Word Representation

2/2565: FRA501 Introduction to Natural Language Processing with Deep learning

Week 05

Paisit Khanarsa, Ph.D.

Institute of Field Robotics (FIBO), King Mongkut's University of Technology Thonburi

Outlines

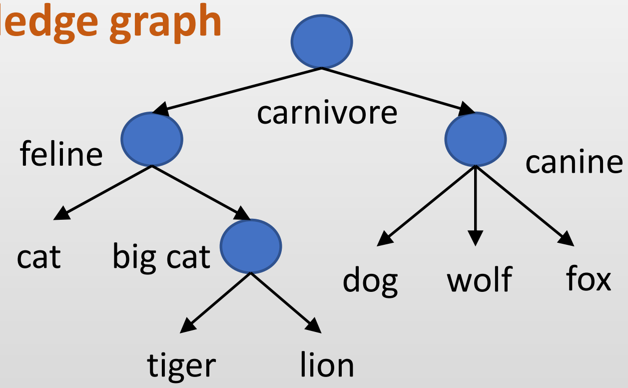
- How to represent words?
- Distributional: Sparse vector
- Distributional: Dense vector representations
- Word2Vec Evaluation
- Advanced Topics

How to represent words?

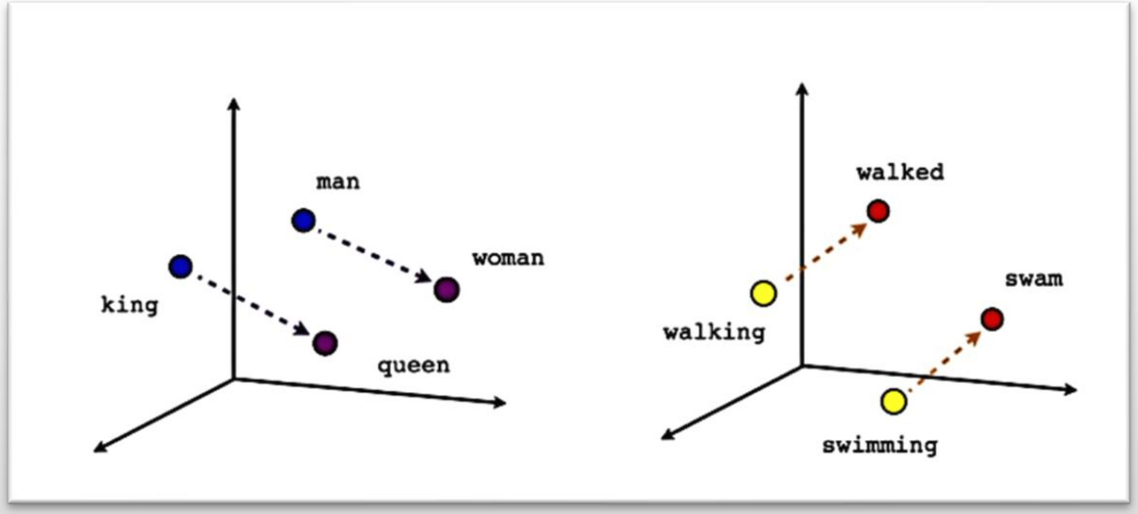
- Symbolic vs Distributional representations
  - Word representation is one of the most important tasks in NLP.
  - Words as input for our models

Dog = [0,0,0,1,0,0,…,0] **One-hot model**

Knowledge graph



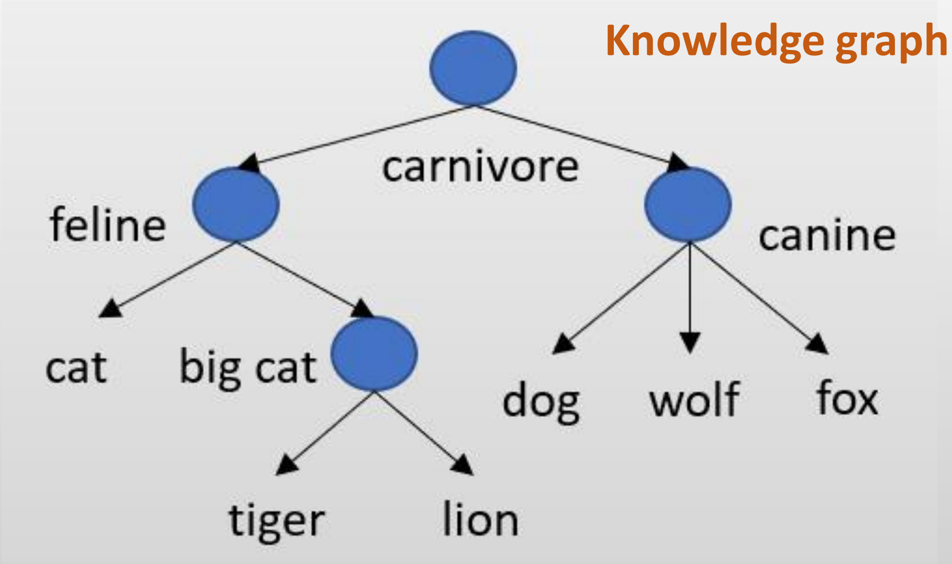
Symbolic



Distributional

How to represent words? (cont.)

- Symbolic representations
  - A lexical database, such as WordNet that has hypernyms and synonyms
  - Cons:
    - Requires human labor to create and update
    - Missing new words
    - Hard to compute accurate word similarity



How to represent words? (cont.)

- Symbolic representations
  - Earlier work in NLP, the vast majority of (rule-based and statistical) NLP models considered words as discreate atomic symbols.
  - E.g. One-hot model
    - Cat = [0 1 0 0 0 0 0 0 … 0]
    - Dog = [0 0 1 0 0 0 0 0 … 0]
    - Each point in the vector represents each vocab
  - Cons:
    - Cannot capture similarity between words

How to represent words? (cont.)

- Distributed representations
  - The meaning of a word is computed from the distribution of words around it.
  - Can encode similarity between words

การลดจำนวนลงชอง**เสือ**ซึ่งเป็นสัตว์กินเนื้อ
จะส่งผลกระทบต่อระบบนิเวศ

These words represent “เสือ”

เสือเป็นสัตว์เลี้ยงลูกด้วยนมในวงศ์เดียวกับ
แมว

Distributional: Sparse vector

- Term – Frequency (Raw frequency)
- Co – Occurrence (Raw frequency)
- Positive Pointwise Mutual Information (PPMI)
- Term Frequency – Inverse Document Frequency (TF-IDF)

Sparse vector: Term-document matrix

- Each row representations a word in the vocabulary and term-document matrix.
- Each column represents a document.

Vocabulary

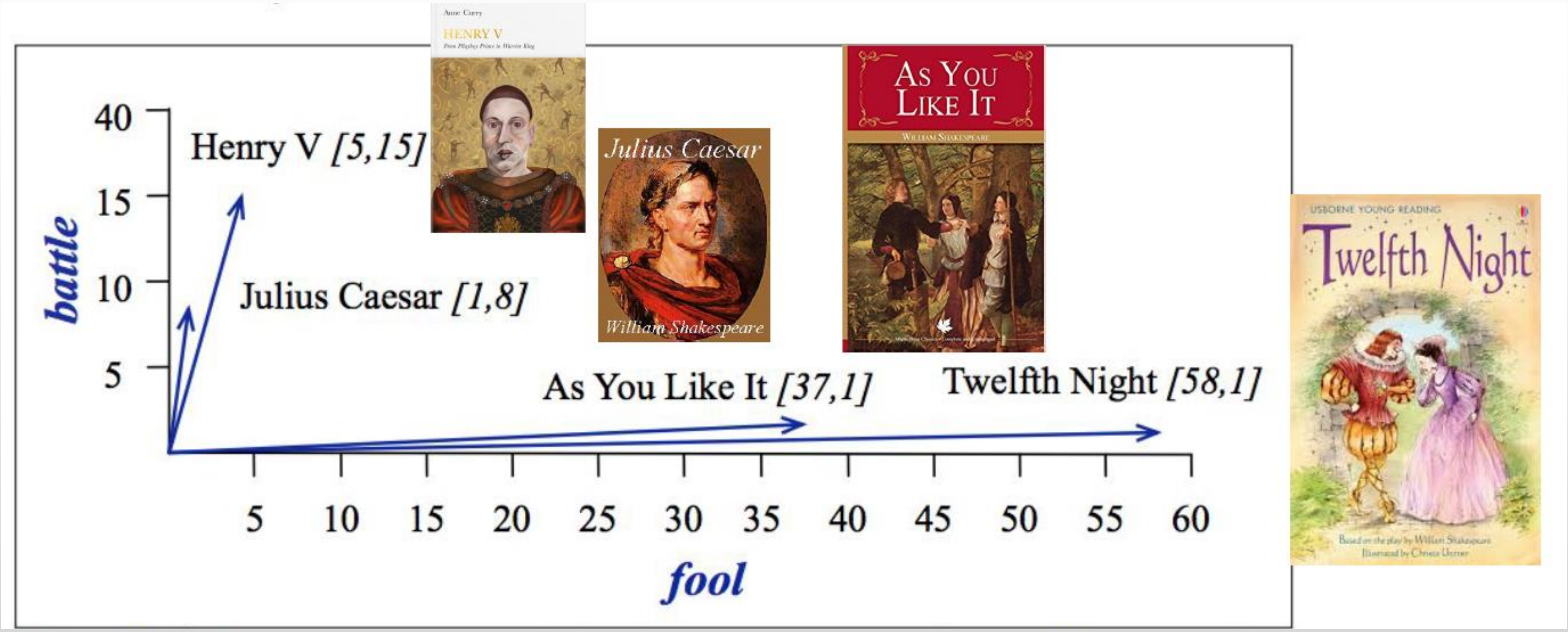
Documents

	As You Like It	Twelfth Night	Julius Caesar	Henry V
Battle	1	1	8	15
Soldier	2	2	12	36
Fool	37	58	1	5
Clown	5	117	0	0

The term-document matrix for four words in four Shakespears plays.

Sparse vector: Term-document matrix (cont.)

- Application: Document Information Retrieval
  - Two documents that are similar tend to have similar word/vector (document similarity)



Sparse vector: Term-document matrix (cont.)

- Two documents are similar if their vectors are similar (document similarity)
- Two words are similar if their vector are similar (word similarity)

Vocabulary

Document similarity

Documents

	As You Like It	Twelfth Night	Julius Caesar	Henry V
Battle	1	1	8	15
Soldier	2	2	12	36
Fool	37	58	1	5
Clown	5	117	0	0

Word similarity

Sparse vector: Co-occurrence matrix

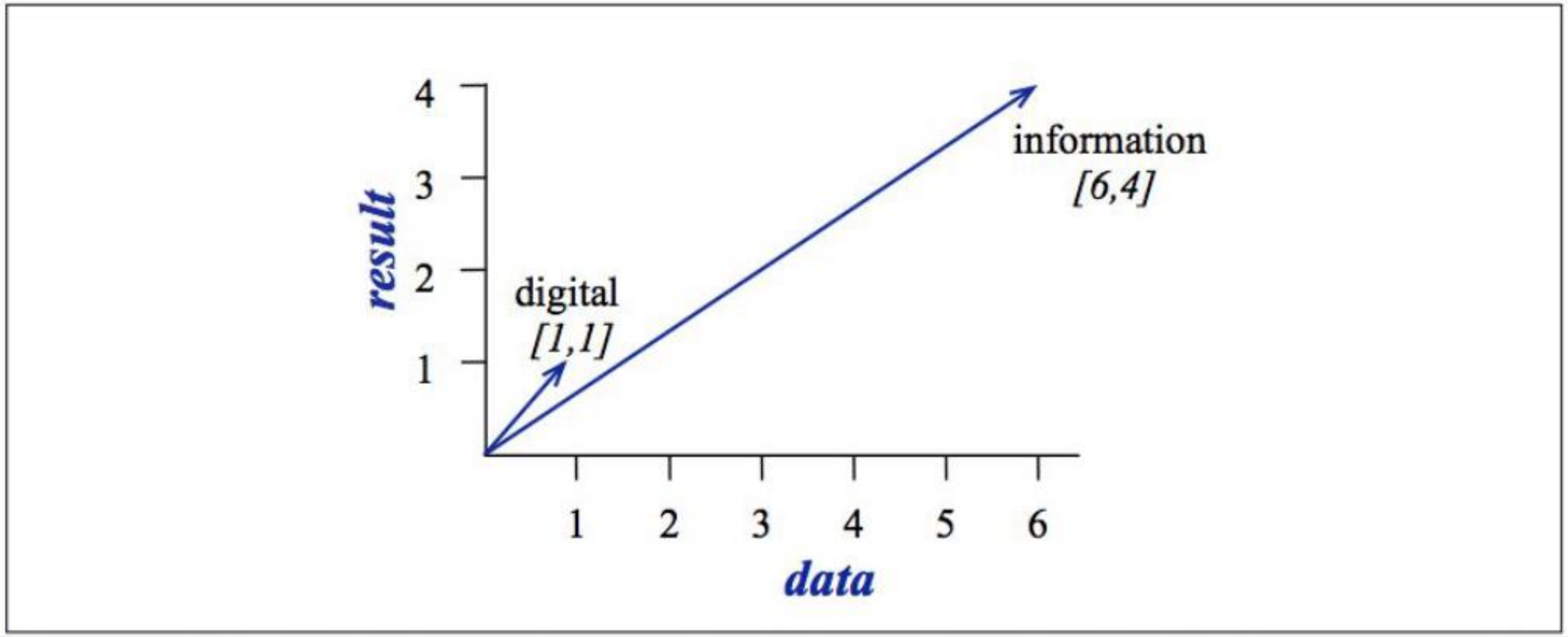
- Word-word or word-context matrix
- Two words are similar if their vector are similar

Window = 4

Vocabulary	aarbvark	…	computer	data	pinch	result	sugar	…
apricot	0	…	0	0	1	0	1	…
pineapple	0	…	0	0	1	0	1	…
digital	0	…	2	1	0	1	0	…
information	0	…	1	6	0	4	0	…

Sparse vector: Co-occurrence matrix (cont.)

- Two similar words tend to have similar vector (word similarity)



Window = 4

Vocabulary	aarbvark	...	computer	data	pinch	result	sugar	...
apricot	0	...	0	0	1	0	1	...
pineapple	0	...	0	0	1	0	1	...
digital	0	...	2	1	0	1	0	...
information	0	...	1	6	0	4	0	...

Sparse vector: Positive Pointwise Mutual Information (PPMI)

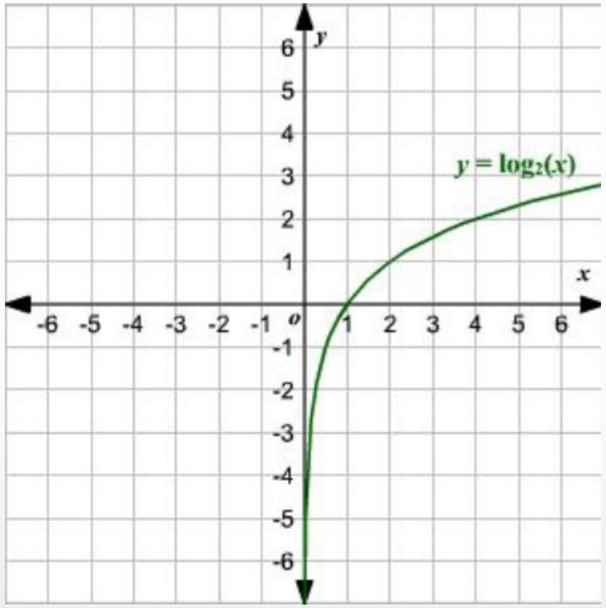
- Problems with raw frequency !!!
  - Not very discriminative (need normalization)
    - Words such as “it, the, they, a, an, the” occur very frequently
- PPMI incorporates the idea of mutual information to determine the context words

Sparse vector: Positive Pointwise Mutual Information (PPMI) (cont.)

How often the two words occur together

$$PMI(w,c) = \log_2 \frac{\boxed{P(w,c)}}{\boxed{P(w)P(c)}}$$

How often the two words occur if they occur independently (occur by chance)



- w: target word
- c: context word

- + : occur together > occur by chance
- 0 : occur together = occur by chance
- - : occur together < occur by chance

Sparse vector: Positive Pointwise Mutual Information (PPMI) (cont.)

- Negative PMI values tend to be unreliable, so we replace all negative PMI values with zero

$$PPMI(w,c) = max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

Co-occurrence matrix

Vocabulary	computer	data	pinch	result	sugar	...
apricot	0	0	1	0	1	...
pineapple	0	0	1	0	1	...
digital	2	1	0	1	0	...
information	1	6	0	4	0	...

PPMI matrix

Vocabulary	computer	data	pinch	result	sugar	...
apricot	0	0	2.25	0	2.25	...
pineapple	0	0	2.25	0	2.25	...
digital	1.66	0	0	0	0	...
information	0	0.57	0	0.47	0	...

Sparse vector: TF-IDF

- Term Frequency (TF) – per each document
  - $TF(w) = \frac{Frequency\ of\ word\ "w"\ in\ a\ ducument(f)}{Total\ number\ of\ words\ in\ the\ ducument}$

Doc 1 → cat = 5/10
Doc 2 → cat = 50/100

- Inverse Document Frequency (IDF) – per corpus (all documents)
  - $IDF(w) = \log_e(\frac{Total\ number\ of\ ducments}{Number\ of\ documents\ that\ contain\ word\ "w"})$

Penalty score
i.e., a, an ,the

- $TFIDF(w) = TF(w) \times IDF(w)$

Sparse vector: TF-IDF (cont.)

- Term Frequency – Original TF

Document 1

The sky is blue. The sky is beautiful.

$$f(\text{The}, \text{doc1}) = 2$$

$$f(\text{sky}, \text{doc1}) = 2$$

$$f(\text{is}, \text{doc1}) = 2$$

$$f(\text{blue}, \text{doc1}) = 1$$

$$f(\text{beautiful}, \text{doc1}) = 1$$

$$TF(w) = \frac{Frequency\ of\ word\ "w"\ in\ a\ ducument(f)}{Total\ number\ of\ words\ in\ the\ ducument}$$

$$\text{TF}(\text{The}, \text{doc1}) = \frac{2}{8}$$

$$\text{TF}(\text{sky}, \text{doc1}) = \frac{2}{8}$$

$$\text{TF}(\text{is}, \text{doc1}) = \frac{2}{8}$$

$$\text{TF}(\text{blue}, \text{doc1}) = \frac{1}{8}$$

$$\text{TF}(\text{beautiful}, \text{doc1}) = \frac{1}{8}$$

Sparse vector: TF-IDF (cont.)

This method is suitable for data with a very different number of word frequencies.

- Term Frequency – Log Normalization

Document 1

The sky is blue. The sky is beautiful.

$$logNormalization(w) = \log(1 + Frequency\ of\ word\ "w"\ in\ a\ ducument(f))$$

$$f(\text{The}, \text{doc1}) = 2$$
$$f(\text{sky}, \text{doc1}) = 2$$
$$f(\text{is}, \text{doc1}) = 2$$
$$f(\text{blue}, \text{doc1}) = 1$$
$$f(\text{beautiful}, \text{doc1}) = 1$$

$$logNormalization(\text{The}, \text{doc1}) = \log(1 + 2)$$
$$logNormalization(\text{sky}, \text{doc1}) = \log(1 + 2)$$
$$logNormalization(\text{is}, \text{doc1}) = \log(1 + 2)$$
$$logNormalization(\text{blue}, \text{doc1}) = \log(1 + 1)$$
$$logNormalization(\text{beautiful}, \text{doc1}) = \log(1 + 1)$$

Sparse vector: TF-IDF (cont.)

This method is suitable for data whose document length is different or very different.

- Term Frequency – Double Normalization

Document 1

The sky is blue. The sky is beautiful.

$$DoubleNormalization(w) = K + (1-K)\frac{Frequency\ of\ word\ "w"\ in\ a\ ducument(f)}{Max\ Frequency\ of\ word\ in\ a\ ducument(f)}$$

$K = 0.5$

$$f(\text{The}, \text{doc1}) = 2$$
$$f(\text{sky}, \text{doc1}) = 2$$
$$f(\text{is}, \text{doc1}) = 2$$
$$f(\text{blue}, \text{doc1}) = 1$$
$$f(\text{beautiful}, \text{doc1}) = 1$$

$$DoubleNormalization(\text{The}, \text{doc1}) = 0.5 + 0.5\frac{2}{2}$$
$$DoubleNormalization(\text{sky}, \text{doc1}) = 0.5 + 0.5\frac{2}{2}$$
$$DoubleNormalization(\text{is}, \text{doc1}) = 0.5 + 0.5\frac{2}{2}$$
$$DoubleNormalization(\text{blue}, \text{doc1}) = 0.5 + 0.5\frac{1}{2}$$
$$DoubleNormalization(\text{beautiful}, \text{doc1}) = 0.5 + 0.5\frac{1}{2}$$

Sparse vector: TF-IDF (cont.)

TF-IDF Matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.0018	0.022

- Example
  - $TF(\text{wit, As You Like It}) = logNormalization(\text{wit, As You Like It}) = \log_{10}(20 + 1) = 1.322$
  - $IDF(\text{wit}) = 0.037$
  - $TF - IDF(\text{wit}) = 1.322 \times 0.037 = 0.049$

Distributional: Dense vector

- SVD-based method
- Word2Vec
  - Skip-gram
  - CBOW
- Word2Vec training methods
- Pre-trained vector representations

Distributional: Dense vector (cont.)

- Sparse vector representations
  - Long (length of vectors ≈ 20,000 to 50,000)
  - Sparse (most element are zero)
- Dense vector representations
  - Reduce length of vector (length of vector ≈ 200 to 1,000)
  - Reduce sparsity

Dense vector: SVD-based method

- Single Value Decomposition (SVD)
- Matrix factorization

numpy.linalg.svd

linalg.**svd**(*a*, *full_matrices=True*, *compute_uv=True*, *hermitian=False*) [source]

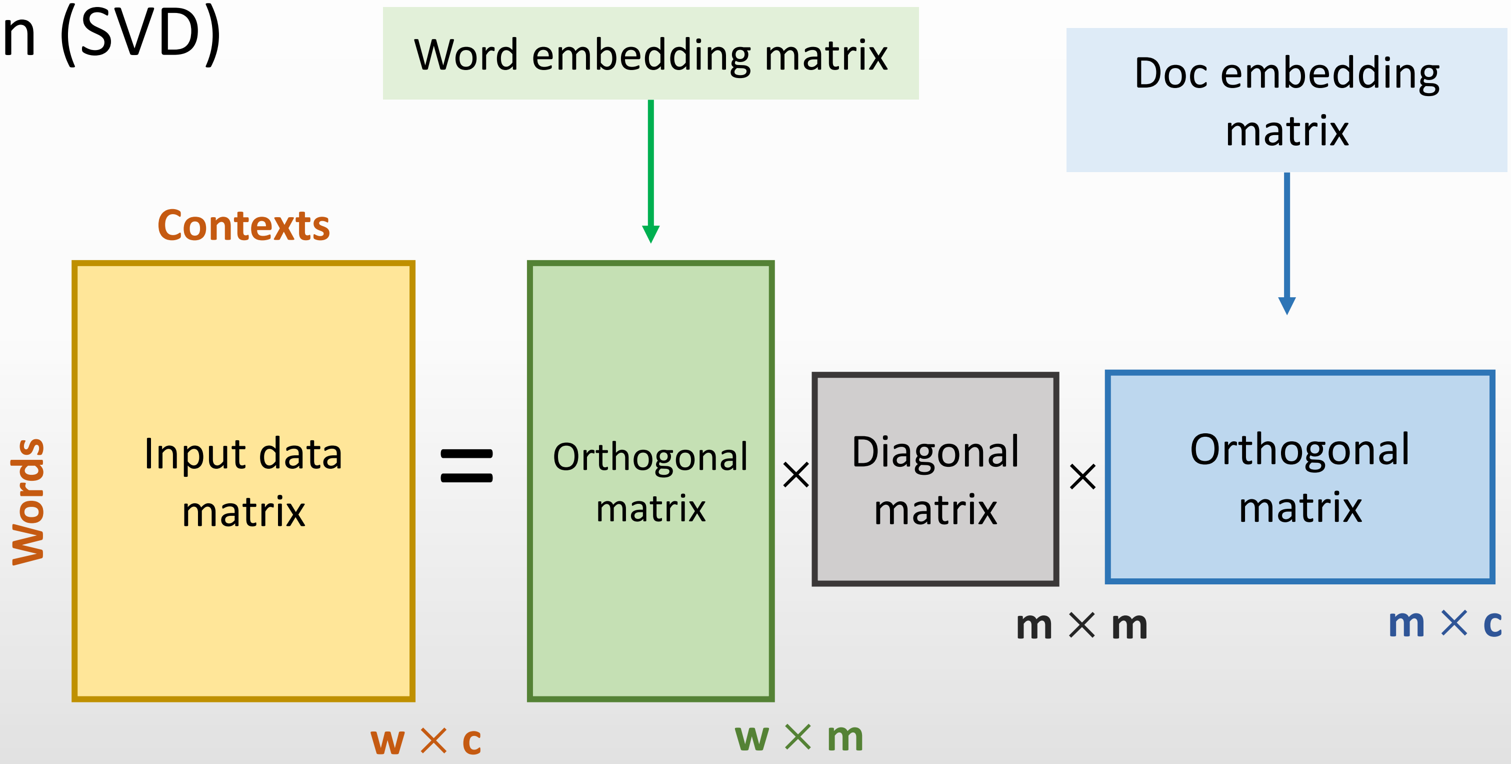
Singular Value Decomposition.

When *a* is a 2D array, and `full_matrices=False`, then it is factorized as `u @ np.diag(s) @ vh = (u * s) @ vh`, where *u* and the Hermitian transpose of *vh* are 2D arrays with orthonormal columns and *s* is a 1D array of *a*'s singular values. When *a* is higher-dimensional, SVD is applied in stacked mode as explained below.

```
>>> a = np.random.randn(9, 6) + 1j*np.random.randn(9, 6)
```

```
>>> u, s, vh = np.linalg.svd(a, full_matrices=True)
>>> u.shape, s.shape, vh.shape
((9, 9), (6,), (6, 6))
```

https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html



- w: total words
- c: total documents or context words
- m: total latent factors

Dense vector: SVD-based method (cont.)

- Single Value Decomposition (SVD)
- Matrix factorization

numpy.linalg.svd

`linalg.svd(a, full_matrices=True, compute_uv=True, hermitian=False)` [source]

Singular Value Decomposition.

When *a* is a 2D array, and `full_matrices=False`, then it is factorized as `u @ np.diag(s) @ vh = (u * s) @ vh`, where *u* and the Hermitian transpose of *vh* are 2D arrays with orthonormal columns and *s* is a 1D array of *a*'s singular values. When *a* is higher-dimensional, SVD is applied in stacked mode as explained below.

```
>>> a = np.random.randn(9, 6) + 1j*np.random.randn(9, 6)
```

```
>>> u, s, vh = np.linalg.svd(a, full_matrices=True)
>>> u.shape, s.shape, vh.shape
((9, 9), (6,), (6, 6))
```

https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html

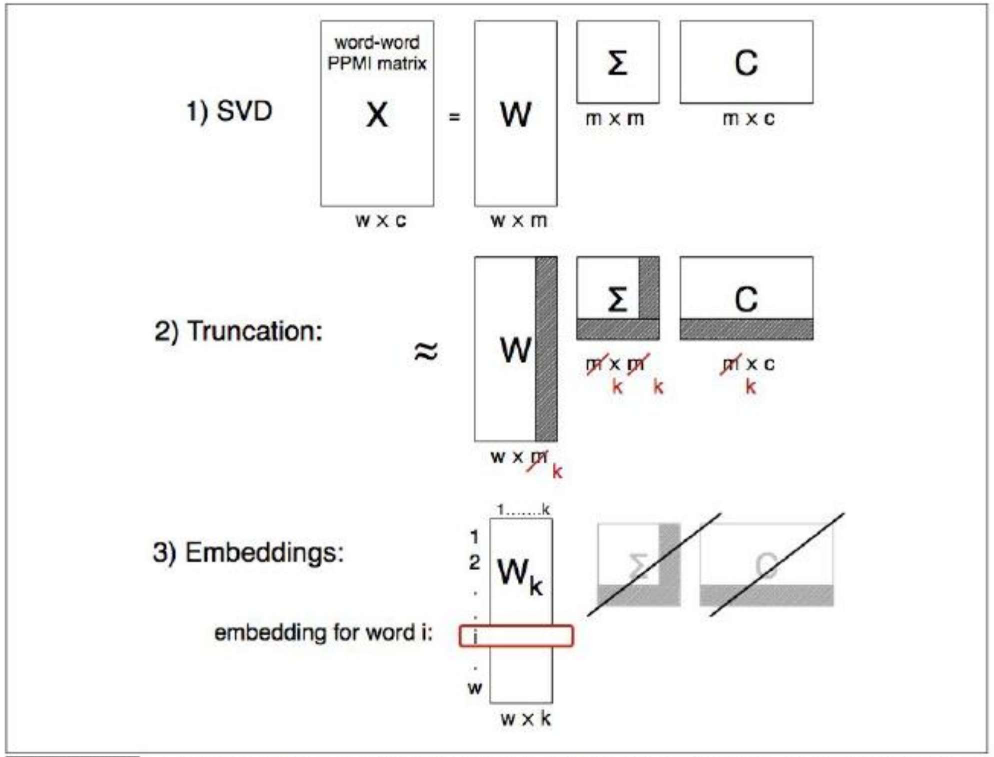
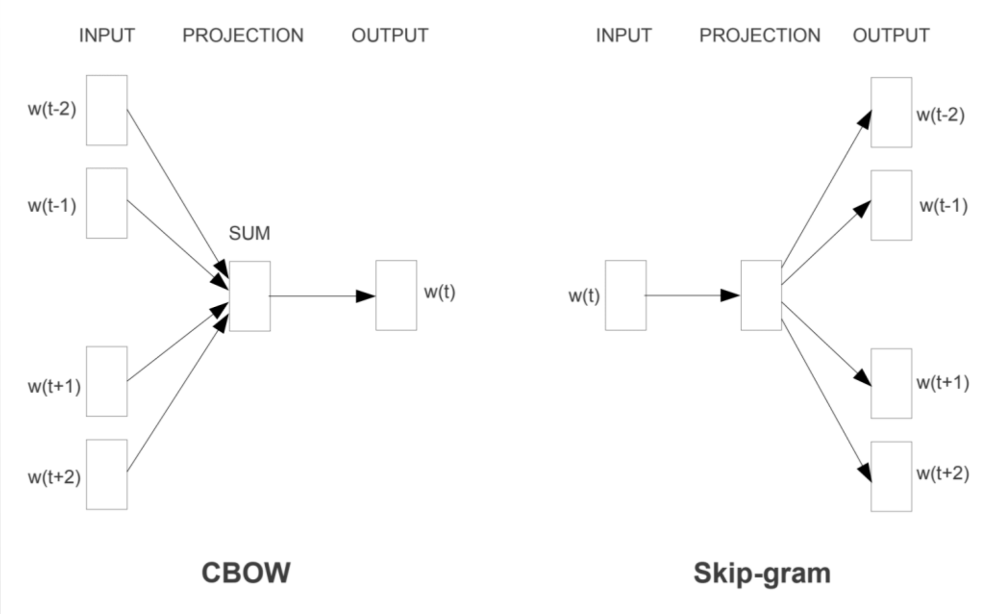


Figure 16.3 Sketching the use of SVD to produce a dense embedding of dimensionality k from a sparse PPMI matrix of dimensionality c. The SVD is used to factorize the word-word PPMI matrix into a W, Σ, and C matrix. The Σ and C matrices are discarded, and the W matrix is truncated giving a matrix of k-dimensionality embedding vectors for each word.

Dense vector: Word2Vec

- How do we train embeddings in neural network?
- Tomas Mikolov introduced **Skip-gram** in 2023
- CBOW was proposed before by other researches
- Train a neural network to predict neighboring words
- Pros
    - Faster than SVD
    - Pre-trained word representations are available online

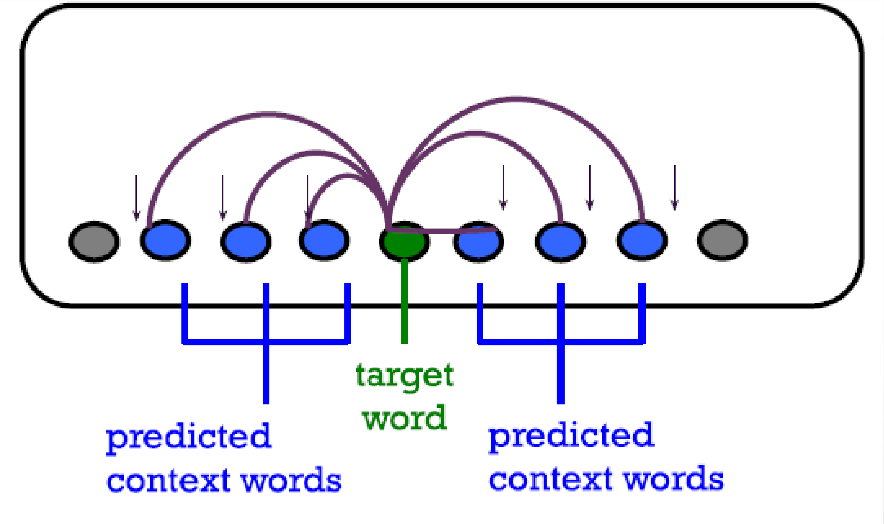
Tomas Mikolov



https://machinelearningmastery.com/what-are-word-embeddings/

Dense vector: Word2Vec – Skip-gram

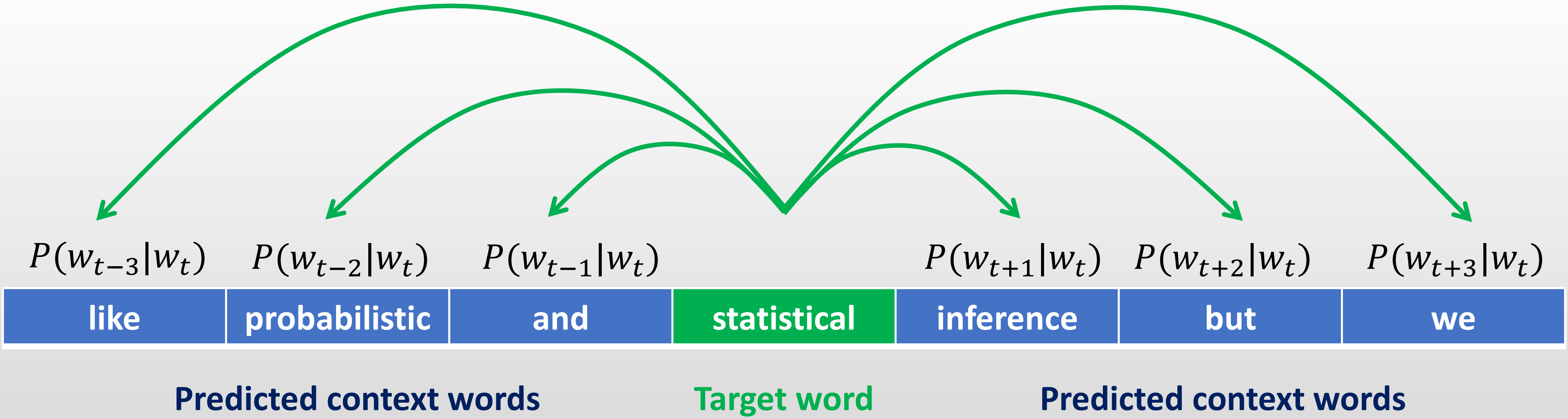
- In skip-gram neural language model, several context words are predicted from one target word.
- In “Efficient estimation of word representation in vector space”, Mikolov shows that skip-grams performs better than CBOW but it requires more training time.



Example
Outside of China there have been more than 500 cases in nearly 30 countries. Four people have died – in **France**, Hong Kong, the Philippines and Japan

Dense vector: Word2Vec – Skip-gram (cont.)

- Example
  - "I think it is much more likely that human language learning involves something like probabilistic and **statistical** inference but we just don't know yet"



Dense vector: Word2Vec – Skip-gram (cont.)

- Likelihood function: Given the target word, maximize the probability of each context word

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

- Cost/Loss function (Negative Log-likelihood)

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

Dense vector: Word2Vec – Skip-gram (cont.)

- Likelihood function: Given the target word, maximize the probability of each context word

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

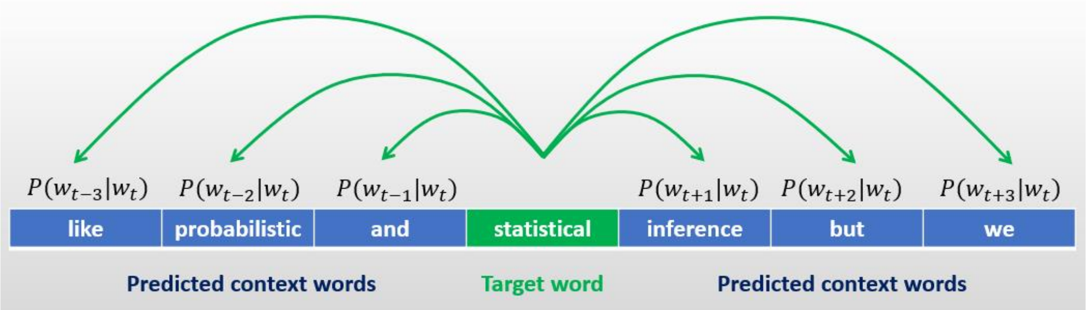
- Cost/Loss function (Negative Log-likelihood)

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

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

Dense vector: Word2Vec – Skip-gram (cont.)

- How to calculate $P(w_{t+j} \,|w_t;\theta)$?
  - v when w is a target/center word
  - u when w is a context word



- Then for a center word c and a context word o

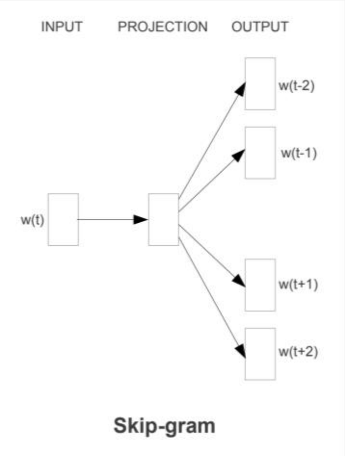
Dot product compares similarity of o and c.
Larger dot product = larger probability

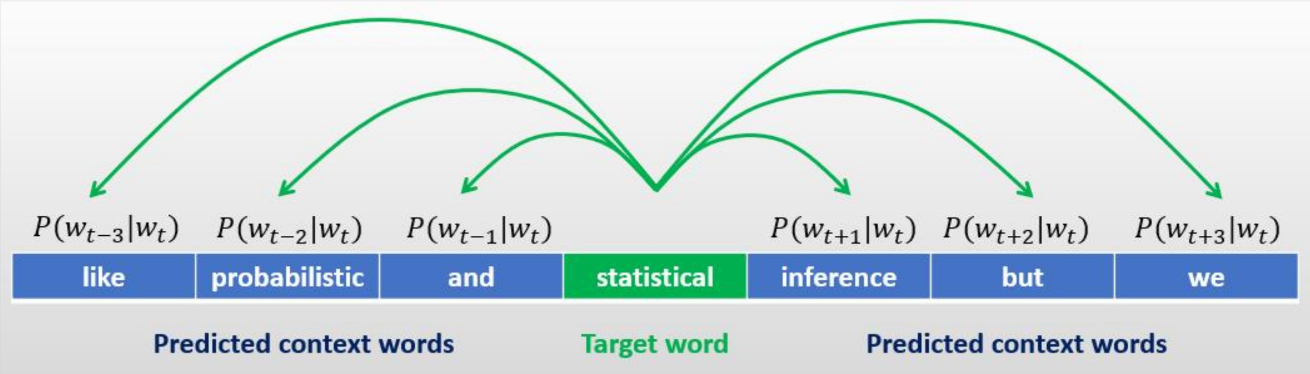
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \epsilon V} \exp(u_w^T v_c)}$$

After taking exponent, normalize over entire vocabulary

Dense vector: Word2Vec – Skip-gram (cont.)

- Skip-gram model step-by-step:
  1. Generate a one hot input vector for of the target word (center word)
  2. Get the embedded vector for the target center word
  3. Generate $2 \times m$ score vectors (where m is the window size)
  4. Turn the score vector into probabilities
  5. We desire our probabilities vector generated to match the true probabilities which are the one hot vectors of the actual output.

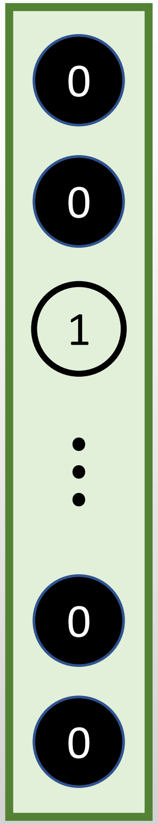




Dense vector: Word2Vec – Skip-gram (cont.)

- 1. Generate a one hot input vector x; $x \in \mathbb{R}^{|V|}$ for of the target word (center word)
  - Example: statistical = [0 0 1 0 … 0 0]

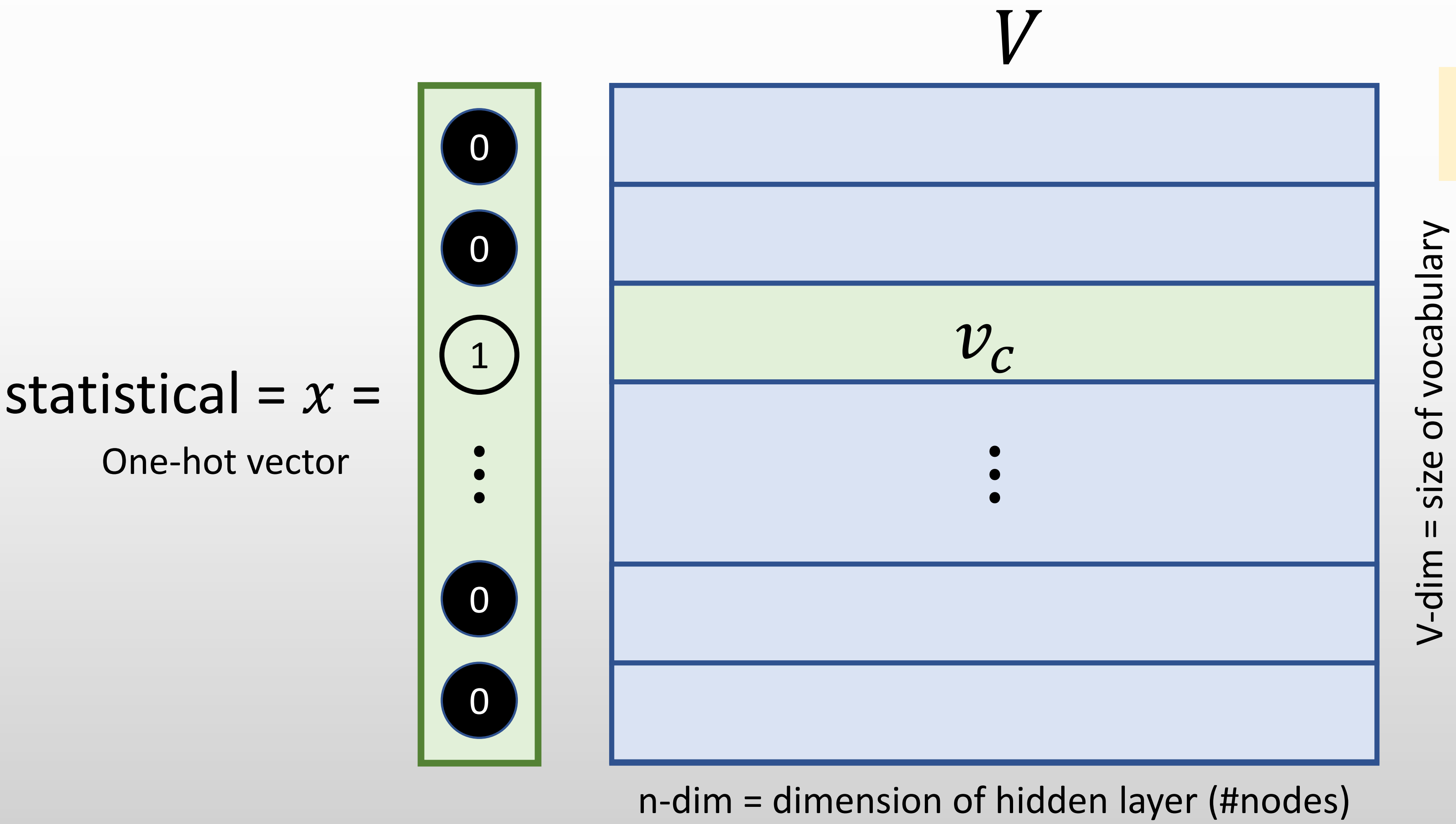
statistical = x =



Dim = Size of the vocabulary

Dense vector: Word2Vec – Skip-gram (cont.)

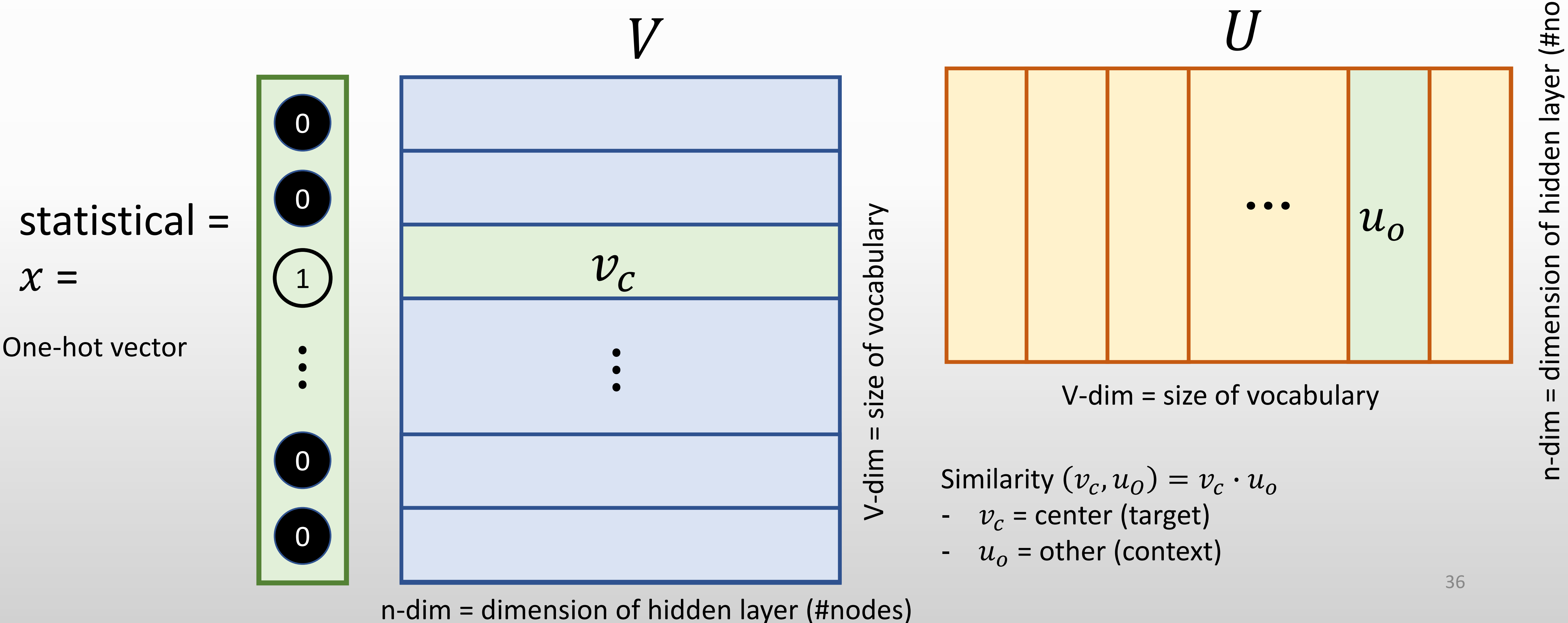
- 2. Get the embedded vector for the target center word



$$v_c = Vx \in \mathbb{R}^n$$

Dense vector: Word2Vec – Skip-gram (cont.)

- 3. Generate a score vector $z = Uv_c \in \mathbb{R}^V$



Similarity $(v_c, u_O) = v_c \cdot u_o$

- v_c = center (target)
- u_o = other (context)

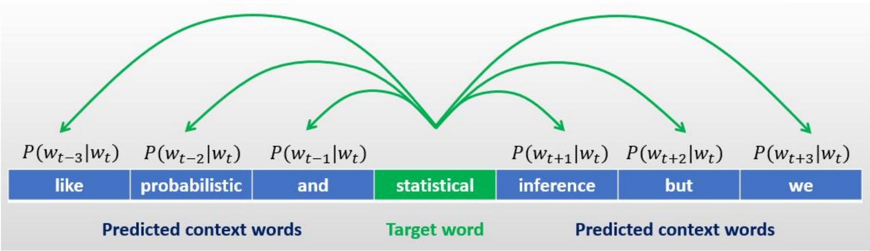
Dense vector: Word2Vec – Skip-gram (cont.)

- 4. Turn score into probabilities

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

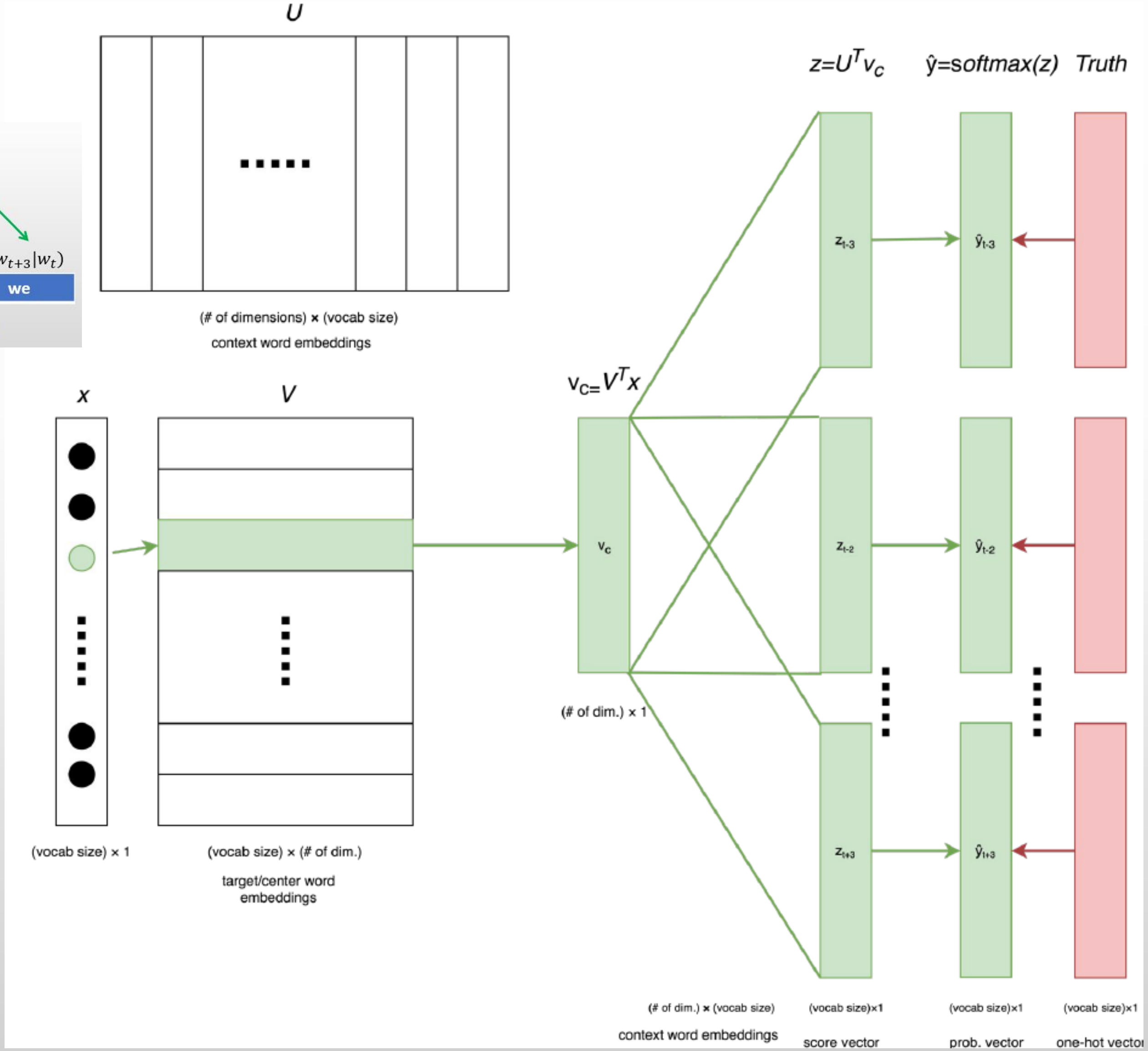
- 5. We desire our probabilities vector generated to match the true probabilities which are the one hot vectors of the actual output.

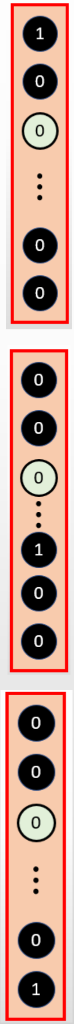
Dense vector: Word2Vec – Skip-gram (cont.)



statistical = x =

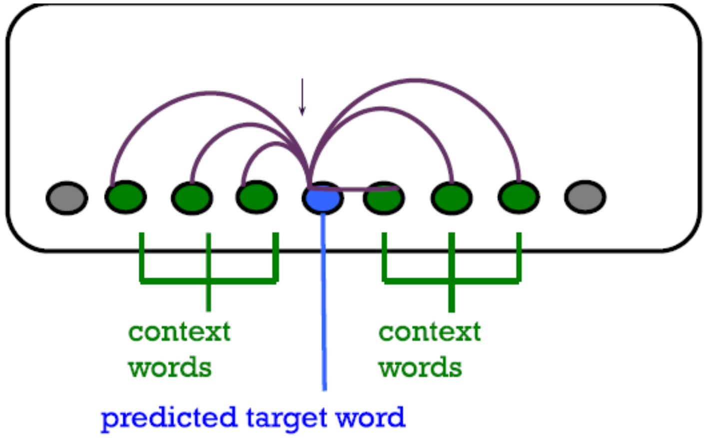
One-hot vector





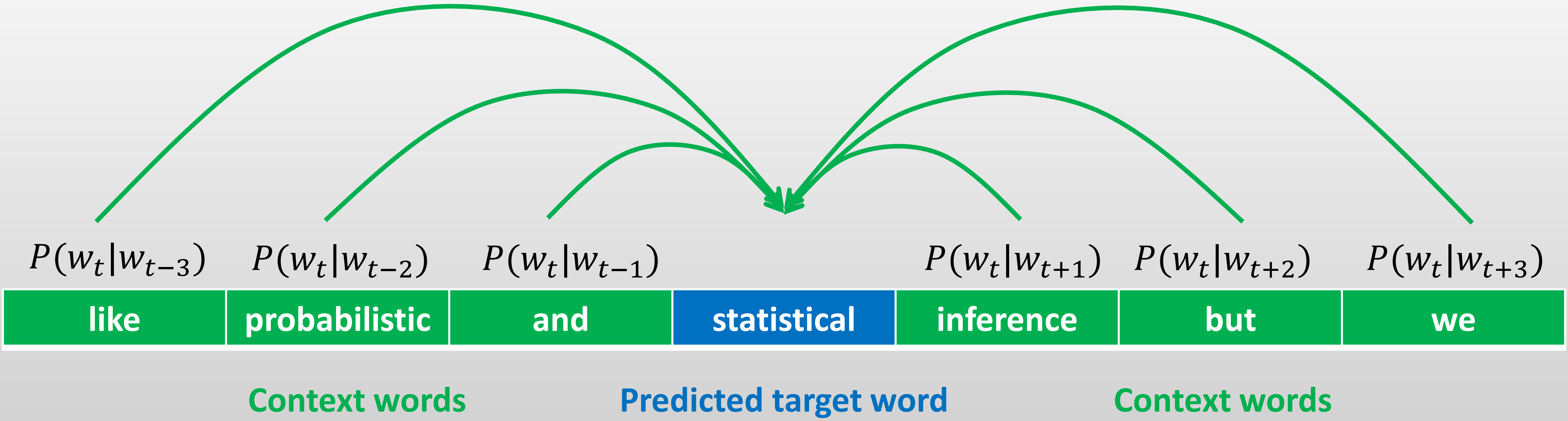
The size of output layer = #vocab

Dense vector: Word2Vec – CBOW

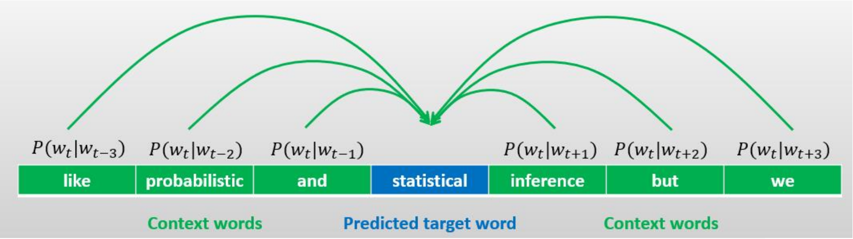


- Continuous Bag-of-Words (CBOW)
- In CBOW neural language model, one target word is predicted form several context words

- Example
  - "I think it is much more likely that human language learning involves something like probabilistic and <u>statistical</u> inference but we just don't know yet"

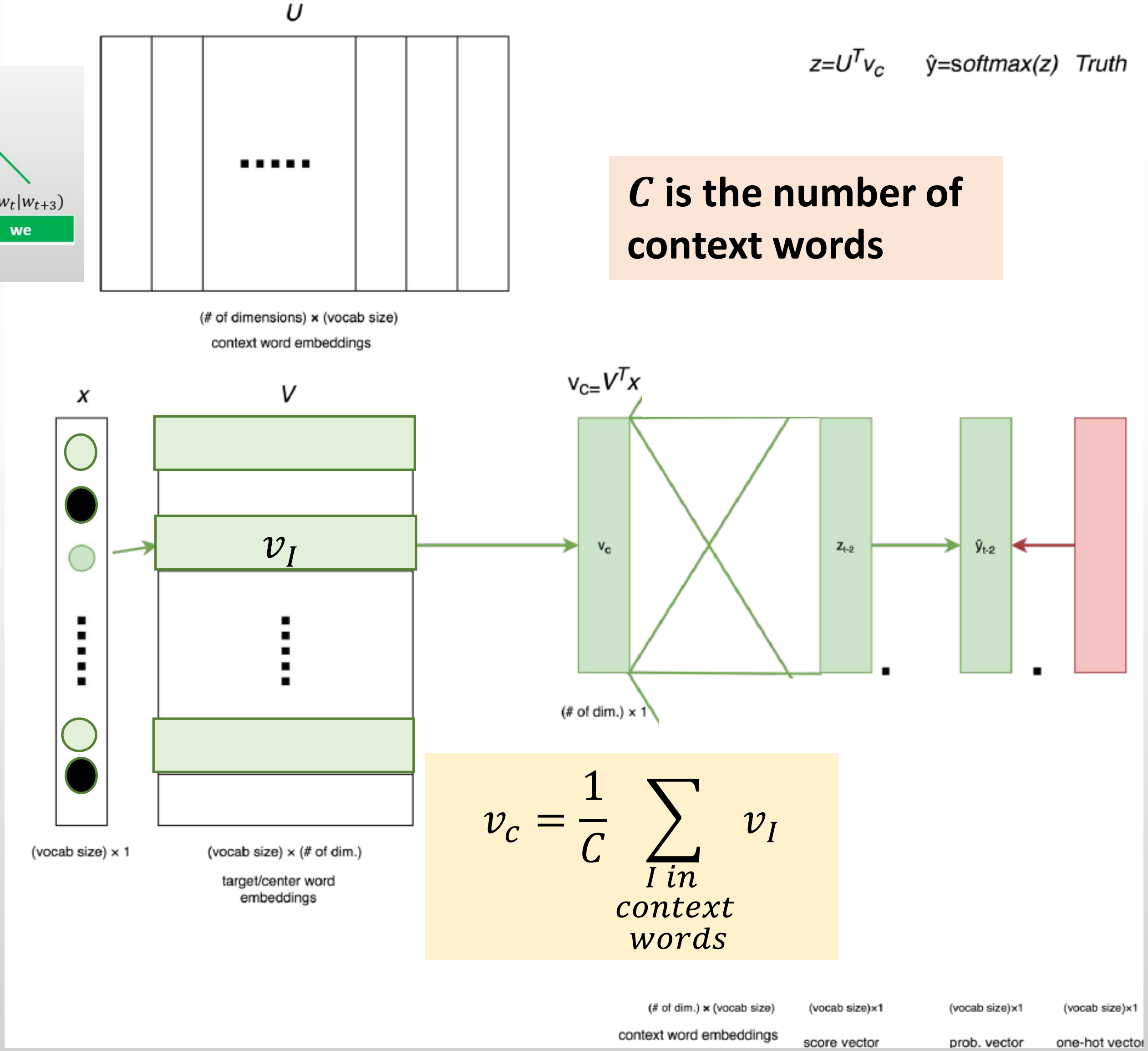


Dense vector: Word2Vec – CBOW (cont.)



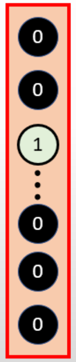
Like, probabilistic, and, inference, but we $= x =$

One-hot vector



C is the number of context words

$$v_c = \frac{1}{C} \sum_{I \text{ in context words}} v_I$$



The size of output layer = #vocab

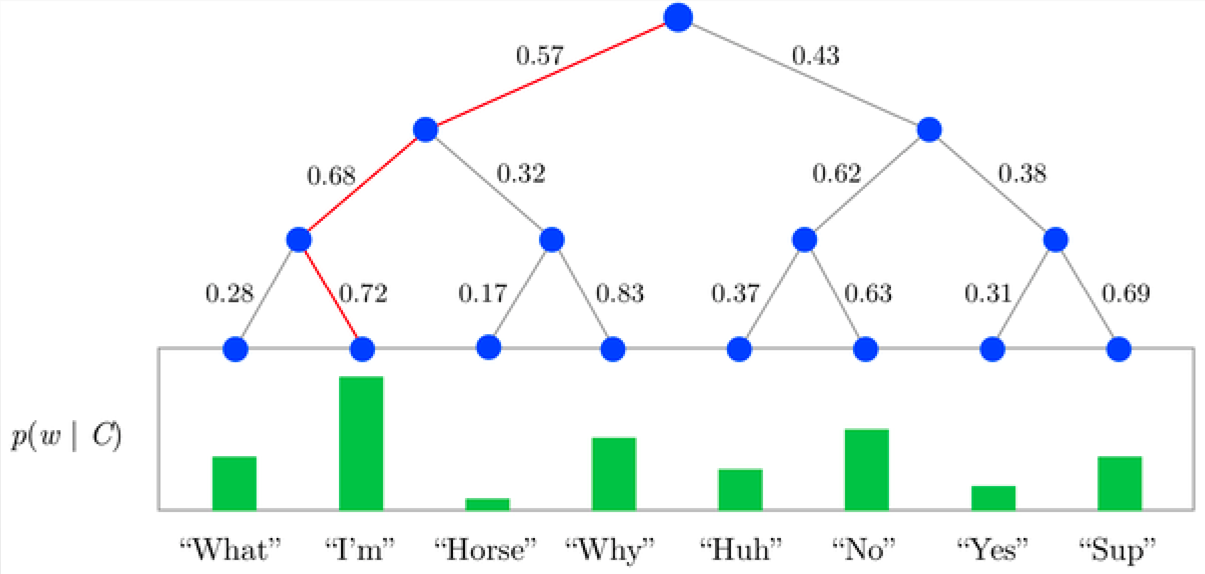
Dense vector: Word2Vec – Skip-gram (cont.)

- Pros
  - In "Efficient estimation of word representation in vector space", Mikolov shows that skip-grams performs better than CBOW
- Cons
  - Softmax is not very efficient (slow)
  - High computational cost
- Solutions
  - Hierarchical Softmax
  - Negative Sampling

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

Dense vector: Word2Vec – Skip-gram (cont.)

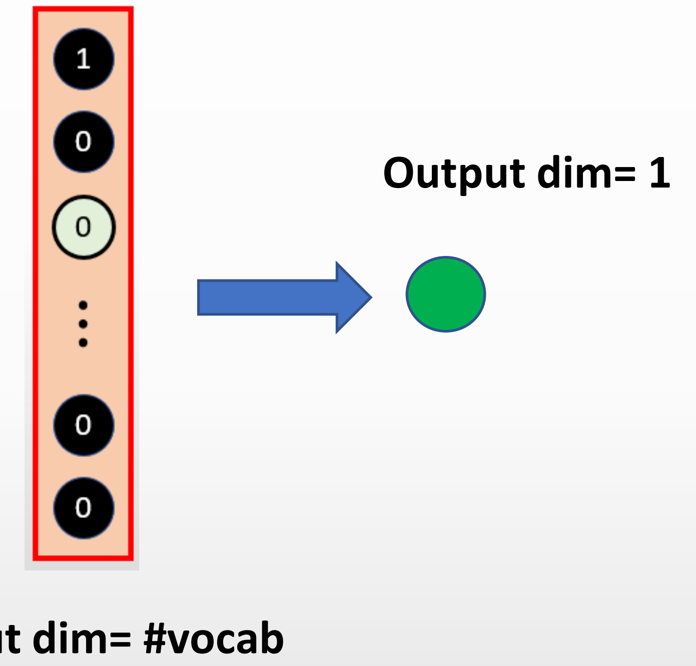
- Hierarchical Softmax
  - Softmax as tree traversal
  - Each leaf is a word : There's a unique path from root to leaf
  - The probability of each word is the product of branch selection decisions from the root to the word's leaf



https://paperswithcode.com/method/hierarchical-softmax

Dense vector: Word2Vec – Skip-gram (cont.)

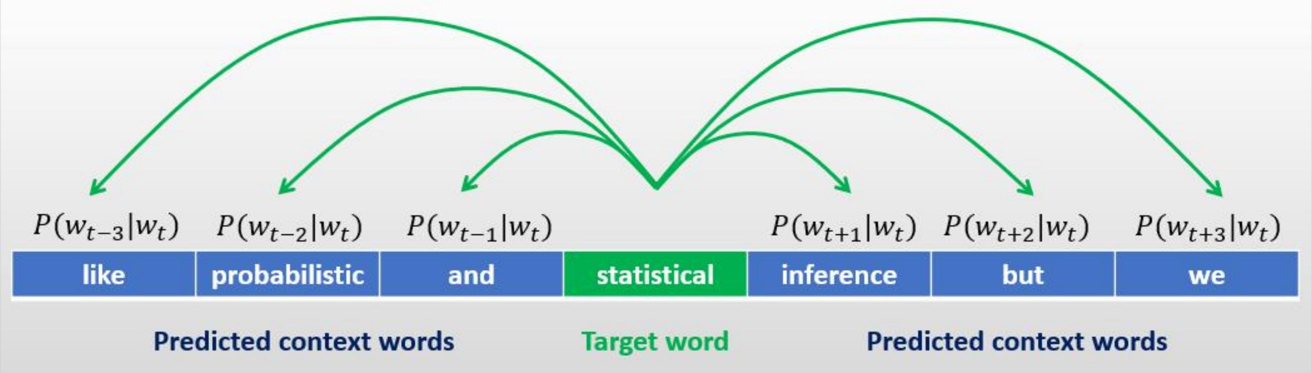
- Negative Sampling
  - We assume that the dataset is noisy containing
    - Positive examples: correct output words
    - Negative examples: incorrect output words



Example

- 6 pairs of positive samples
  - "statistical" and "like" → 1
  - "statistical" and "probabilistic" → 1
  - "statistical" and "and" → 1
  - …
- 6 pairs of negative samples
  - "statistical" and "cat" → 0
  - "statistical" and "dog" → 0
  - "statistical" and "bird" → 0
  - …

Sampling



Dense vector: Word2Vec – Skip-gram (cont.)

- The objective function for skip-gram with negative sampling:

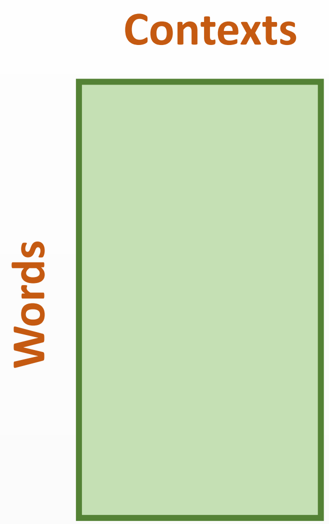
$$J(\theta) = \boxed{\log \sigma(u_0^T v_c)} + \boxed{\sum_{i=1}^{k} \mathrm{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]}$$

Context word
(Positive, +1)

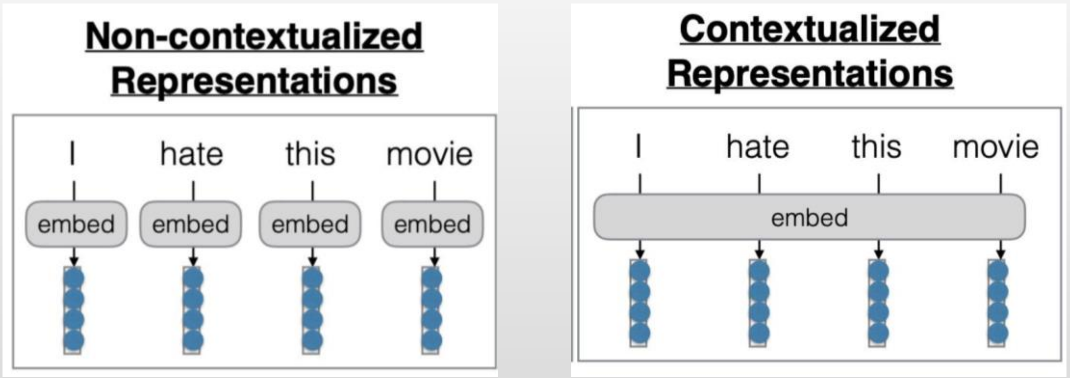
Negative samples
(Negative, -1)

Sigmoid function: $\sigma = \frac{1}{1+e^{-x}}$

Pre-trained Word2Vec



- 1) Non-contextualized Word Embedding (fixed vector)
  - ❖GloVe (Stanford)
    - o https://nlp.stanford.edu/projects/glove/
  - ❖fastText [Available in Thai language] (Facebook)
    - o https://github.com/facebookresearch/fastText
  - ❖TLTK (Aj. Wirote)
    - o https://pypi.org/project/tltk/
- 2) Contextualized Word Embedding
  - ❖thai2fit: ULMFit
    - o https://github.com/cstorm125/thai2fit/
  - ❖BERT [Available in Thai language] (Google)
    - o https://github.com/google-research/bert



http://phontron.com/class/nn4nlp2020/assets/slides/nn4nlp-08-wordemb.pdf

Pre-trained Word2Vec: GloVe

- GloVe: Global Vector for Word Representation
  - GloVe is an unsupervised learning algorithm.
  - Training is performed on **aggregated global word-word co-occurrence** from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space
  - Domains: **Wikipedia, Gigaword, Common Crawl, Twitter**
  - **822MB:** 50,100,200,300 - dimensional
  - **Not available in Thai language**

Pre-trained Word2Vec: GloVe (cont.)

- Main ideas
  - Capture meaning in vector space
  - Takes advantage of co-occurrence statistics instead of only local information

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

$$\boxed{f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & if\ x < x_{max} \\ 1 & otherwise \end{cases}}$$

$$F(w_i, w_j, \widetilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

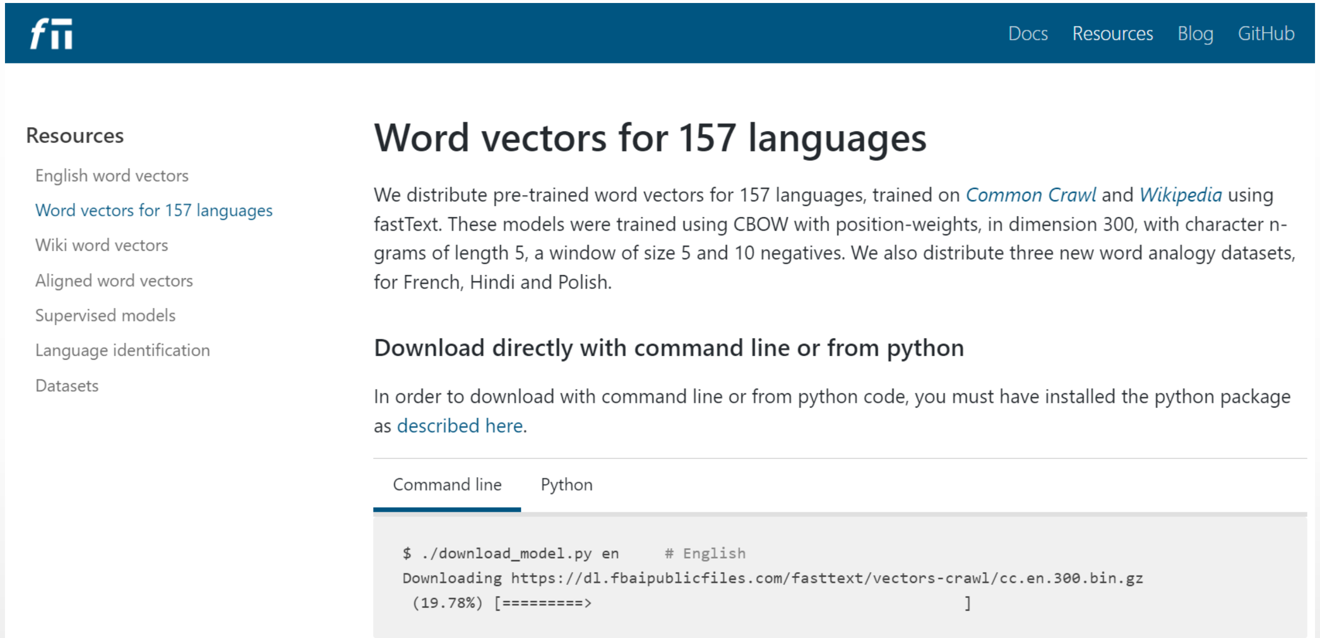
Word co-occurrence matrix

i = ice, j = steam

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(k\|ice)}{P(k\|steam)}$	8.9	8.5×10^{-2}	1.36	0.96

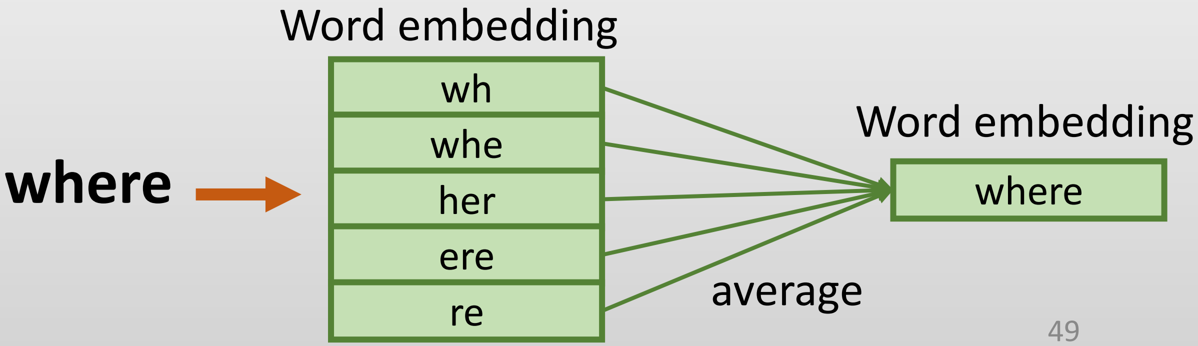
Pre-trained Word2Vec: fastText

- CBOW of "sum of words (char n-grams)"
- Character n-grams as additional features to capture some partial information about local word order.
- Pre-trained for 294 languages (included Thai) trained on Wikipedia



https://fasttext.cc/docs/en/crawl-vectors.html

Do tri-grams on character level
<where> → < wh, whe, her, ere, re >



Pre-trained Word2Vec: Thai Word2Vec (TLTK)

Wirote Aroonmanakun
Nov 21, 2018 · 4 min read

เราเรียนรู้อะไรจาก word2vec

word2vec เป็นการแปลงคำให้อยู่ในรูปเว็กเตอร์ที่สามารถนำไปใช้กับคอมพิวเตอร์เพื่อการประมวลผลภาษาต่อไปได้ง่ายขึ้น คำถามสำคัญ คือข้อมูลที่ได้จากการแปลงคำเป็นเว็กเตอร์ด้วยวิธีการของ Mikolov et al. (2013) นั้นให้ข้อมูลอะไรบ้างเกี่ยวกับคำ ในที่นี้ได้ทดลองสร้าง word2vec จากข้อมูล Thai National Corpus v.3 ขนาด 33 ล้านคำ โดยใช้ gensim และติดตั้งไว้ใน TLTK

```
>>> tltk.corpus.similar_words('สวย',score='n',cutoff=0.6,n=10)
['น่ารัก', 'เซ็กซี่', 'หล่อ', 'เท่', 'สะดุดตา', 'เนี้ยบ', 'งาม', 'เก๋', 'สวยงาม', 'สดใส']

>>> tltk.corpus.similar_words('กิน',score='n',cutoff=0.6,n=10)
['รับประทาน', 'ทาน', 'หุง', 'เคี้ยว', 'ดื่ม', 'คลุก', 'กินน้ำ', 'ดอง', 'ต้ม', 'กินที่']
```

Search projects
Help Sponsors Log in Register

tltk 1.6.3

`pip install tltk`

Latest version

Released: Feb 13, 2023

Thai Language Toolkit

Navigation
- Project description
- Release history
- Download files

Project description

TLTK is a Python package for Thai language processing: syllable, word, discouse unit segmentation, pos tagging, named entity recognition, grapheme2phoneme, ipa transcription, romanization, etc. TLTK requires Python 3.4 or higher. The project is a part of open source software developed at Chulalongkorn University. Since version 1.2.2 pacakge license is changed to New BSD License (BSD-3-Clause)

Input : must be utf8 Thai texts.

https://pypi.org/project/tltk/

https://awirote.medium.com/%E0%B9%80%E0%B8%A3%E0%B8%B2%E0%B9%80%E0%B8%A3%E0%B8%B5%E0%B8%A2%E0%B8%99%E0%B8%A3%E0%B8%B9%E0%B9%89%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3%E0%B8%88%E0%B8%B2%E0%B8%81-word2vec-8517862e9a07

Pre-trained Word2Vec: Benefit

Gives better performances

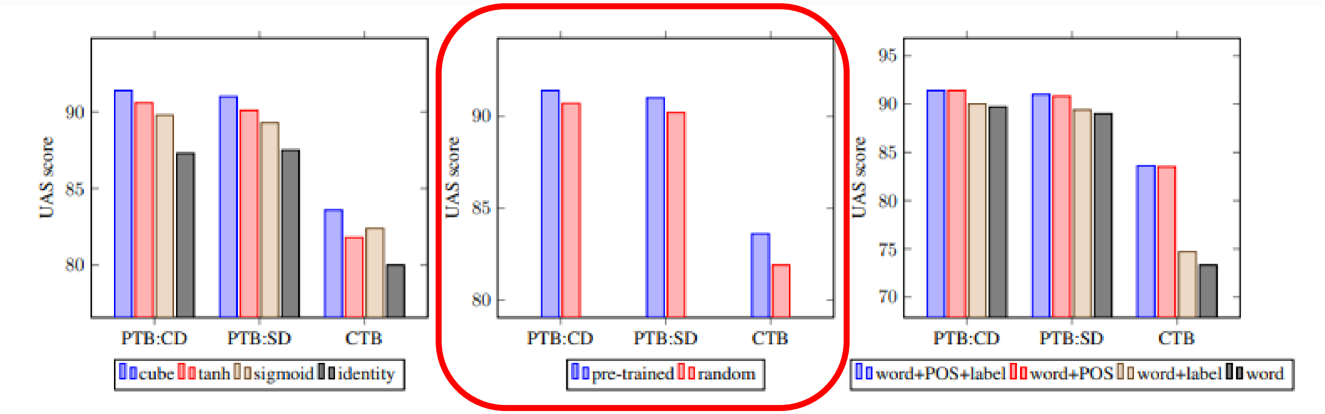


Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

Chen, D., & Manning, C. D. (2014, October). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 740-750).

Learning semantic-syntactic Relationship

- Word embedding can also capture semantic and syntactic relationship between words
- King – Queen = Man – Woman
- We can search for word that is closest to X
- X = King – Man + Woman → X = ???

Word2Vec Evaluation

- **Extrinsic** Evaluation:
  - Use pre-trained word vectors to initialize or concatenate as extra features
  - Then evaluate on real tasks (e.g., PoS, NER, sentimental analysis, etc.)
- **Intrinsic** Evaluation:
  - Evaluate on specific subtasks (e.g., Analogy completion)

City 1 : State containing City 1 : : City 2 : State containing City 2

Input	Result Produced
Chicago : Illinois : : Houston	Texas
Chicago : Illinois : : Philadelphia	Pennsylvania
Chicago : Illinois : : Phoenix	Arizona
Chicago : Illinois : : Dallas	Texas
Chicago : Illinois : : Jacksonville	Florida
Chicago : Illinois : : Indianapolis	Indiana
Chicago : Illinois : : Austin	Texas
Chicago : Illinois : : Detroit	Michigan
Chicago : Illinois : : Memphis	Tennessee
Chicago : Illinois : : Boston	Massachusetts

Word2Vec Evaluation (cont.)

- **Intrinsic** Evaluation:
  - **Relatedness:** Correlation between word vectors similarity and human judgment of word similarity
  - **Analogy:** The goal is to find a term x for a given term y so that x:y best resembles a sample relationship a:b
  - **Categorization**: Word vector are clustered, then measure the purity of cluster based on the labeled dataset

Advanced Topics: OOV problem

- Why do we need a UNK token
  - Not all words are available in training data
  - Large vocab size = more memory and computation time
- Common ways
  - If word count <= 1, then UNK
  - Rank threshold (frequency): only include top 50,000 words, the rest are UNK

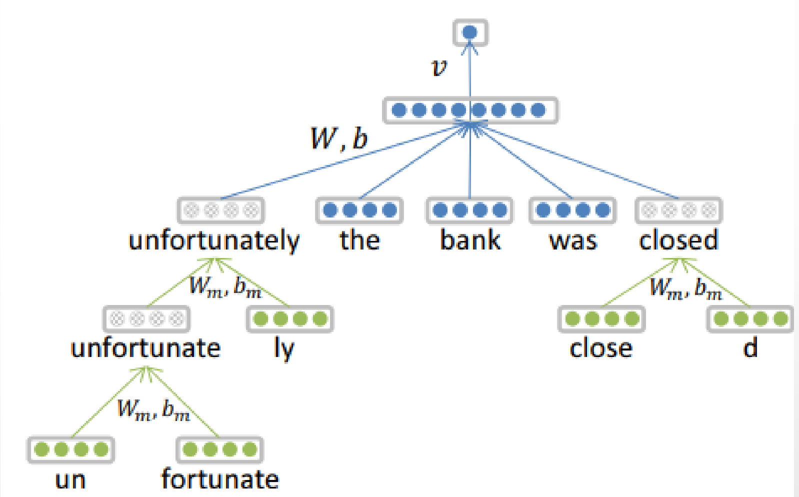
Advanced Topics: Limitation of word embedding

- Sensitive to subtle/superficial morphological difference
  - student **vs** students
  - Solution: Subwords
- One representation for all unknown words
  - Solution: Subwords
- Insensitive to context
  - Baseball bat **vs** bat can fly
  - Solution: Contextualized Word Embedding
- Interpretability
  - What does each dimension of a word embedding represent?
  - Solution: Sparse Embedding
- Bias
  - Male vs Female vs Engineer
  - Solution: debias algorithm

Fixed table

Advanced Topics: Subword Embedding

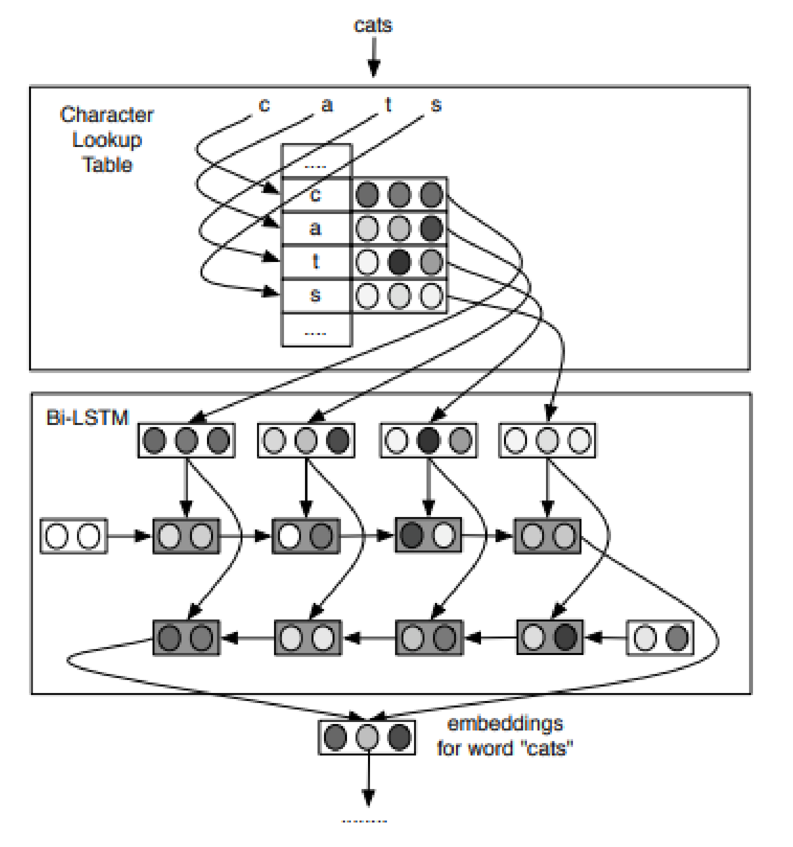
- Morpheme-based [Luong et al.2013] https://aclanthology.org/W13-3512.pdf



- Character n-grams[Wieting et al.2013] https://aclanthology.org/D16-1157.pdf

Do tri-grams on character level
<where> → < wh, whe, her, ere, re >

- Character-based [Ling et al.2015] https://aclanthology.org/D15-1176/



Advanced Topics: Subword Embedding (cont.)

- Bye-Pair Encoding (BPE)
  - BPE was introduced in neural machine translation of Rare Words with Subword Units [Sennrich etal.,2015]
  - Used in Roberta, GPT-2
  - Choose from the pairs with the most frequency.

Example

word = "aaabdaaabac"

- ZabdZabac
  - Z = aa

Frequency of "aa" = 2

- ZYdZYac
  - Z = aa
  - Y = ab

Frequency of "ab" = 2

- XdXac
  - X = ZY
  - Z = aa
  - Y = ab

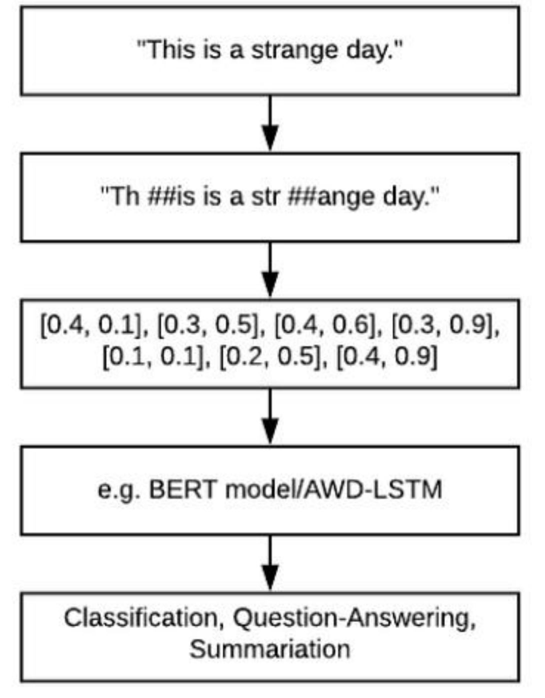
Frequency of "ZY" = 2

Advanced Topics: Subword Embedding (cont.)

- **WordPiece** (Schuster et.al, 2012) is **very similar to BPE**: used for BERT
- Add n-grams that maximally reduces **<u>perplexity (maximize the likelihood)</u>**
- There are 2 types of tokens: start token(no ##), and continuing token (##)

Example: "ug"

- Calculate $\frac{P(\text{"u","g"})}{P(\text{"u"})P(\text{"g"})}$
- If "ug" is the maximum likelihood
- Add to embedding table



https://jacky2wong.medium.com/understanding-sentencepiece-under-standing-sentence-piece-ac8da59f6b08

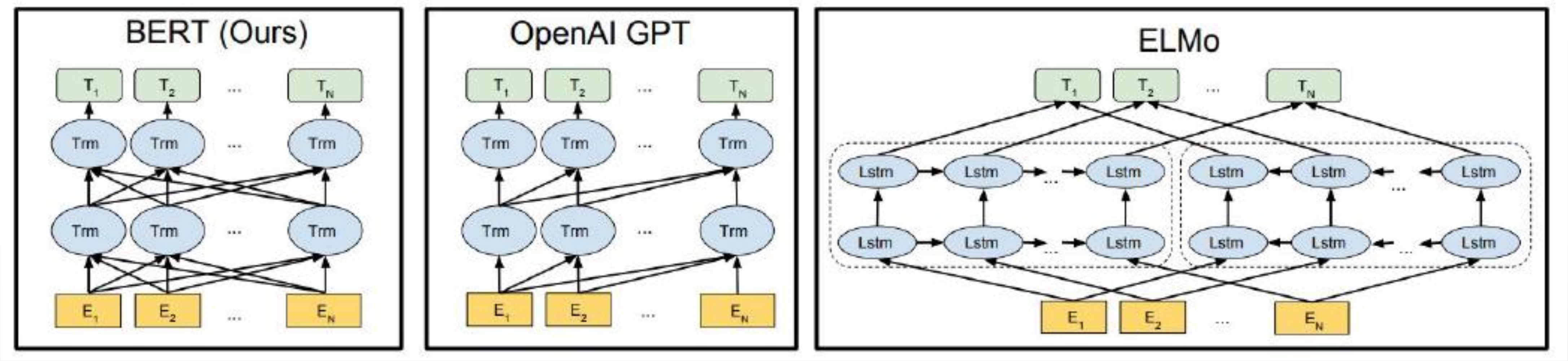
Advanced Topics: Subword Embedding (cont.)

- **SentencePiece** (Kudo et.al, 2018)
  - Aims to solve 2 issues
    - **Issue 1:** Which one should a correct denormalization?
      - Example: “Word.” or “Word .”
    - **Issue 2:** End-to-end to avoid the need of language-specific tokenization → Thai language → no space
  - WordPiece tokenizes inside words
  - **SentencePiece tokenizes inside sentences (across words)**
  - **Avoid space by special token (_)**

WangchanBERTa We name our pretrained language models according to their architectures, tokenizers and the datasets on which they are trained on. The models can be found on HuggingFace[12].

	Architecture	Dataset	Tokenizer
wangchanberta-base-wiki-spm	RoBERTa-base	Wikipedia-only	SentencePiece
wangchanberta-base-wiki-newmm	RoBERTa-base	Wikipedia-only	word (newmm)
wangchanberta-base-wiki-ssg	RoBERTa-base	Wikipedia-only	syllable (ssg)
wangchanberta-base-wiki-sefr	RoBERTa-base	Wikipedia-only	SEFR
wangchanberta-base-att-spm-uncased	RoBERTa-base	Assorted Thai Texts	SentencePiece

Advanced Topics: Contextualized Word Embedding



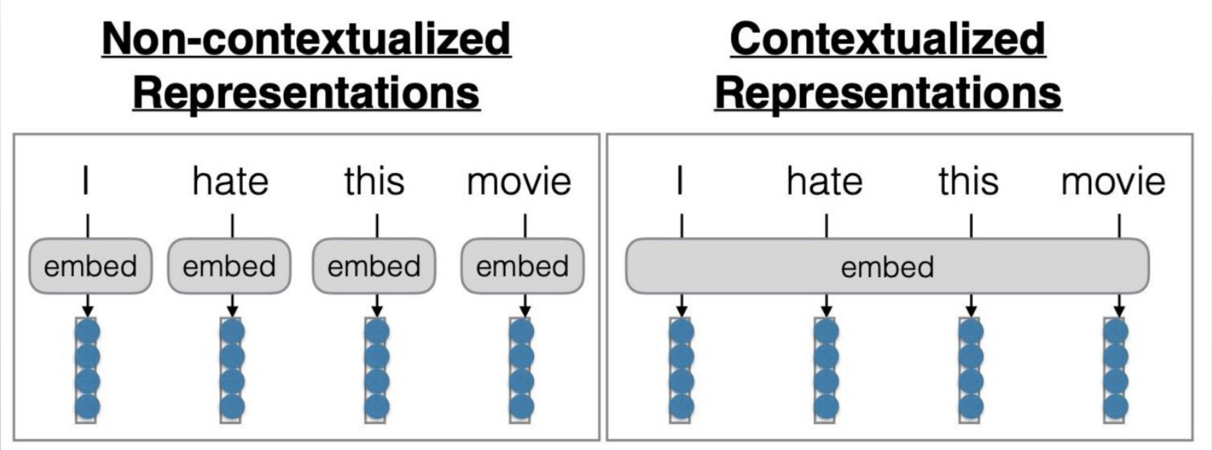
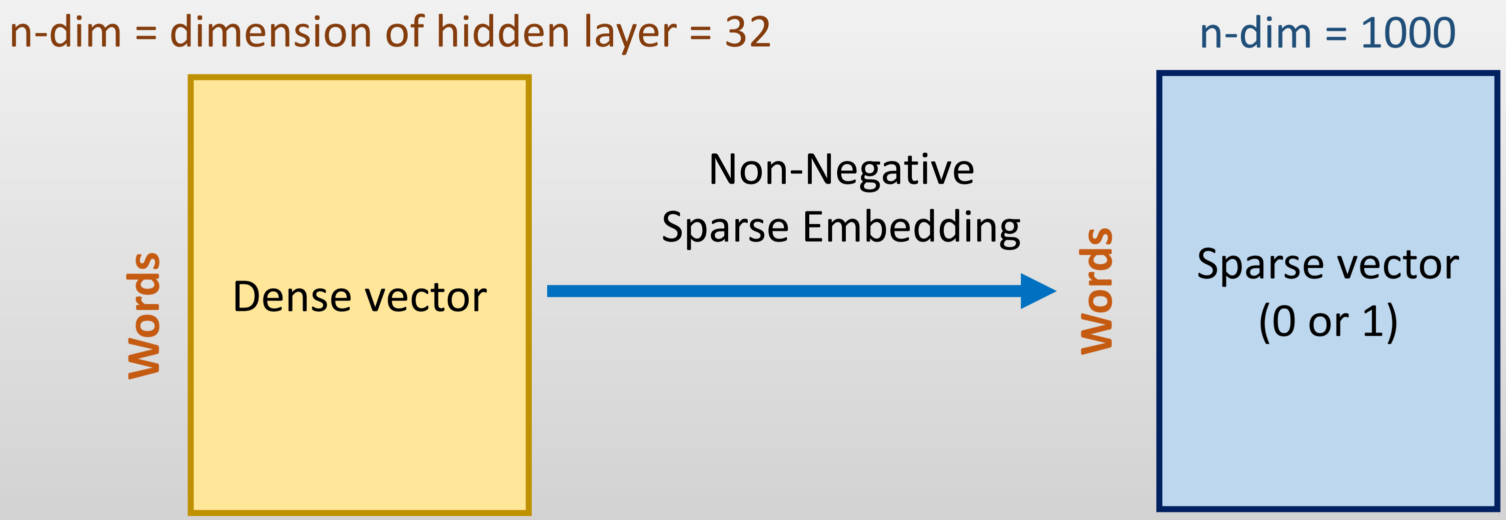


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations are jointly conditioned on both left and right context in all layers.

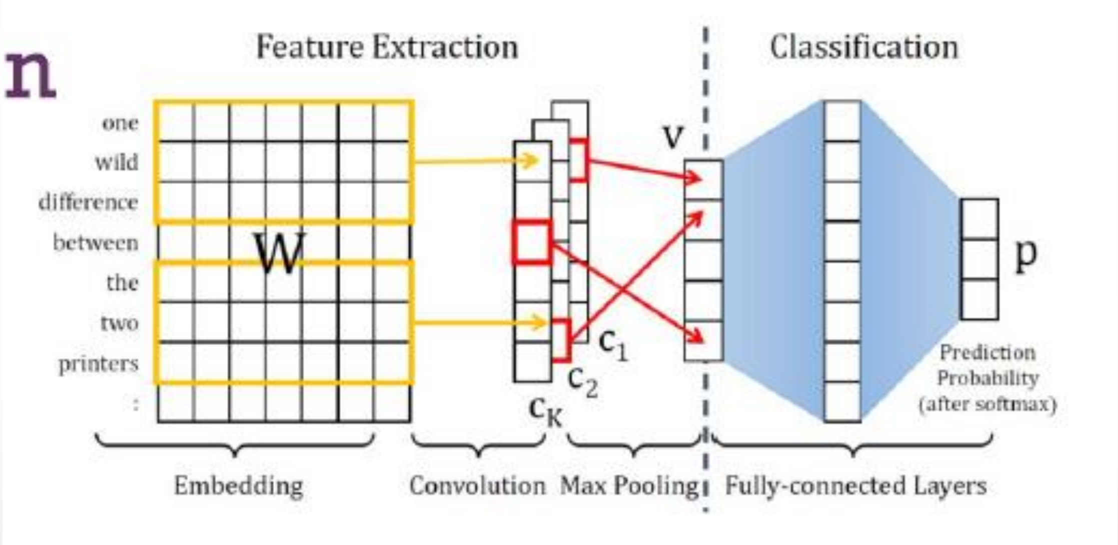
Advanced Topics: Sparse Embedding

- Murphy et al. 2012
- Increase the interpretability of each dimension
  - Non-Negative Sparse Embedding (NNSE)
  - Add sparsity
    - Most of the dimensions are zeros

Model	Top 5 Words (per dimension)
SVD_{300}	well, long, if, year, watch plan, engine, e, rock, very get, no, features, music, via features, by, links, free, down works, sound, video, building, section
$NNSE_{1000}$	inhibitor, inhibitors, antagonists, receptors, inhibition bristol, thames, southampton, brighton, poole delhi, india, bombay, chennai, madras pundits, forecasters, proponents, commentators, observers nosy, averse, leery, unsympathetic, snotty



Advanced Topics: Gradient Analysis



- Sensitivity analysis (Arras et al.,2016)
  - The effect of w_i towards the prediction of class j

$$E_{j,w_i} = \sum_{d} \left(\frac{\partial FC(v)_j}{\partial w_{i,d}} \right)^2$$

Demo: Word Representation

https://drive.google.com/file/d/1SmOnqcG18Y3t9rQIEV9xU2GyLd8WjFpG/view?usp=share_link

HW2

https://drive.google.com/drive/folders/1n_ZokWPuNbti_4qlA13-3Rri50O_KuT4?usp=share_link