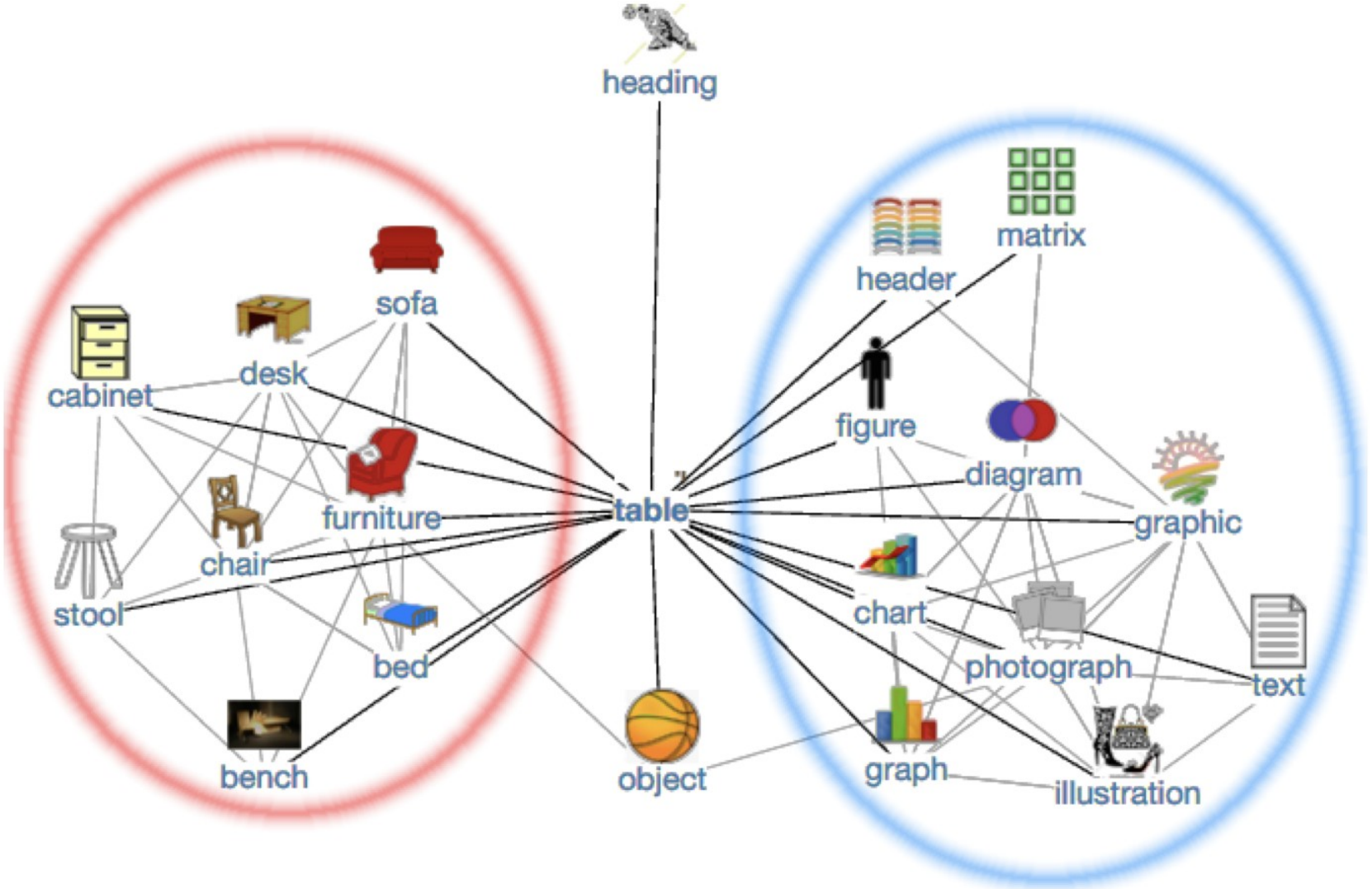
SenseGram: from pre-trained word embeddings to sense embeddings

- Sense embeddings using retrofitting:



SenseGram: from pre-trained word embeddings to sense embeddings

- Sense embeddings using retrofitting:

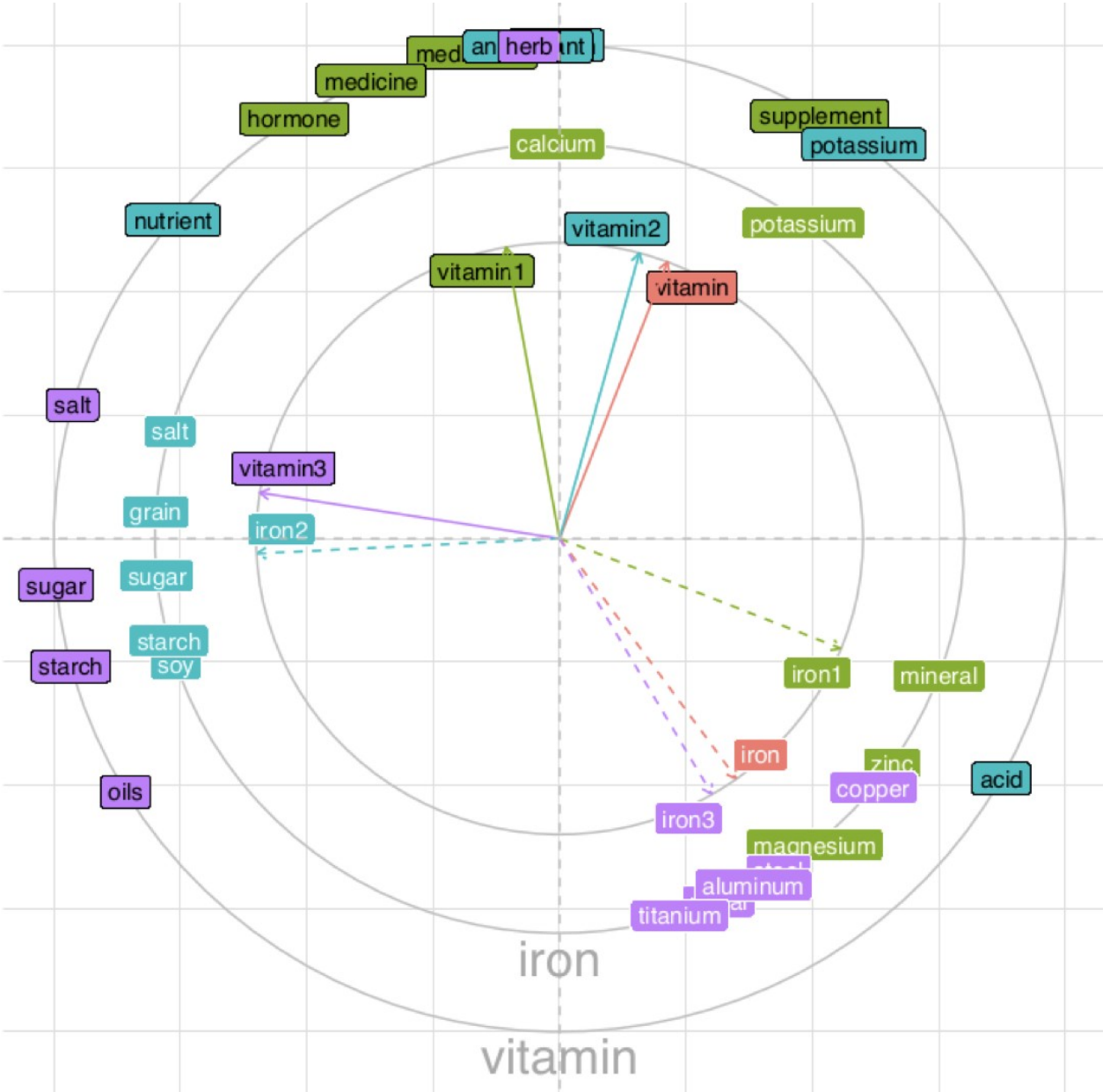


SenseGram: from pre-trained word embeddings to sense embeddings

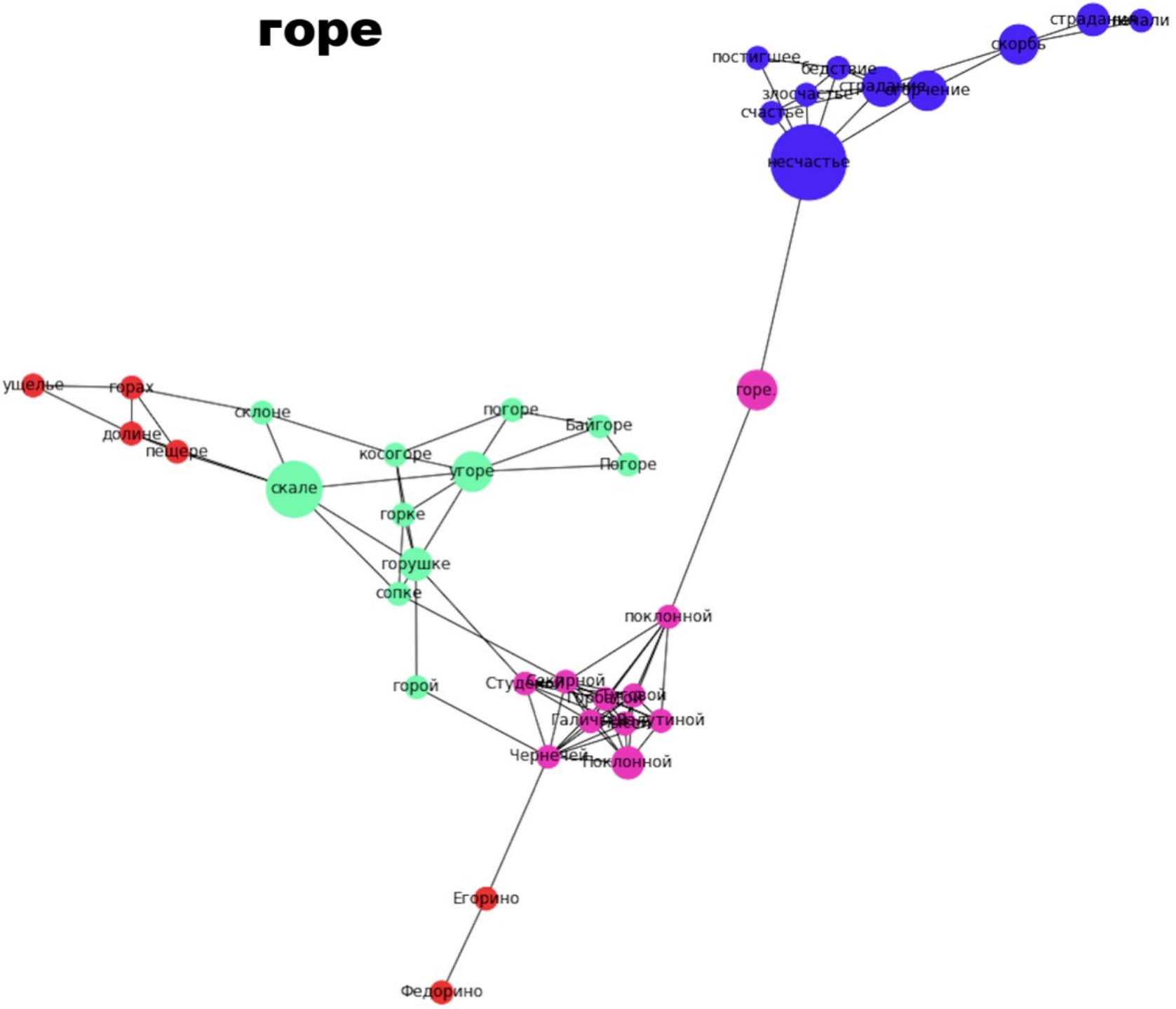
- Neighbors of word and sense vectors:

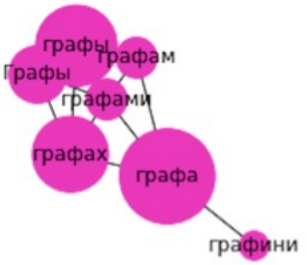
Vector	Nearest Neighbors
table	tray, bottom, diagram, bucket, brackets, stack, basket, list, parenthesis, cup, saucer, pile, playfield, bracket, pot, drop-down, cue, plate
table#0	leftmost#0, column#1, tableau#1, indent#1, bracket#3, pointer#0, footer#1, cursor#1, diagram#0, grid#0
table#1	pile#1, stool#1, tray#0, basket#0, bowl#1, bucket#0, box#0, cage#0, saucer#3, mirror#1, pan#1, lid#0

Word and sense embeddings of words iron and vitamin



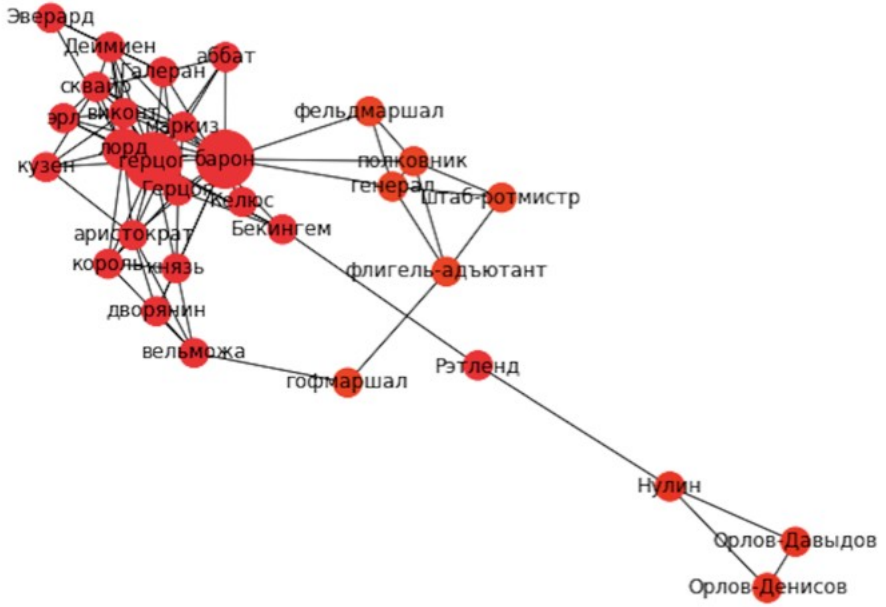
Source: LREC'18 (Remus & Biemann, 2018)

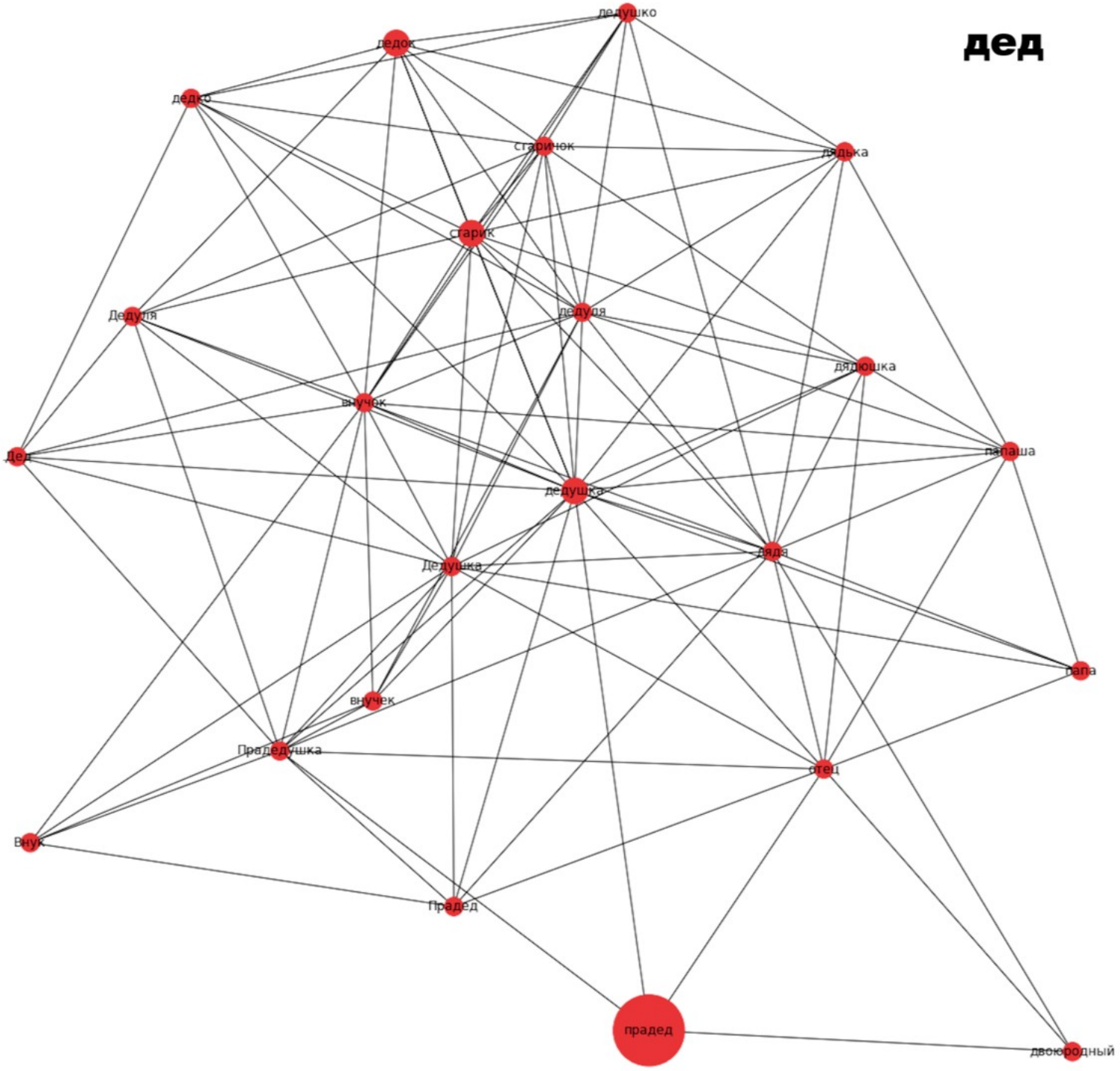

горе
страдания
скорбь
постигшее
бедствие
злосчастье
счастье
несчастье
горе.
ущелье
горах
склоне
погоре
Байгоре
долине
пещере
косогоре
угоре
Погоре
скале
горке
горушке
сопке
поклонной
горой
Студеный
Галичанин
Путиной
Чернечей
Поклонной
Егорино
Федорино


графы
Графы
графам
графами
графах
графа
графини

граф


подграф
орграф

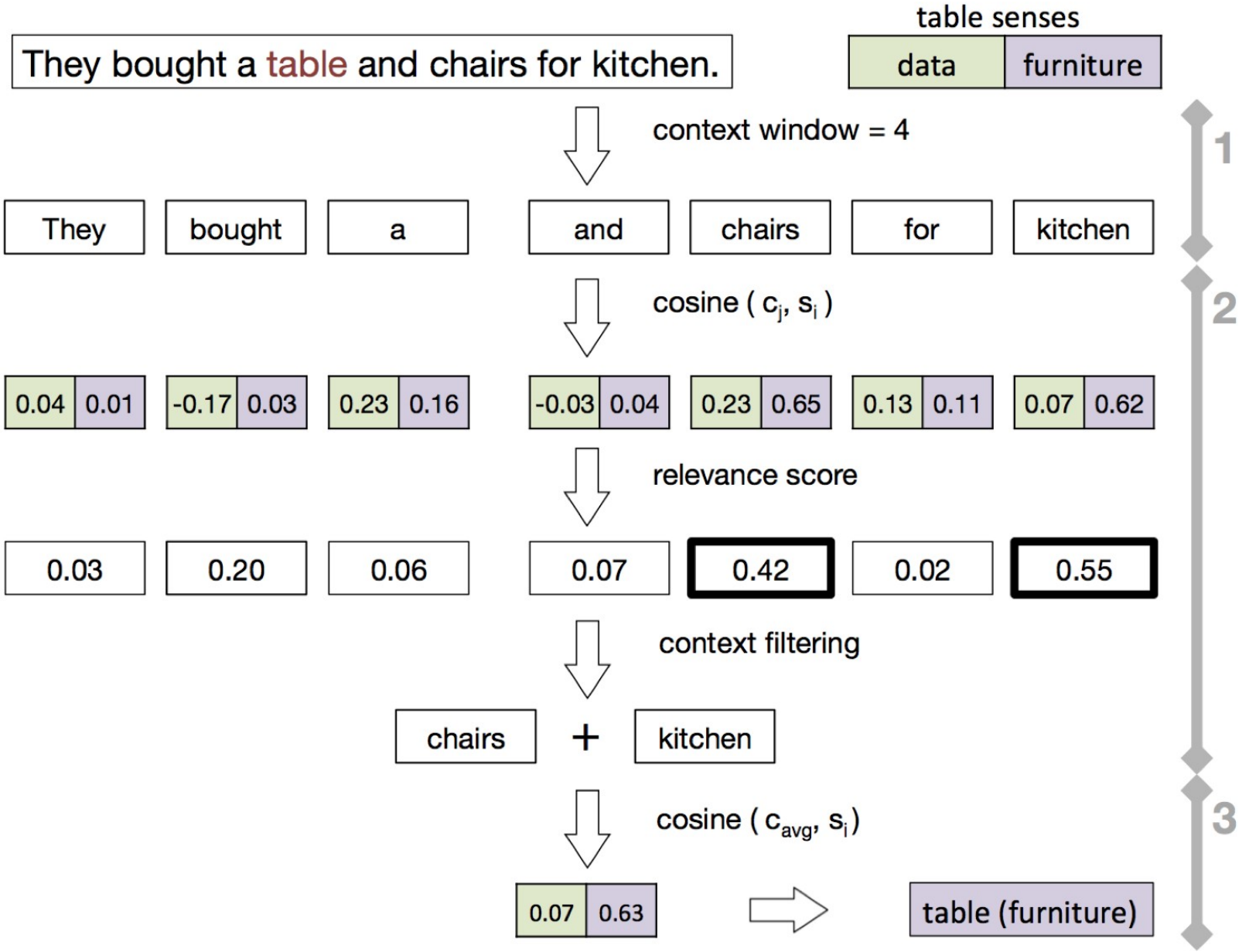

Эверард
Деймиен
аббат
Валеран
виконт
маркиз
лорд
герцог
барон
кузен
фельдмаршал
полковник
генерал
штаб-ротмистр
Герцог
Бекингем
аристократ
король
князь
флигель-адъютант
дворянин
вельможа
гофмаршал
Рэтленд
Нулин
Орлов-Давыдов
Орлов-Денисов


дед
дедушко
дедок
дедко
старичок
дядька
старик
Дедуля
дедуля
дядюшка
внучок
Дед
папаша
дедушка
дядя
Дедушка
папа
внучек
Прадедушка
отец
Внук
прадед
Прадед
двоюродный

SenseGram: word sense disambiguation

- Step 1: Context extraction – use context words around the target word
- Step 2: Context filtering – based on context word's relevance for disambiguation
- Step 3: Sense choice in context – maximise similarity between a context vector and a sense vector

SenseGram: word sense disambiguation



Application of sense representations: humor detection and generation?

Affine transformation of word embedding spaces

- Input: word vector (embedding)
- Output: word vector
  - In the same space (different transformation yield different properties. e.g. semantic and morphological relations)
  - In a different space, e.g. in a different language → machine translation

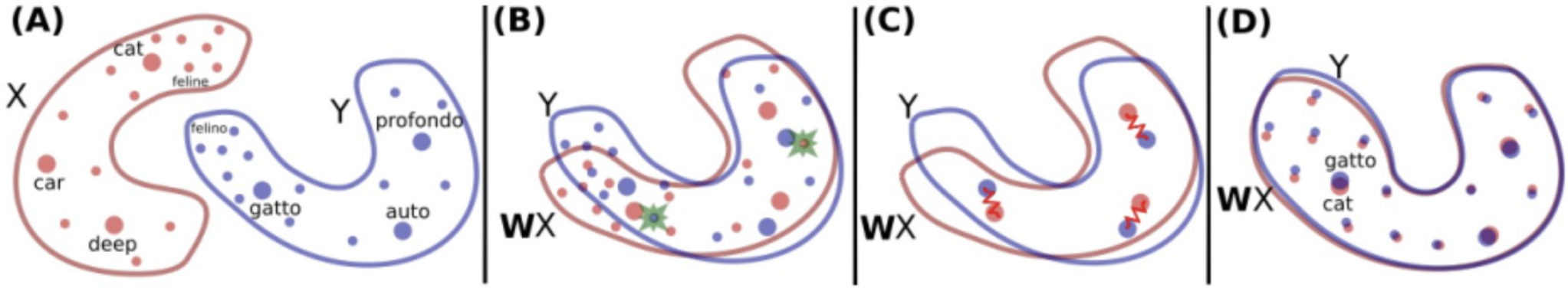
$$\vec{y} = A\vec{x} + \vec{b}.$$

- Reflection
- Rotation
- Scaling
- Translation

Cross-lingual embeddings

- $x_i \in \mathrm{R}^d$ – word embedding in the source language
- $y_i \in \mathrm{R}^d$ – word embedding in the target language
- Learn a linear transform for some subset of word embeddings (Procrustes problem):

$$W^* = \arg\min_W \sum_{i=1}^{n} ||Wx_i - y_i||_2$$



Cross-lingual embeddings

- Making it better (orthogonal Procrustean problem):

$$W^* = \arg\min_W ||WX - Y||_F, \text{ where: } W^T W = I$$

$$I\text{- identity matrix}$$

- Solution via SVD:

$$X^T Y = U\Sigma V^T, \text{ singular value decompostion}$$
$$W^* = UV^T$$

RU-UK cross-lingual mapping example

```
ru_emb.most_similar([np.matmul(uk_emb["серпень"], W)])
```

```
[('апрель', 0.8237907290458679),
 ('сентябрь', 0.8049713373184204),
 ('март', 0.802565336227417),
 ('июнь', 0.802184241287231),
 ('октябрь', 0.8001736402511597),
 ('ноябрь', 0.793448269367218),
 ('февраль', 0.7914119958877563),
 ('июль', 0.790810823440551 8),
 ('август', 0.789101779460907),
 ('декабрь', 0.7686372995376587)]
```

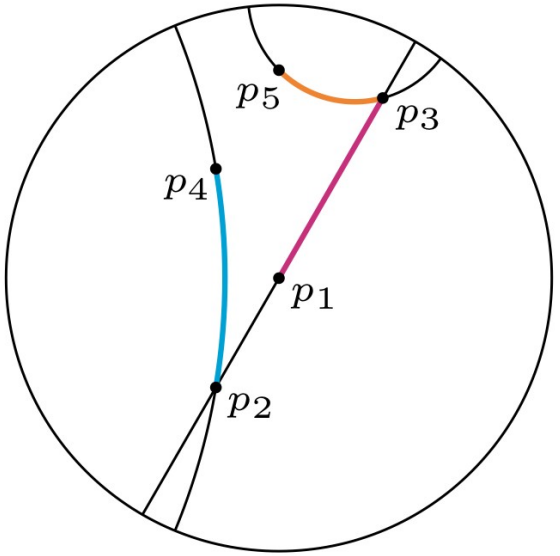
Affine transformation for prediction of hypernymy relations (Fu et al., 2014)

- Hypernyms: cat → animal, dog → animal, banana → fruit, apple → fruit, …
- Learn a linear projection from more specific word (hyponym) to more generic word (hypernym) using:

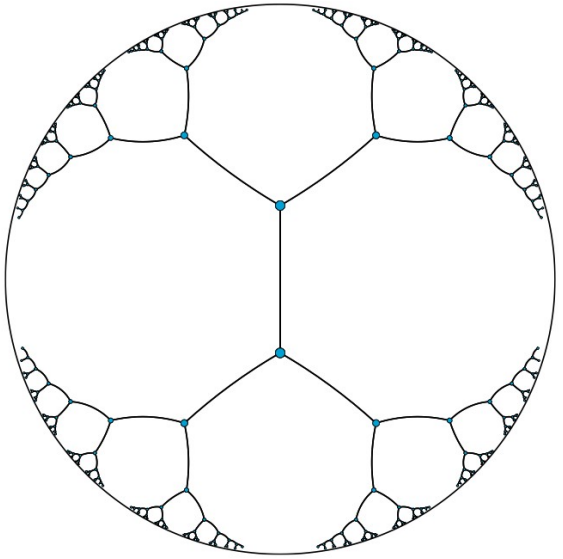
$$\mathbf{\Phi}^* = \arg\min_{\mathbf{\Phi}} \frac{1}{|\mathcal{P}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{P}} \|\boldsymbol{x}\mathbf{\Phi} - \boldsymbol{y}\|^2$$

- *P* – is a set of training hyponym-hypernym pairs

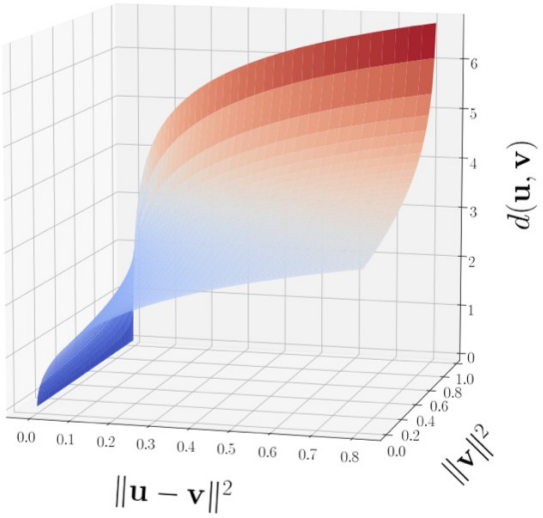
Hyperbolic (Poincaré) embeddings



(a) Geodesics of the Poincaré disk



(b) Embedding of a tree in $\mathcal{B}^2$



(c) Growth of Poincaré distance

Figure 1: (a) Due to the negative curvature of $\mathcal{B}$, the distance of points increases exponentially (relative to their Euclidean distance) the closer they are to the boundary. (c) Growth of the Poincaré distance $d(\boldsymbol{u}, \boldsymbol{v})$ relative to the Euclidean distance and the norm of $\boldsymbol{v}$ (for fixed $\|\boldsymbol{u}\| = 0.9$). (b) Embedding of a regular tree in $\mathcal{B}^2$ such that all connected nodes are spaced equally far apart (i.e., all black line segments have identical hyperbolic length).

Source: https://arxiv.org/pdf/1705.08039.pdf

Hyperbolic (Poincaré) embeddings

- Poincaré ball: $\mathcal{B}^d = \{\boldsymbol{x} \in \mathbb{R}^d \mid \|\boldsymbol{x}\| < 1\}$
- Distance on a ball between two points:

$$d(\boldsymbol{u}, \boldsymbol{v}) = \operatorname{arcosh}\left(1 + 2\frac{\|\boldsymbol{u} - \boldsymbol{v}\|^2}{(1 - \|\boldsymbol{u}\|^2)(1 - \|\boldsymbol{v}\|^2)}\right)$$

- Loss:

$$\mathcal{L}(\Theta) = \sum_{(u,v)\in\mathcal{D}} \log \frac{e^{-d(\boldsymbol{u},\boldsymbol{v})}}{\sum_{\boldsymbol{v}'\in\mathcal{N}(u)} e^{-d(\boldsymbol{u},\boldsymbol{v}')}}$$

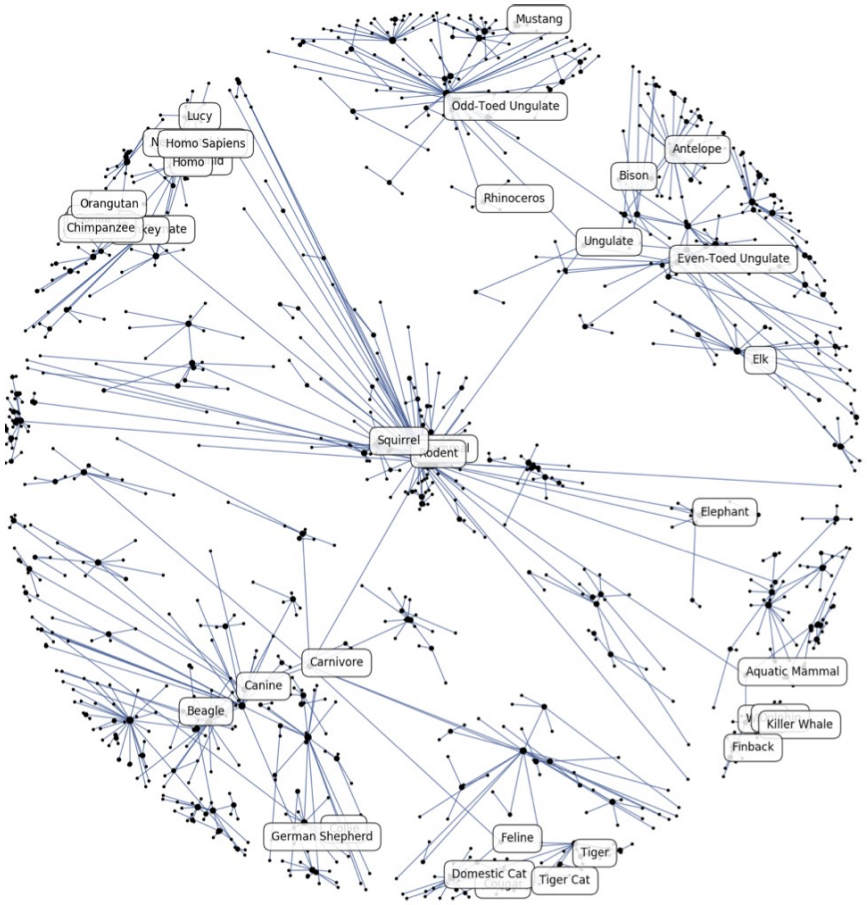
  - set of negative examples for *u:* $\mathcal{N}(u) = \{v \mid (u, v) \notin \mathcal{D}\} \cup \{u\}$
  - 10 negative samples per 1 positive

$$\Theta' \leftarrow \arg\min_{\Theta} \mathcal{L}(\Theta) \qquad \text{s.t. } \forall\, \boldsymbol{\theta}_i \in \Theta : \|\boldsymbol{\theta}_i\| < 1$$

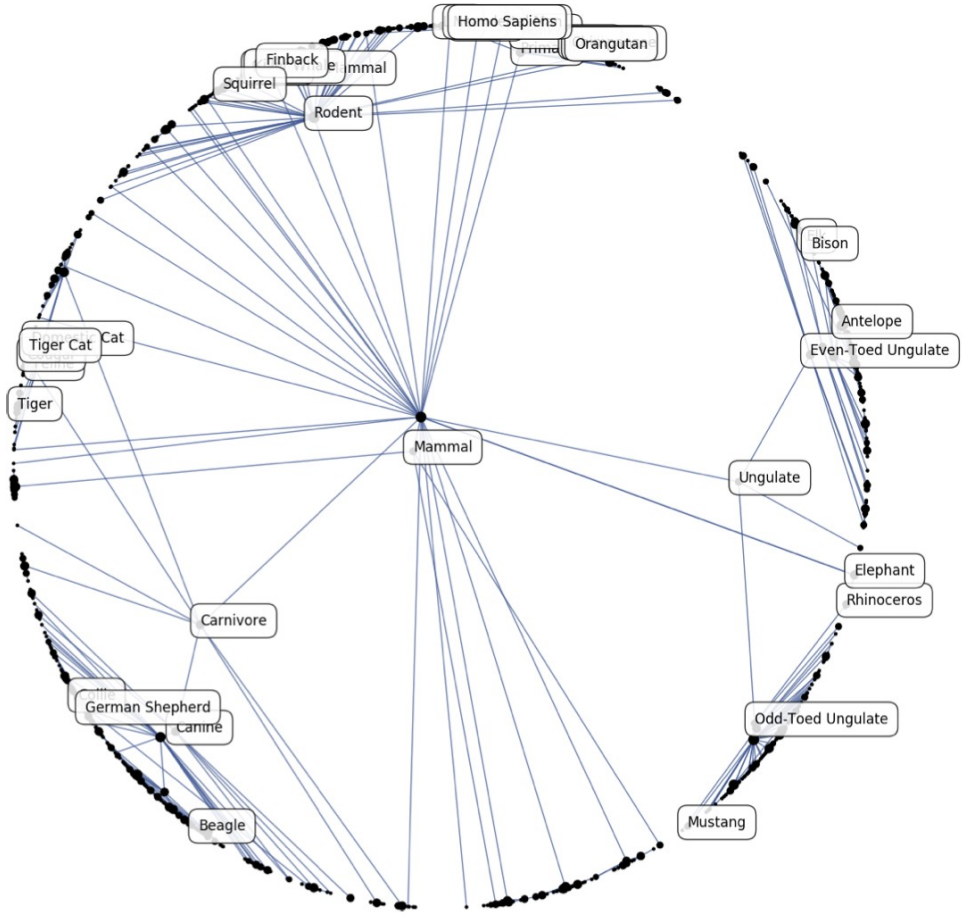
Source: https://arxiv.org/pdf/1705.08039.pdf

Trained on WordNet relations

- Two-dimensional Poincaré embeddings of transitive closure of the WordNet mammals subtree.



(a) Intermediate embedding after 20 epochs



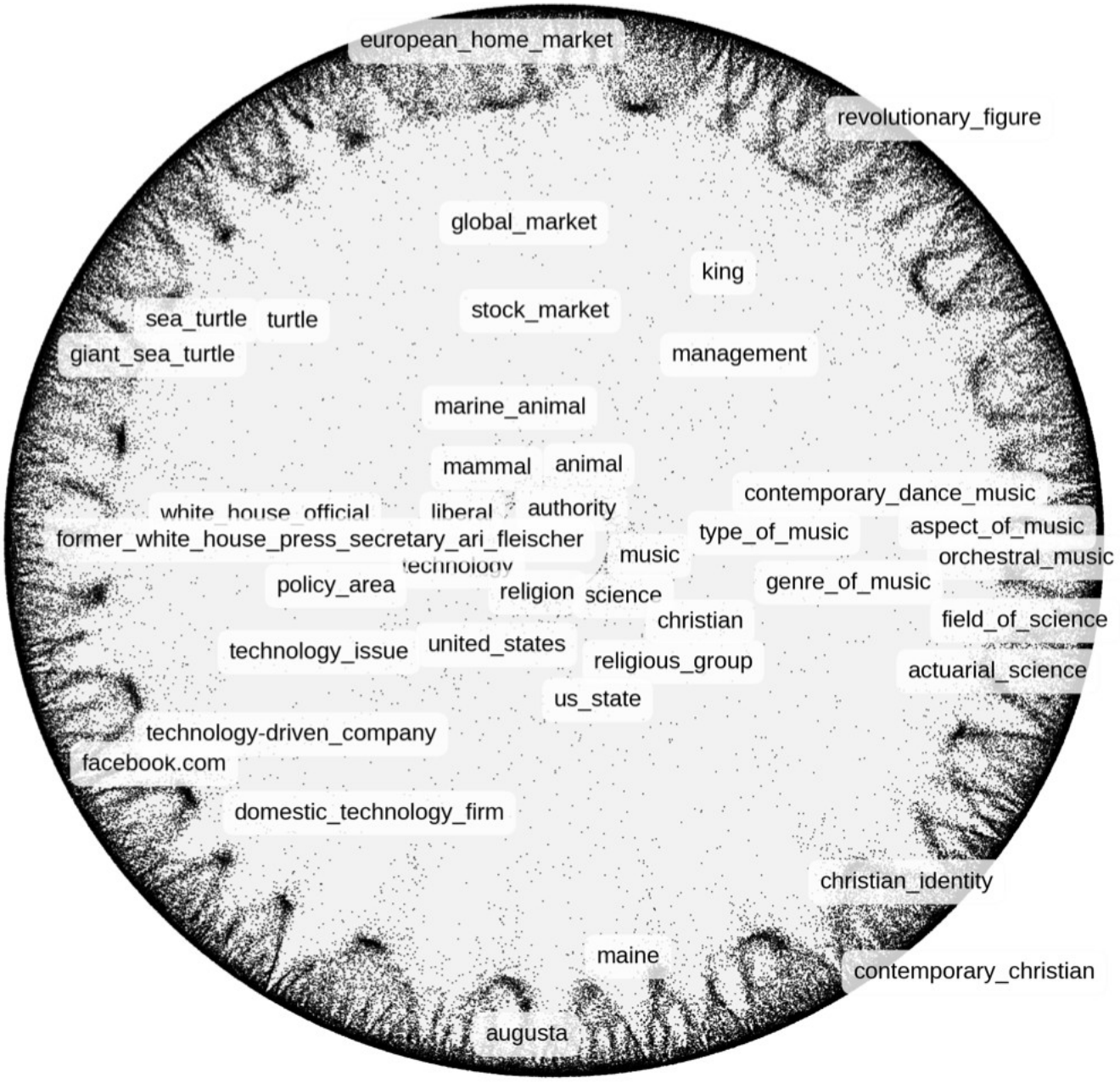
(b) Embedding after convergence

Hyperbolic (Poincaré) embeddings: Hearst patterns

Pattern
X which is a (example | class | kind | . . .) of Y
X (and | or) (any | some) other Y
X which is called Y
X is JJS (most)? Y
X a special case of Y
X is an Y that
X is a !(member | part | given) Y
!(features | properties) Y such as X_1, X_2, . . .
(Unlike | like) (most | all | any | other) Y, X
Y including X_1, X_2, . . .

Source: https://arxiv.org/pdf/1902.00913.pdf

Hyperbolic (Poincaré) embeddings



Source: https://arxiv.org/pdf/1902.00913.pdf