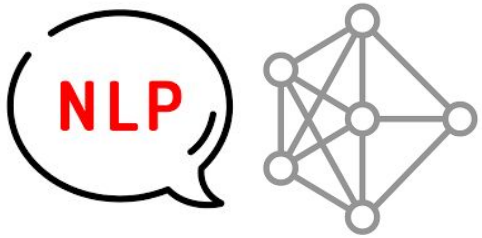
CHULA ENGINEERING
Foundation toward Innovation

COMPUTER



Word Representations

2110572: Natural Language Processing Systems

Assoc. Prof.Peerapon Vateekul, Ph.D.
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

Credit: Can Udomcharoenchaikit, Kasidis Kanwatchara
Last updated on 2023/01/26

Outline

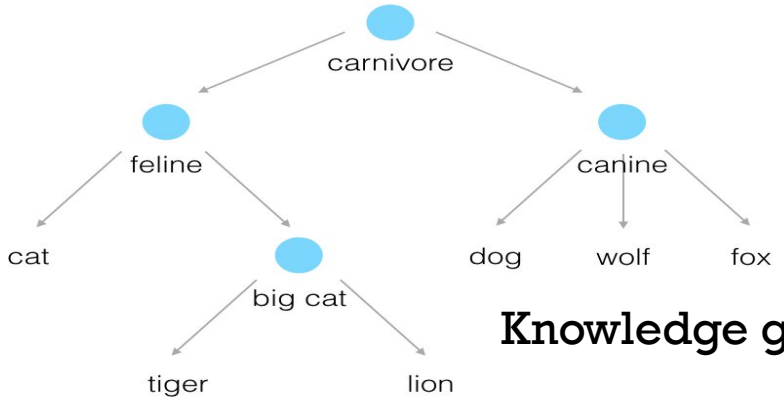
- **1) How to represent words?**
  - Symbolic vs distributional word representations
- **2) Distributional: Sparse vector** representations (discrete representation)
  - Term-document matrix
  - Co-occurrence matrix
  - Positive Pointwise Mutual Information (PPMI)
  - TF-IDF
- **3) Distributional: Dense vector representations**
  - SVD-based method
  - Word2Vec
    - Skip-gram
    - CBOW
  - Word2Vec training methods
  - Pre-trained vector representations
    - Adaptation
  - Leaned Semantic-Syntactic Relationships
  - Compositionality
- **4) Word2Vec Evaluation**
- **5) Advanced Topics**

Part 1: How to represent words?

Symbolic vs distributional word representations

How to represent words?
Symbolic vs Distributional representations

- Representing words in computer is one of the most important tasks in NLP.
- Words = Input for our models

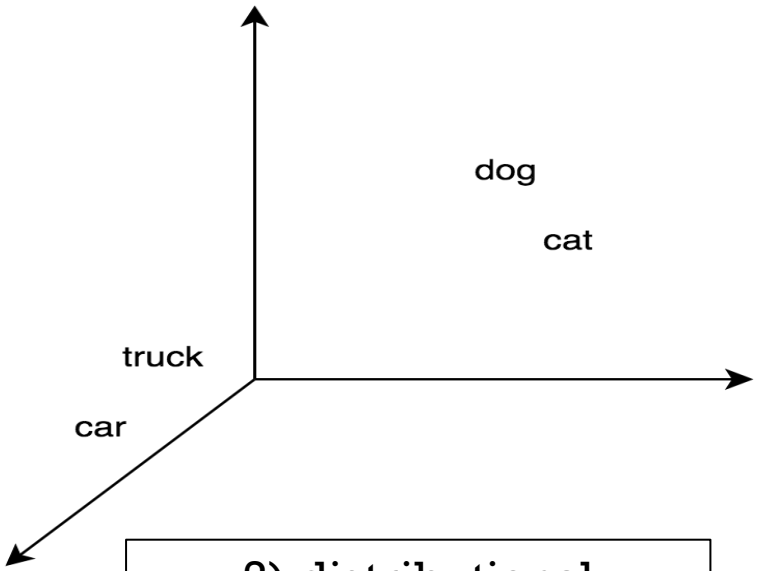


Knowledge graph

เสือ = [0 1 0 0 0 ... 0 0]

One-hot model

1) symbolic

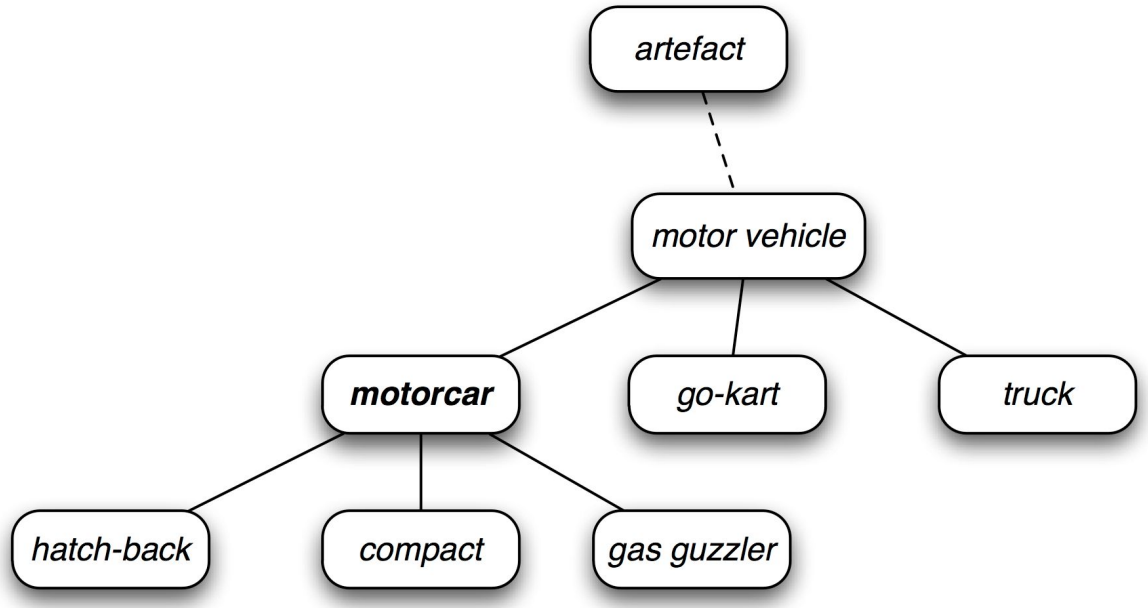


2) distributional

How to represent words?
Symbolic vs Distributional representations (cont.)

1

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



Knowledge graph

How to represent words?
Symbolic vs Distributional representations (cont.)

1

- Symbolic Representations
  - Earlier work in NLP, the vast majority of (rule-based and statistical) NLP models considered words as discrete atomic symbols.
  - E.g. One-hot model

    เสือ = [0 1 0 0 0 … 0 0]

    สัตว์กินเนื้อ = [0 0 0 0 1 … 0 0]

    * Each point in the vector represents each vocab.

  - Drawback:
    - Cannot capture similarity between words

How to represent words?
Symbolic vs Distributional representations (cont.)

2

- Distributed representations (aka distributional methods)
  - "You shall know a word by the company it keeps" (J. R. Firth 1957)
  - The meaning of a word is computed from the distribution of words around it.
  - **Can encode similarity between words!**

การลดจำนวนลงของเสือซึ่งเป็นสัตว์กินเนื้อ จะส่งผล
กระทบต่อโครงสร้างและระบบนิเวศทั้งหมด

เสือ เป็นสัตว์เลี้ยงลูกด้วยนมในวงศ์ฟิลิดีซึ่งเป็นวงศ์เดียวกับแมว

How to represent words?
Symbolic vs Distributional representations (cont.)

- Most modern NLP models use distributional word representations to represent words
- Word meaning as a vector

- In this class, we will examine the development of distributed word representations **before** the rise of deep learning, then we will introduce you to word representation techniques used in deep learning models.

1) Sparse Representation
2) Dense Representation

Part 2: Distributional: Sparse vector

Term-document matrix

Co-occurrence matrix

Positive Pointwise Mutual Information (PPMI)

TF-IDF

Sparse vector representations in Distributional Representations

1. Term-Frequency (Raw Frequency)
2. Co-occurrence (Raw Frequency)
3. PPMI
4. TF-IDF

1 Sparse vector representations: term-document matrix

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

vocabulary

	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

document

Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

Sparse vector representations: term-document matrix (cont.)

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

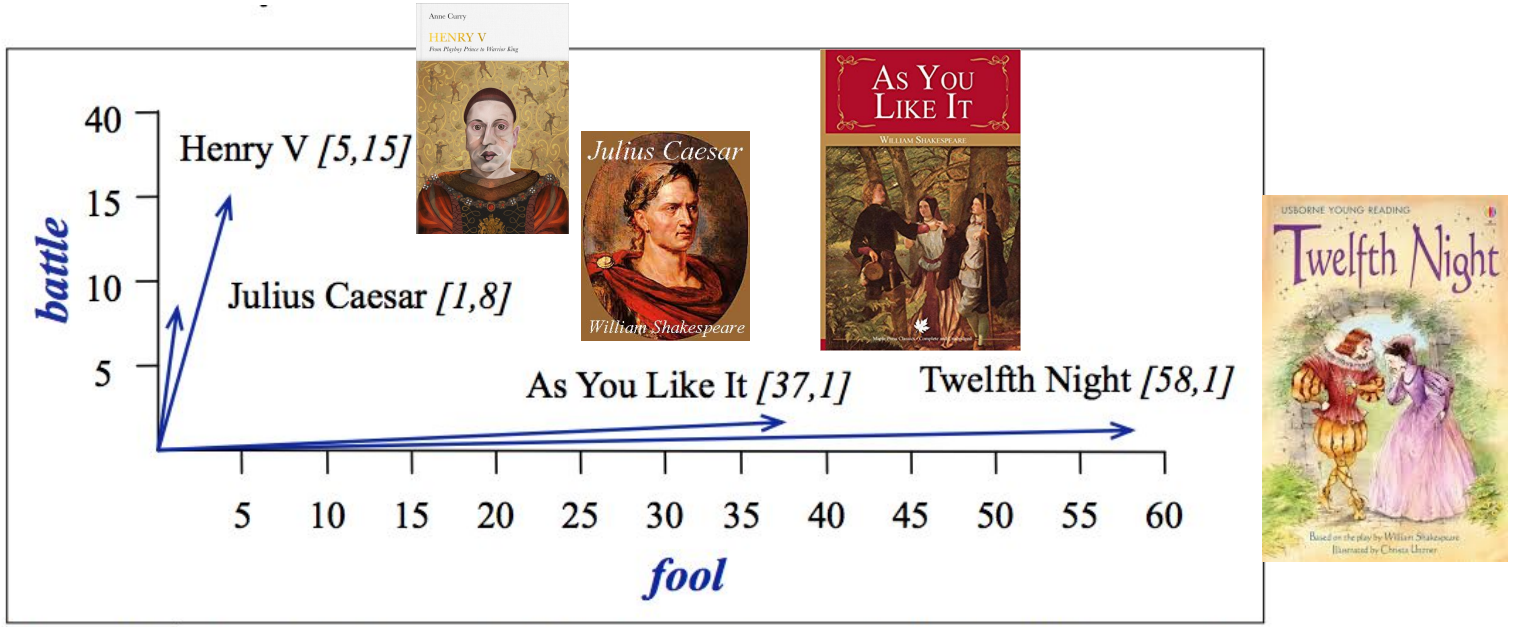


Figure 15.3 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

Sparse vector representations: term-document matrix (cont.)

- Documents as vectors
- Two documents are similar if their vectors are similar (document similarity)

vocabulary

	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

document

Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

Sparse vector representations: term-document matrix (cont.)

- Words as vectors
  - Two words are similar if their vectors are similar (word similarity)

vocabulary

	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

document

Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

2 Sparse vector representations: co-occurrence matrix (1)

- Word-word or word-context matrix
  - Instead of entire documents, use smaller contexts
- Two words are similar if their vectors are similar

vocabulary →

window = 4

	aardvark	...	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	

Figure 15.4 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

Sparse vector representations: co-occurrence matrix (2)

- Two words are similar if their vectors are similar

window = 4

vocabulary →	aardvark	...	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	

Figure 15.4 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

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

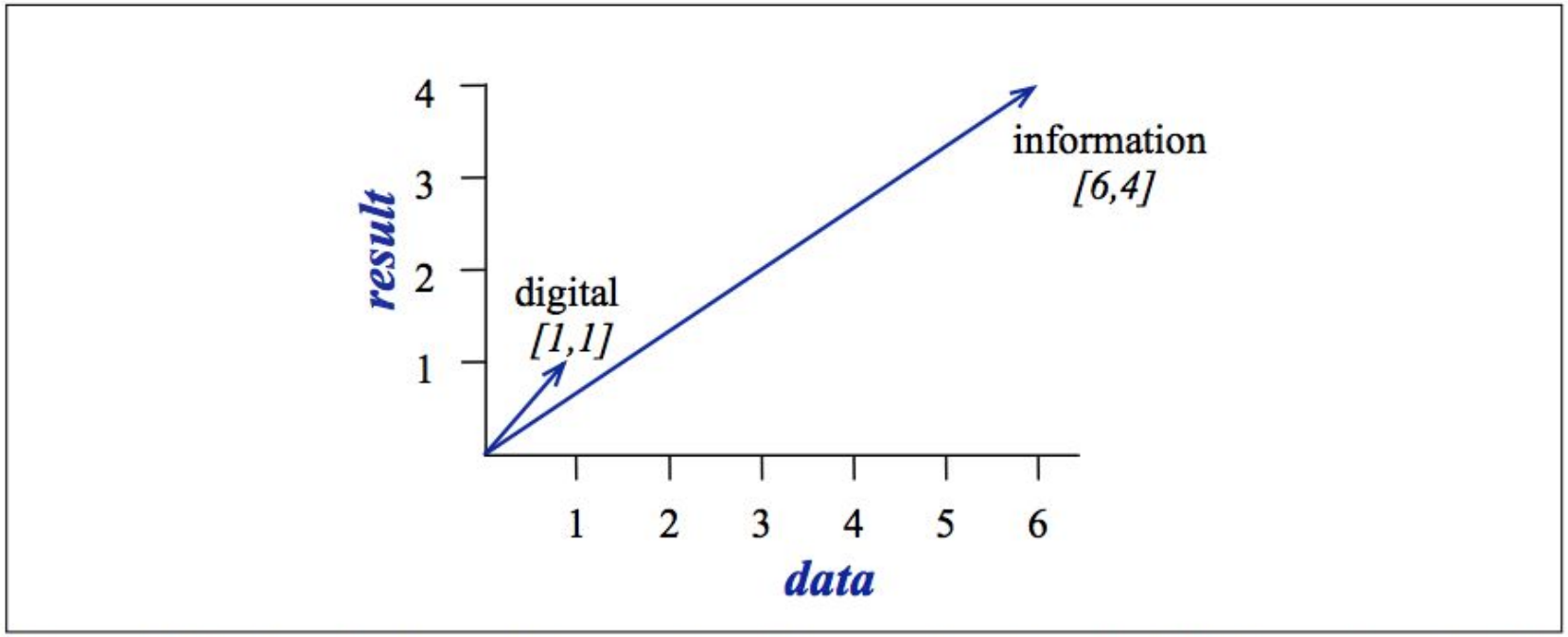


Figure 15.5 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *result*.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

3 Sparse vector representations: Positive Pointwise Mutual Information (PPMI)

- Two words are similar if their vectors are similar

window = 4

vocabulary →	aardvark	...	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	

Figure 15.4 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

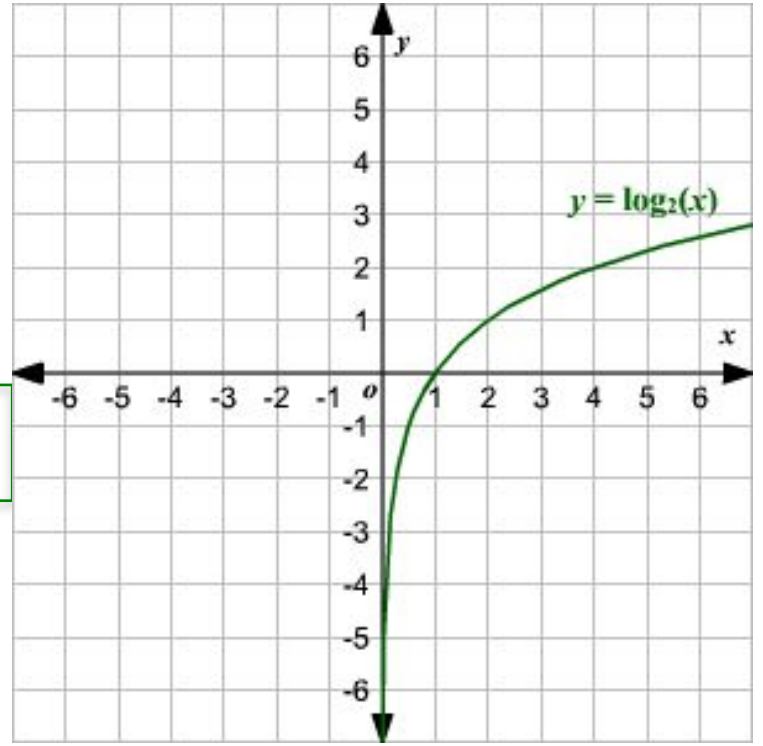
- **Problems with raw frequency**
  - Not very discriminative (need normalization)
    - Words such as "it, the, they, a, an, the" occur very frequently
    - , but are not very informative
- PPMI incorporates the idea of mutual information to determine the informative context words
  - We need a measure which tells us which context words are informative about the target word

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

How often the two words occur together

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

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



w – target word
c – context word
Do words "w" and "c" co-occur more than if they were independent?

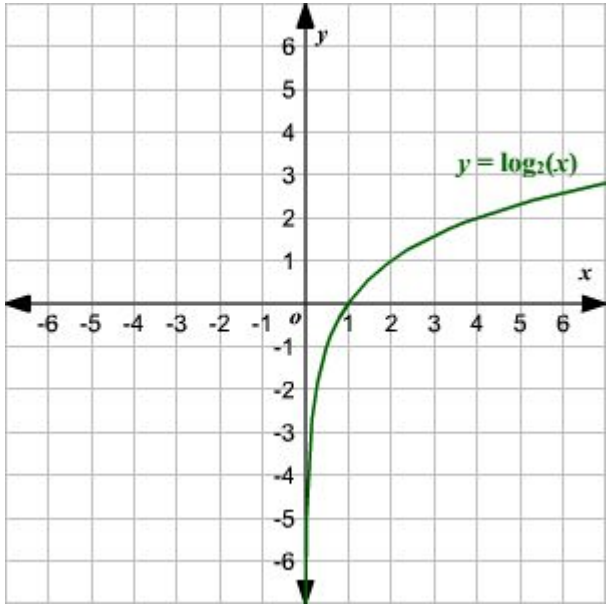
+ : occur together > occur by chance
0 : occur together = occur by chance
- : occur together < occur by chance

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

Negative PMI values tend to be unreliable.

It is common to replace all negative PMI values with zero

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



vocabulary

word co-occurrence matrix

	aardvark	...	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	

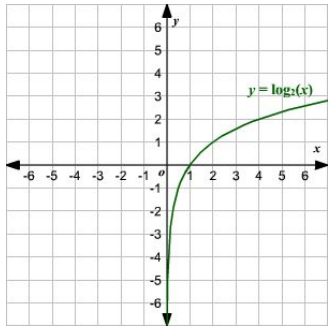
PPMI matrix

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

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

Figure 15.7 The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 15.4 again showing five dimensions. Note that the 0 ppmi values are ones that had a negative pmi; for example pmi(*information*,*computer*) = $\log 2(.05/(.16 * .58)) = -0.618$, meaning that *information* and *computer* co-occur in this mini-corpus slightly less often than we would expect by chance, and with ppmi we replace negative values by zero. Many of the zero ppmi values had a pmi of $-\infty$, like pmi(*apricot*,*computer*) = $\log 2(0/(0.16 * 0.11)) = \log 2(0) = -\infty$.



Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

4 Sparse vector representations: TF-IDF

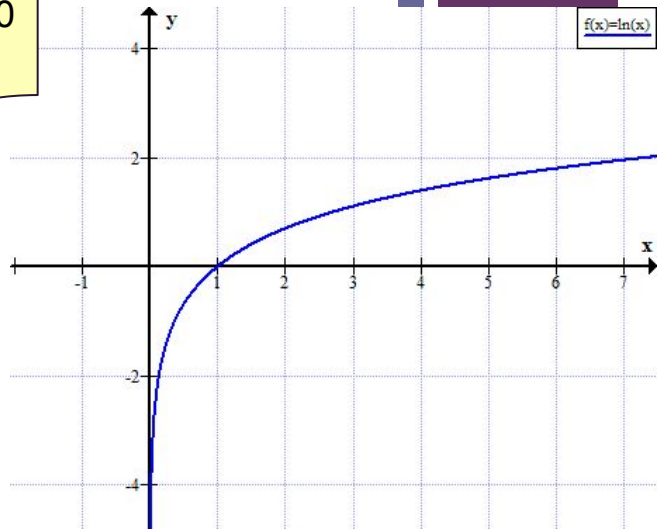
Need for normalization in TF

Doc1
cat = 5/10

Doc2
cat = 50/1000

- Term Frequency (TF) – per each document

$$TF(w) = \frac{\text{Frequency of word } w \text{ in a document}}{\text{Total number of words in the document}}$$



- Inverse Document Frequency (IDF) – per corpus (all documents)

$$IDF(w) = \log_e\left(\frac{\text{Total number of documents}}{\text{Number of documents that contain word } w}\right)$$

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

- TF-IDF

$$TFIDF(w) = TF(w) * IDF(w)$$

Term Frequency (TF)
1) Original TF

document 1

The sky is blue. The sky is beautiful

f("*The*", *document1*) → 2
f("*sky*", *document1*) → 2
f("*is*", *document1*) → 2
f("*blue*", *document1*) → 1
f("*beautiful*", *document1*) → 1

ตัวอย่างนับ term *ใน* document

Term Frequency

หลังจากนั้นนำค่าที่ได้แต่ละคำ ไปหารกับจำนวนคำทั้งหมดใน document ใน ตัวอย่างมีทั้งหมด 8 คำ

$$tf(term,document) = \frac{f(term,document)}{\sum_{term' \in document} f(term', document)}$$

function *สำหรับคำนวณ* term-frequency

จะได้ว่า

tf("*The*", *document1*) → 2/8 = 0.25
tf("*sky*", *document1*) → 2/8 = 0.25
tf("*is*", *document1*) → 2/8 = 0.25
tf("*blue*", *document1*) → 1/8 = 0.125
tf("*beautiful*", *document1*) → 1/8 = 0.125

ตัวอย่างคำนวณ term frequency *แบบ* raw counts

https://lukkiddd.com/tf-idf-%E0%B8%84%E0%B8%B3%E0%B9%84%E0%B8%AB%E0%B8%99%E0%B8%AA%E0%B8%B3%E0%B8%84%E0%B8%B1%E0%B8%8D%E0%B8%99%E0%B8%B0-dd1e1568312e

Term Frequency (TF)
2) Log Normalization

document 1

The sky is blue. The sky is beautiful

f("The", document1) → 2
f("sky", document1) → 2
f("is", document1) → 2
f("blue", document1) → 1
f("beautiful", document1) → 1

ตัวอย่างนับ term *ใน* document

Log Normalization

อีกรูปแบบนึงในการหาค่า Term Frequency คือการหาค่าความถี่แบบ log scaled

> *เหมาะสำหรับข้อมูลที่จำนวนความถี่ของคำมีช่วงต่างกันมากๆ เช่น มีคำนึงมีความถี่แค่ 1 แต่อีกคำมีความถี่เป็น 1000*

Log Normalization จะมาช่วย normalized ค่าไม่ให้แตกต่างกันเกินไป

$$\log Normalization(term, document) = 1 + \log(f(term, document))$$

function *สำหรับคำนวณ* term frequency *แบบ* log normalization

กำหนดให้ logTF = logNormalization

logTF("The", document1) → 1 + log(2) ≈ 1.3
logTF("sky", document1) → 1 + log(2) ≈ 1.3
logTF("is", document1) → 1 + log(2) ≈ 1.3
logTF("blue", document1) → 1 + log(1) = 1
logTF("beautiful", document1) → 1 + log(1) = 1

ตัวอย่างคำนวณ term frequency *แบบ* log normalization

https://lukkiddd.com/tf-idf-%E0%B8%84%E0%B8%B3%E0%B9%84%E0%B8%AB%E0%B8%99%E0%B8%AA%E0%B8%B3%E0%B8%84%E0%B8%B1%E0%B8%8D%E0%B8%99%E0%B8%B0-dd1e1568312e

Term Frequency (TF)
3) Double Normalization

document 1

The sky is blue. The sky is beautiful

f("The", document1) → 2
f("sky", document1) → 2
f("is", document1) → 2
f("blue", document1) → 1
f("beautiful", document1) → 1

ตัวอย่างนับ term ใน document

Double Normalization

> เหมาะสำหรับข้อมูลที่มีความยาวของ เอกสาร (document) ไม่เท่ากัน หรือต่างกันมาก เช่น
> document a มีแค่ 10 คำ แต่ document b มี 1000 คำ

K = 0.5

Double Normalization จะมาช่วยป้องกันไม่ให้เกิด bias เนื่องจากความยาวของเอกสาร (document) มันมีผลต่อการคำนวณ

$$doubleNormalization(term, document, K) = K + (1\text{-}K)\frac{f(term, document)}{\max(f(term, document))}$$

function สำหรับคำนวณ term frequency แบบ double normalization

กำหนดให้ dn = doubleNormalization

- dn("the", document1, 0.5) = 0.5 + 0.5*(2/2) = 1
- dn("sky", document1, 0.5) = 0.5 + 0.5*(2/2) = 1
- dn("is", document1, 0.5) = 0.5 + 0.5*(2/2) = 1
- dn("blue", document1, 0.5) = 0.5 + 0.5*(1/2) = 0.75
- dn("beautiful", document1, 0.5) = 0.5 + 0.5*(1/2) = 0.75

ตัวอย่างคำนวณ term frequency แบบ double normalization

https://lukkiddd.com/tf-idf-%E0%B8%84%E0%B8%B3%E0%B9%84%E0%B8%AB%E0%B8%99%E0%B8%AA%E0%B8%B3%E0%B8%84%E0%B8%B1%E0%B8%8D%E0%B8%99%E0%B8%B0-dd1e1568312e

+ Term Frequency (TF)
3) Double Normalization (cont.)

Maximum tf normalization

One well-studied technique is to normalize the tf weights of all terms occurring in a document by the maximum tf in that document. For each document d, let $\mathrm{tf}_{\max}(d) = \max_{\tau \in d} \mathrm{tf}_{\tau,d}$, where τ ranges over all terms in d. Then, we compute a normalized term frequency for each term t in document d by

$$\mathrm{ntf}_{t,d} = a + (1-a)\frac{\mathrm{tf}_{t,d}}{\mathrm{tf}_{\max}(d)}, \tag{30}$$

where a is a value between 0 and 1 and is generally set to 0.4, although some early work used the value 0.5. The term a in (30) is a *smoothing* term whose role is to damp the contribution of the second term - which may be viewed as a scaling down of tf by the largest tf value in d. We will encounter smoothing further in Chapter 13 when discussing classification; the basic idea is to avoid a large swing in $\mathrm{ntf}_{t,d}$ from modest changes in $\mathrm{tf}_{t,d}$ (say from 1 to 2). The main idea of maximum tf normalization is to mitigate the following anomaly: we observe higher term frequencies in longer documents, merely because longer documents tend to repeat the same words over and over again. To appreciate this, consider the following extreme example: supposed we were to take a document d and create a new document d' by simply appending a copy of d to itself. While d' should be no more relevant to any query than d is, the use of (23) would assign it twice as high a score as d. Replacing $\text{tf-idf}_{t,d}$ in (23) by $\text{ntf-idf}_{t,d}$ eliminates the anomaly in this example.

https://nlp.stanford.edu/IR-book/html/htmledition/maximum-tf-normalization-1.html?fbclid=IwAR2ainbmXLYONF-qUY1DYqA9Lh8oMsSHNiC_8gy33OPNUSIUNid98YQzYiE

Sparse vector representations: 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	0.018	0.022

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20+1) = 1.322$ and idf = .037. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/6.pdf, Feb 2020

Sparse vector representations: TF-IDF (Cont.)

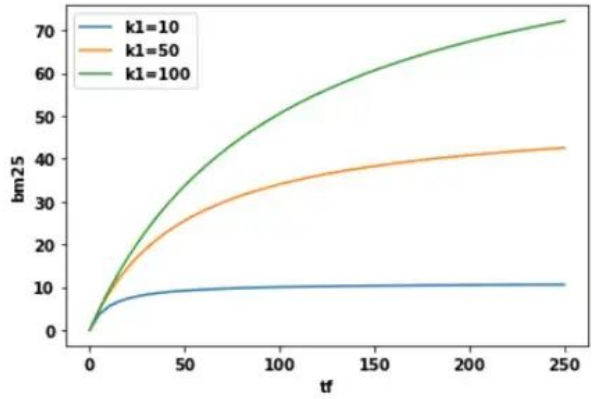
- A very popular way of weighting in Information Retrieval (document similarity)
- Not commonly used as a component to measure **word** similarity (it is designed for **document** similarity)

BM25

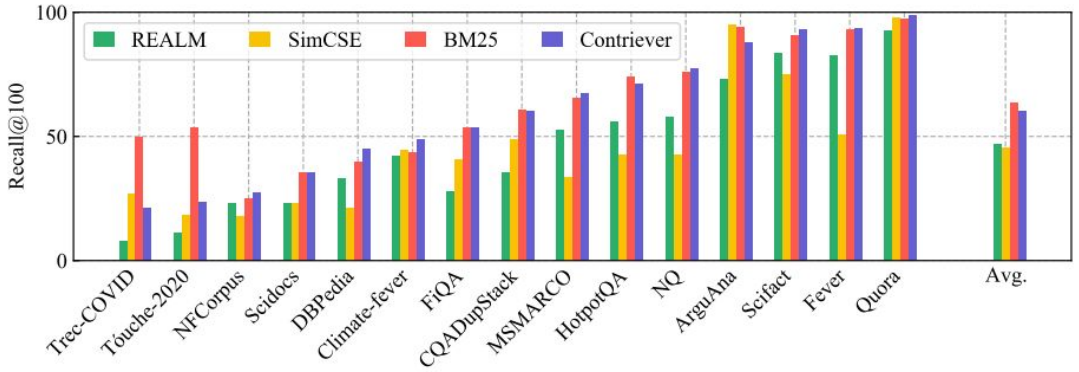
$$BM25 = \sum_{t \in q} \log \left[\frac{N}{df(t)} \right] \cdot \frac{(k_1 + 1) \cdot tf(t,d)}{k_1 \cdot \left[(1 - b) + b \cdot \frac{dl(d)}{dl_{avg}} \right] + tf(t,d)}$$

- k_1, b – parameters
- $dl(d)$ – length of document d
- dl_{avg} – average document length

k_1 – Term Saturation and diminishing return

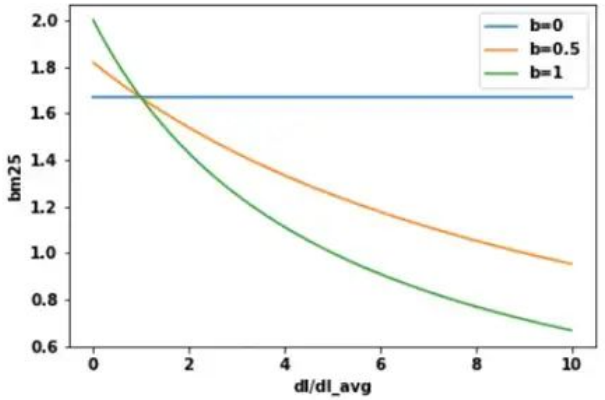


If a document contains 100 occurrences of the term "computer," is it really twice as relevant as a document that contains 50 occurrences?



Still a very strong baseline and is relevant in today's research. Fig. from Meta AI's Contriver (Izacard et al.,2022)

b – Document Length Normalization



If a document is very short and it contains "computer" once, that might already be a good indicator of relevance. But if the document is really long and the term "computer" only appears once, it is likely that the document is not about computers.

https://towardsdatascience.com/understanding-term-based-retrieval-methods-in-information-retrieval-2be5eb3dde9f

Part 3: Distributional: Dense vector representations

- SVD-based method
- Word2Vec
  - Skip-gram
  - CBOW
- Word2Vec training methods
- Pre-trained vector representations
  - Adaptation
- Leaned Semantic-Syntactic Relationships
- Compositionality

Dense vector representations

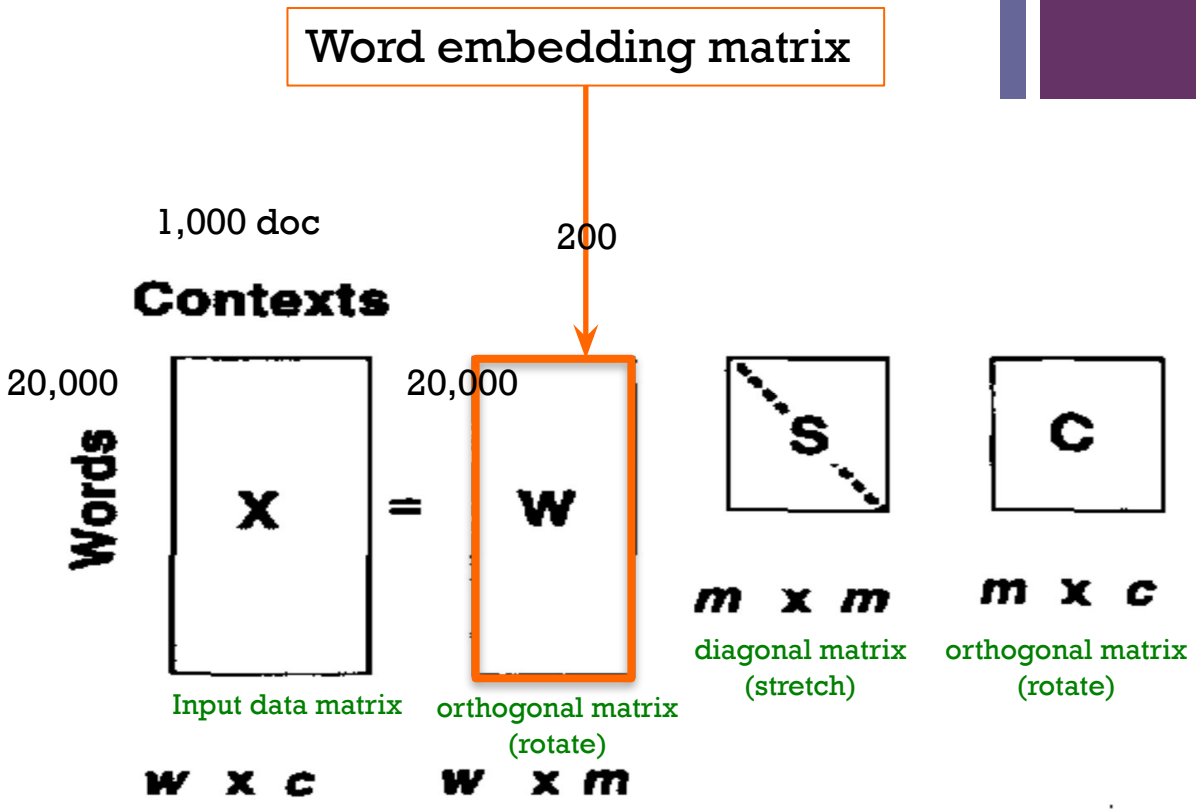
- Sparse vector representations such as PPMI vectors are:
  - Long (length of vector ≈ 20,000 to 50,000)
  - Sparse (most elements are zero)
- Dense vector representations are introduced to:
  - Reduce length of vectors (length of vector ≈ 200 to 1,000)
  - Reduce sparsity (hence the name "dense"; most elements are not zero)
  - **"Dense vectors work better in every NLP task than sparse vectors." - Dan Jurafsky**

Dense vector representations (cont.)

- **Advantages** of dense vector representation
  - Less parameters to tune
  - Generalize better
  - Better at capturing synonymy

1 Dense vector representations: SVD-based method (1)

- Single Value Decomposition (SVD)
  - dimensionality reduction
  - matrix factorization
  - A way to breakup a matrix into 3 pieces
  - The use of SVD as a way to reduce large sparse vector spaces for word meaning was first applied in the context of information retrieval, often referred to as LSA (Latent Semantic Analysis)



w = total words
c = total documents or context words
m = total latent factors (dimensions)

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

numpy.linalg.svd

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

tin. Speech and language processing. 3rd edition draft, August 2017

Dense vector representations: SVD-based method (cont.)

- The matrix X is the product of 3 matrices:
  - **W:** rows corresponding to original but m columns represents a dimension in a new latent space
    - m column vectors are orthogonal to each other
    - Columns are ordered by the amount of variance in the dataset each new dimension accounts for

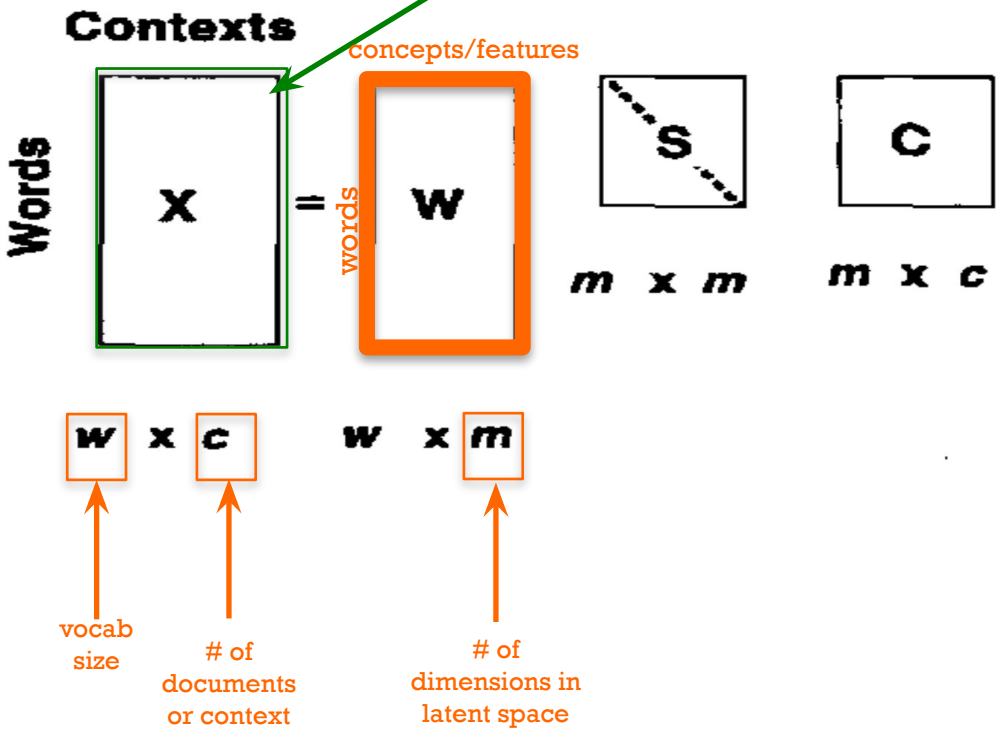
vocabulary

	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

document

Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

The cells are commonly weighted by a product of two weights:
- Local weight: Log term frequency
- Global weight: either idf or an entropy measure

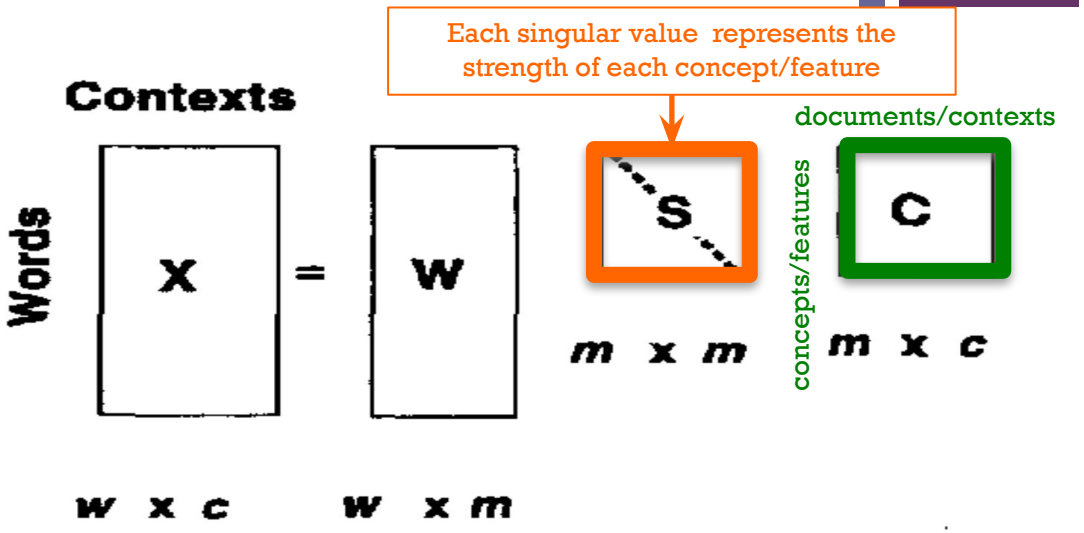


w = total words
c = total documents or context words
m = total latent factors (dimensions)

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

Dense vector representations: SVD-based method (cont.)

- The matrix **X** is the product of 3 matrices:
  - **S**: diagonal m x m matrix of **singular values** expressing the importance/strength of each dimension
  - **C**: columns corresponding to original but m rows corresponding to singular values

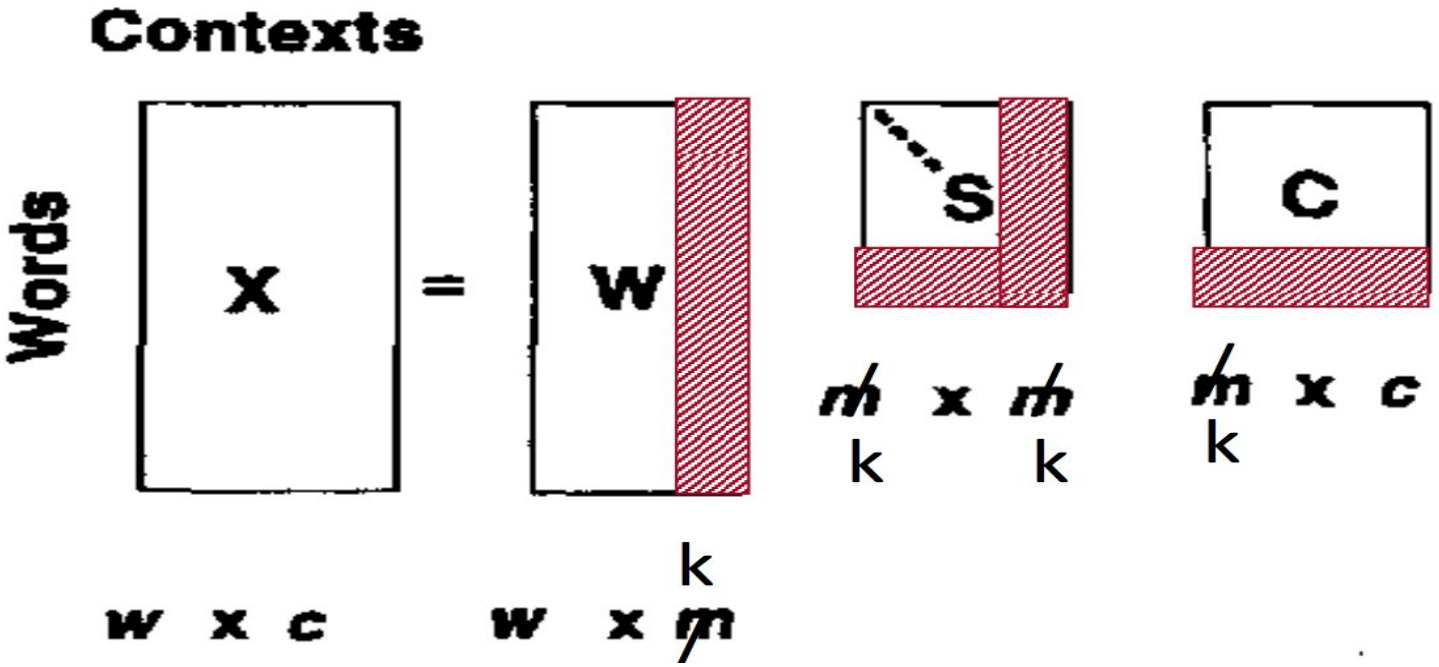


w = total words
c = total documents
m = total latent factors (dimensions)

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

Dense vector representations: SVD-based method (cont.)

- Latent Semantic Analysis (Deerwester et al. (1988))
  - instead of keeping all m dimensions, we just keep the top k singular values. (k=300 is common)



Dense vector representations: SVD-based method (cont.)

- SVD applied to word-word/word-context matrices
  - In this version the context dimensions are words rather than documents
  - The only difference is that we are using a PPMI-weighted word-word matrix as an input data matrix

word-doc matrix

vocabulary ↓ / document →

	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

Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

PPMI matrix

	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

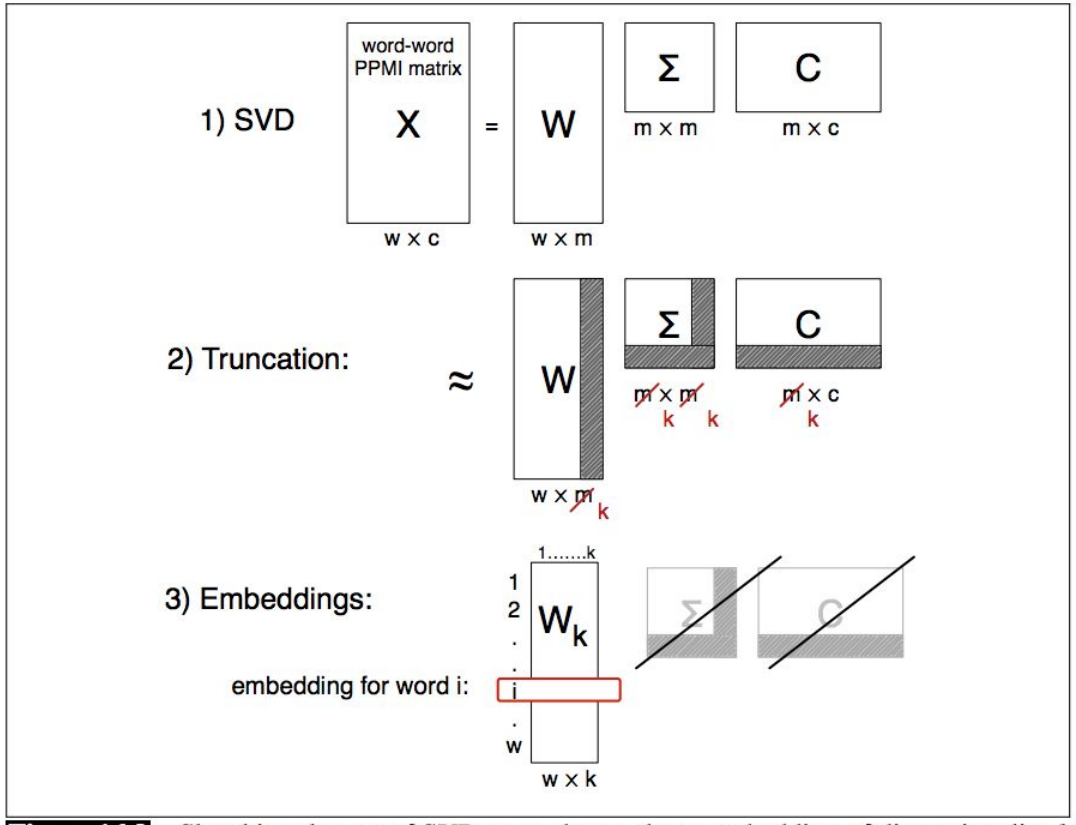


Figure 16.3 Sketching the use of SVD to produce a dense embedding of dimensionality k from a sparse PPMI matrix of dimensionality c. The SVD is used to factorize the word-word PPMI matrix into a W, Σ, and C matrix. The Σ and C matrices are discarded, and the W matrix is truncated giving a matrix of k-dimensionality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

Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

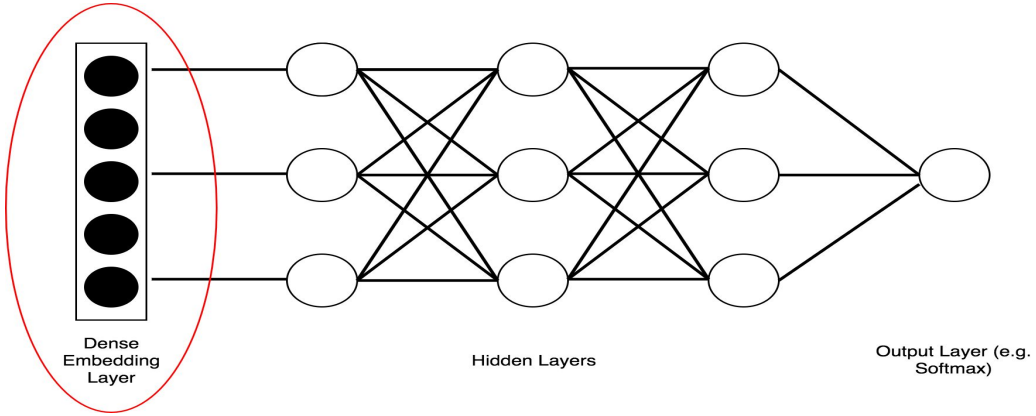
Dense vector representations: SVD-based method (cont.)

- We can apply SVD/LSA to one of the sparse vector representations to yield a word representation with smaller dimensionality.
- Advantages:
  - Generalize better on unseen data
  - Reduce low-order dimensions that may represent unimportant information
  - Less parameters to tune

2 Dense vector representations : neural networks

How do we train embeddings in neural networks?

1. **Initialize randomly** and train it on a your target task.
2. **Pre-train** on unsupervised task (e.g. LM: skip-gram, CBOW)
   - **we will learn how to do this today**
3. **Pre-train** on supervised task
   - e.g. train on POS tagging task and use it on other tasks

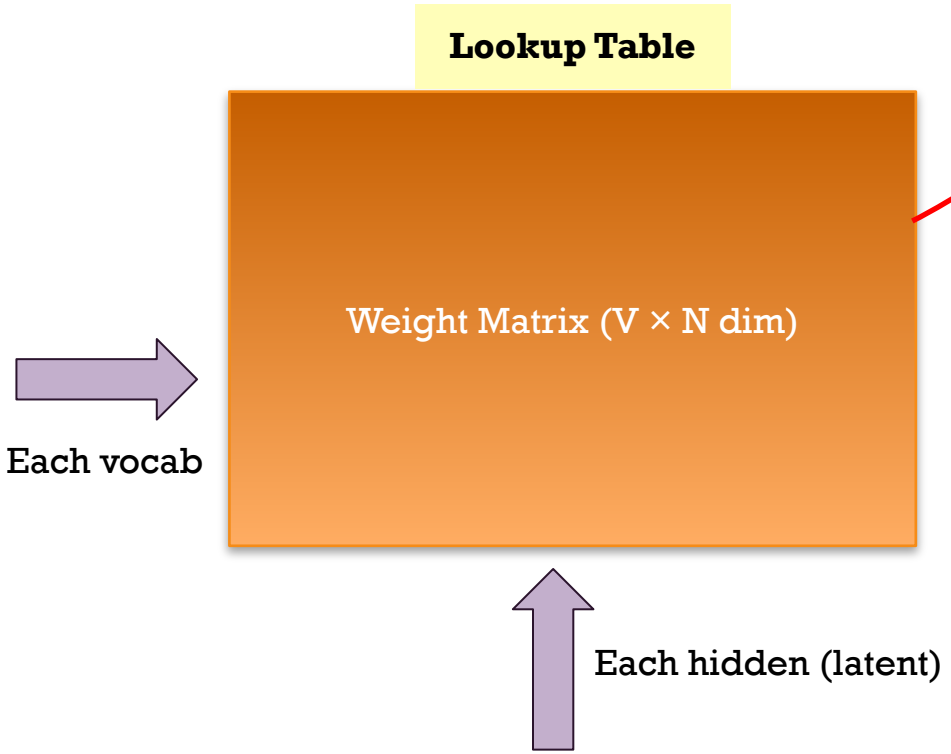


reference: Neubig (2020), https://www.youtube.com/watch?v=RRaU7pz2eT4

Dense vector representations : neural networks (cont.)

- Tomas Mikolov introduced **Skip-gram** in 2013
  - **CBOW** was proposed before by other researchers.
- Train a neural network to predict neighboring words
- Word representation can be learned as a part of the process of word prediction.
- Part of a neural network (embedding layer) can be used as word representation in various NLP tasks
- Advantages:
  - Faster than SVD
  - Pre-trained word representations are available online!

Dense vector representations : neural networks (cont.)



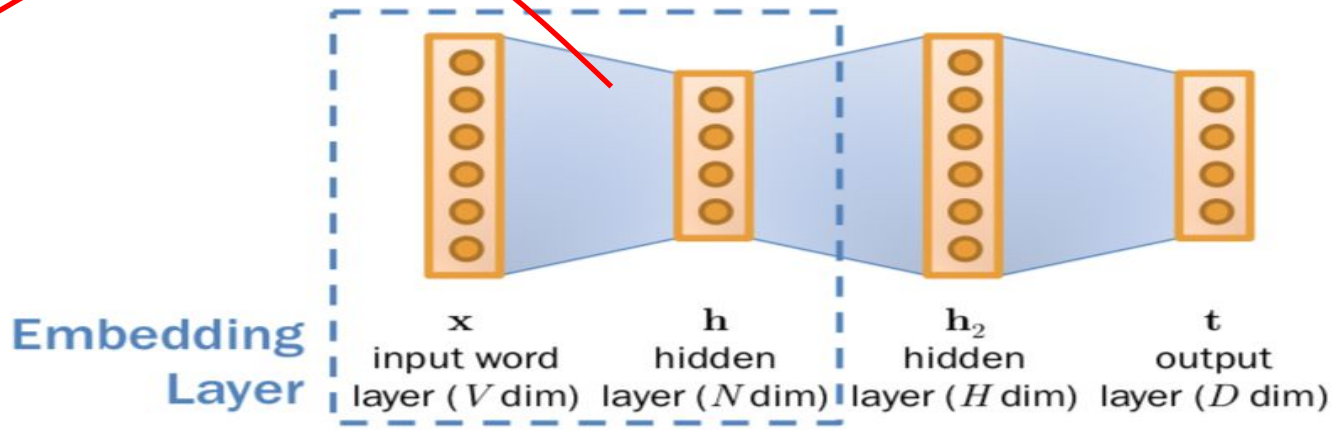
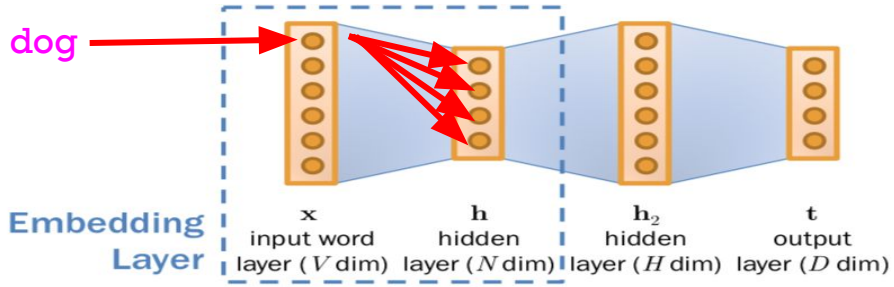
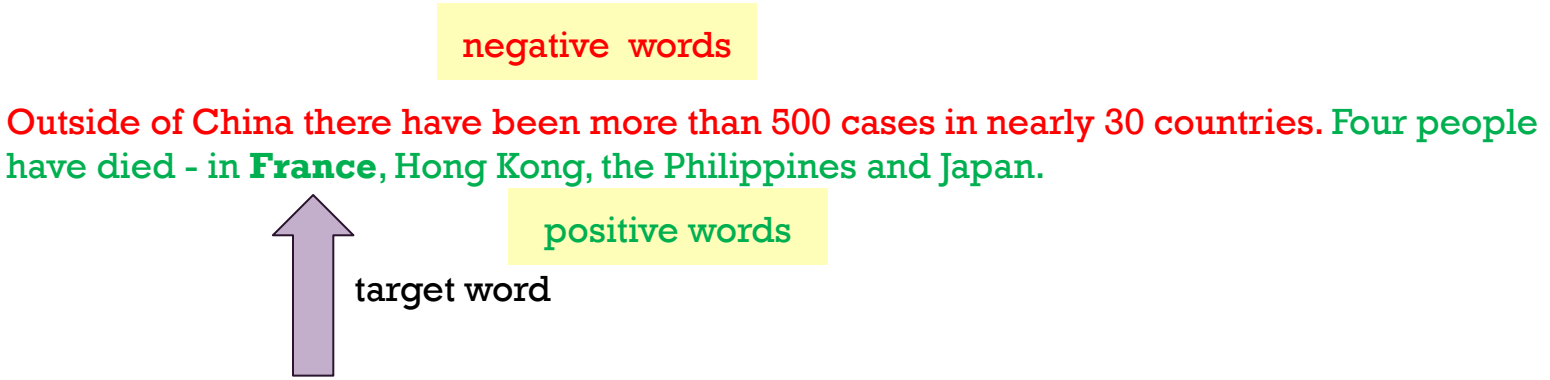


Image reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017



Dense vector representations : neural networks (cont.)

- **CBOW** and **Skip-Gram** intuition: Iteratively make the embeddings for a word
  - (positive class) more like the embeddings of its neighbors and
  - (negative class) less like the embeddings of other words.



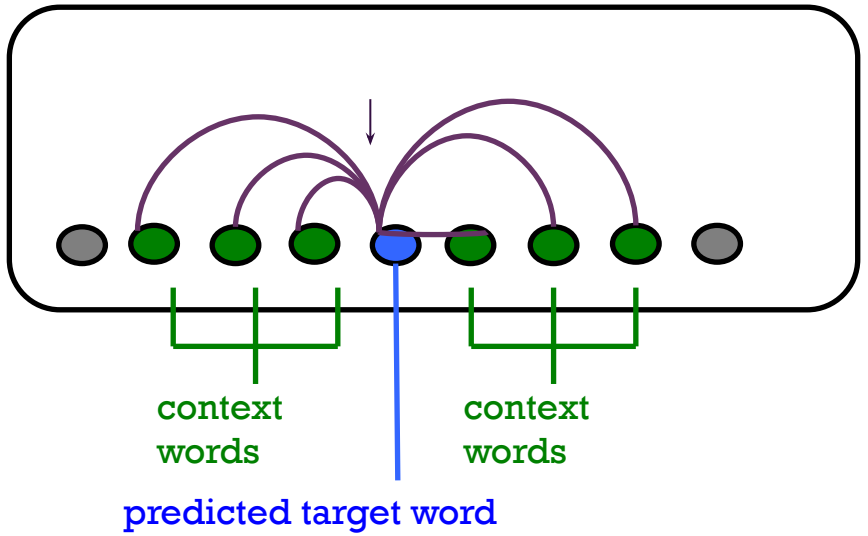
Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

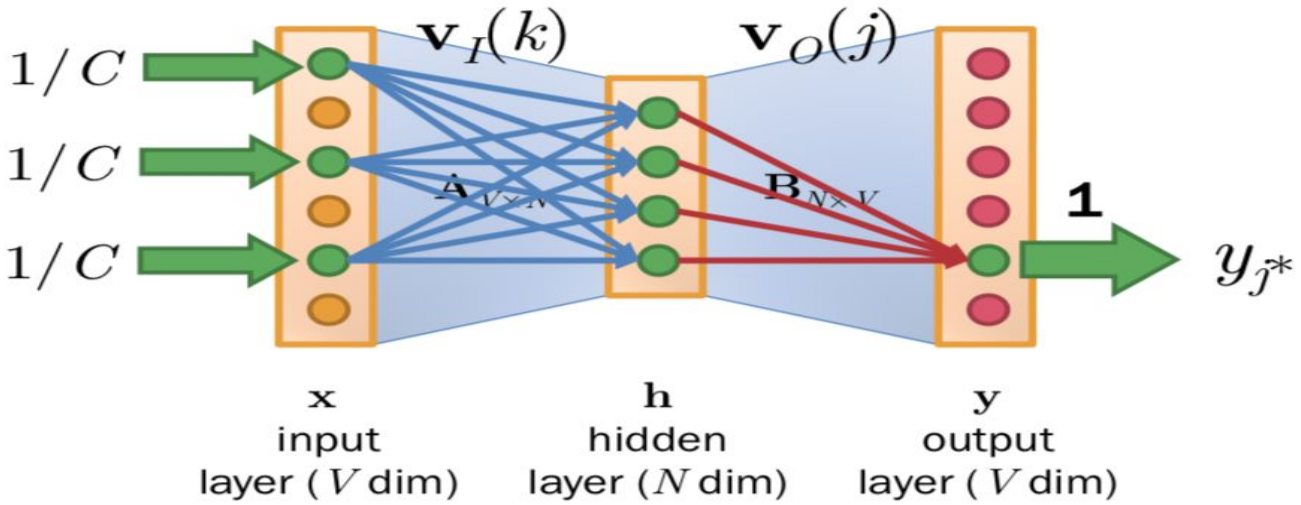
https://www.bbc.com/news/world-asia-china-51519055

1) CBOW

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.

- Continuous Bag-of-Words (CBOW)
- In CBOW neural language model, ONE target word is predicted from SEVERAL context words.





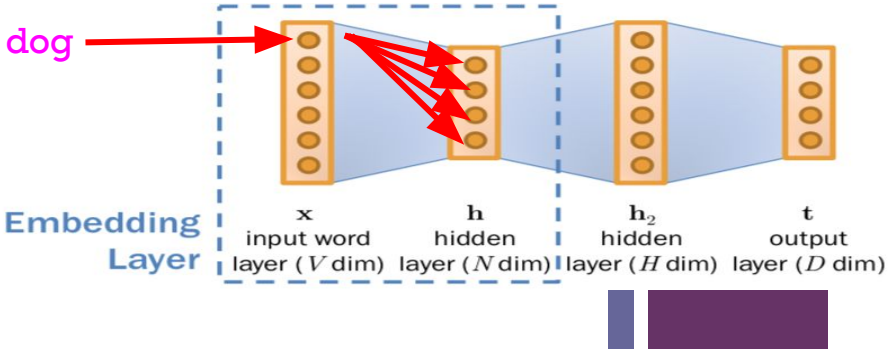
$$\mathbf{h}_{\text{ctx}} = \frac{1}{C}\sum_{q=1}^{C}\mathbf{v}_I(q)$$

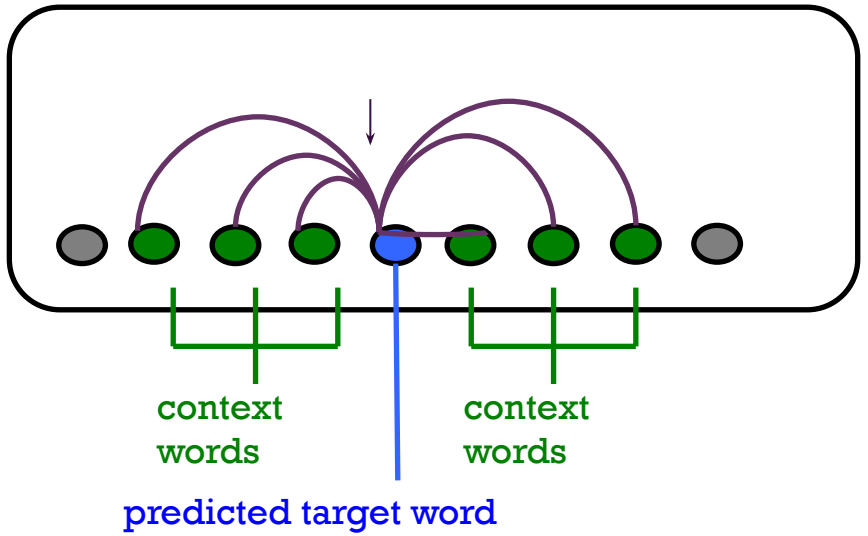
Image reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017

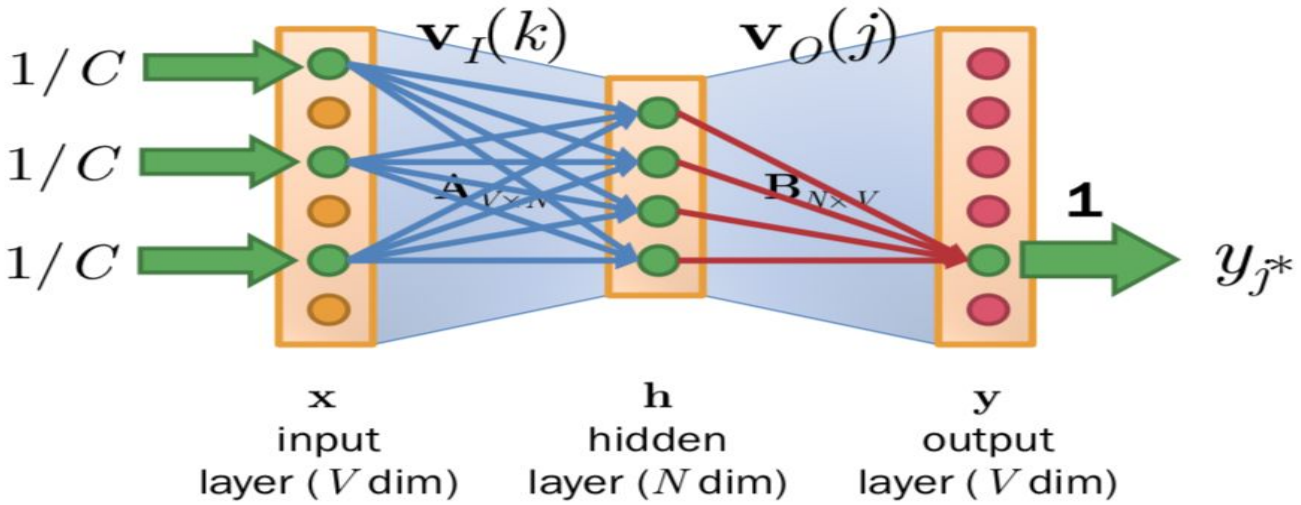
1) CBOW (cont.)

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.

- We simply average the encoding vectors







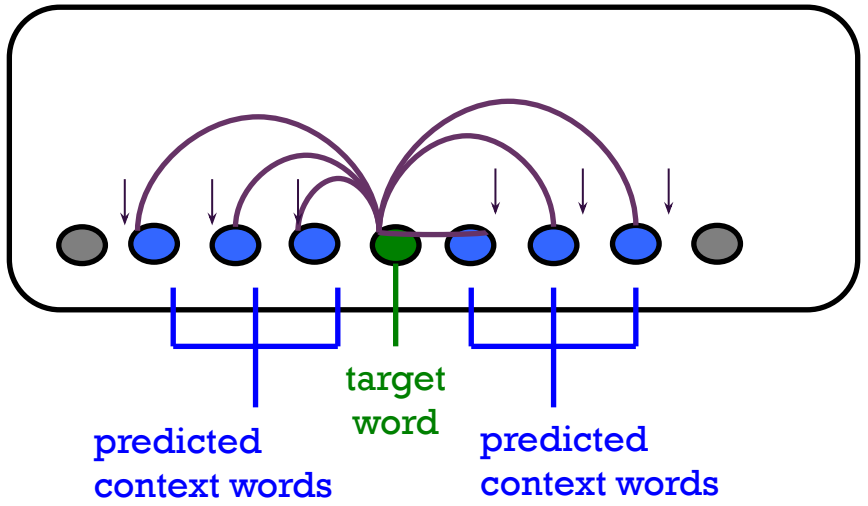
$$\mathbf{h}_{\text{ctx}} = \frac{1}{C}\sum_{q=1}^{C}\mathbf{v}_I(q)$$

Image reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017

2) Skip-gram

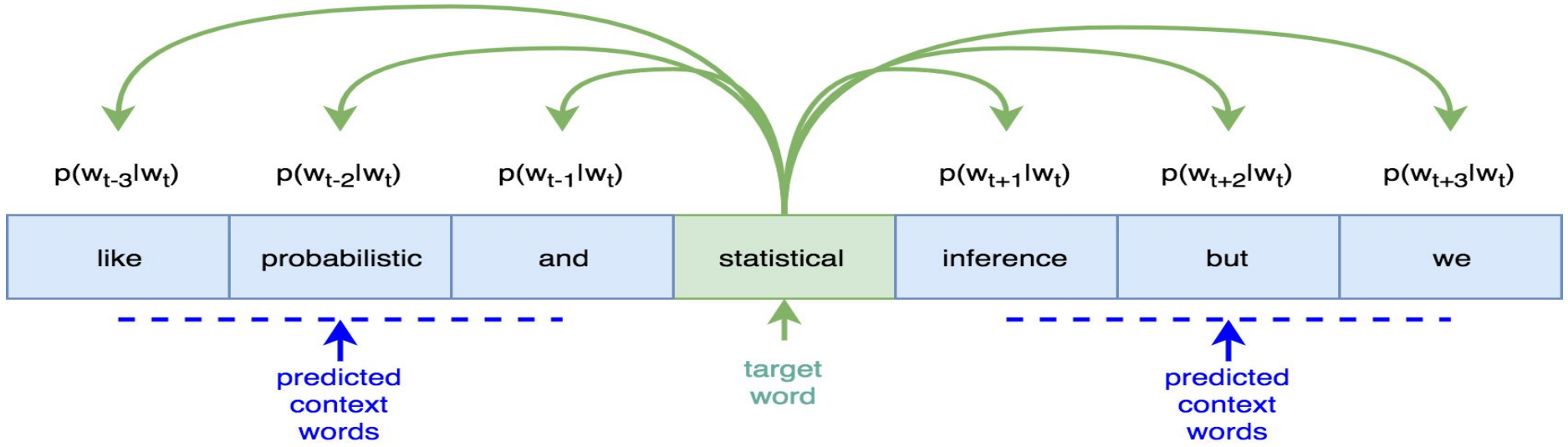
Outside of China there have been more than 500 cases in nearly 30 countries. Four people have died - in **France**, Hong Kong, the Philippines and Japan.

- In skip-gram neural language model, SEVERAL context words are predicted from ONE target word.
- In "Efficient Estimation of Word Representations in Vector Space", Mikolov shows that skip-gram performs better than CBOW in several tasks. BUT skip-gram model requires more training time.
- In this lecture, we will show you how skip-gram works



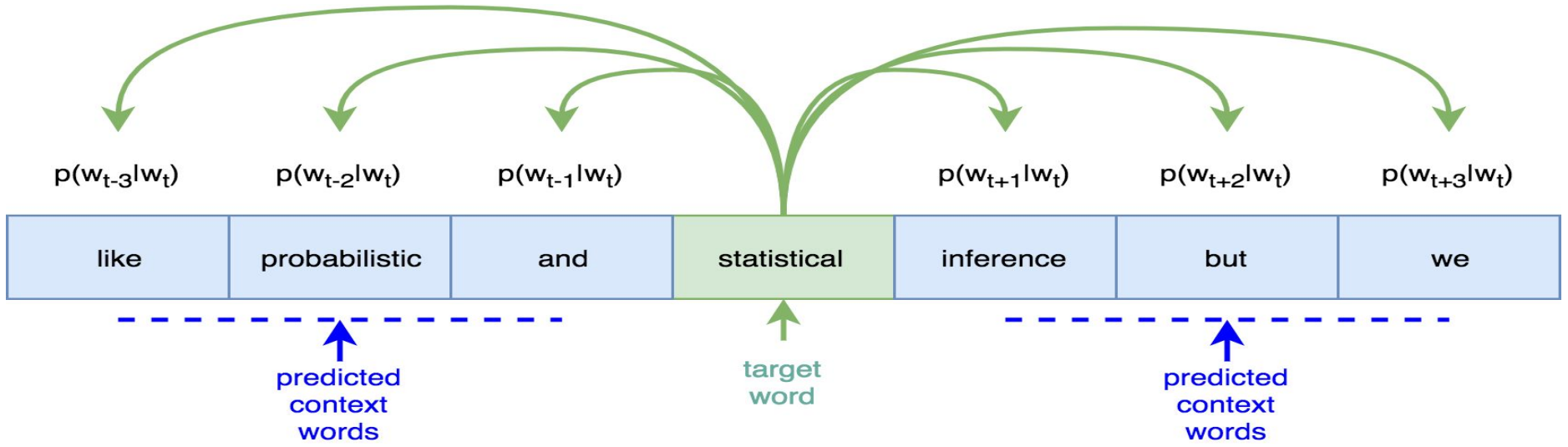
2) Skip-gram (cont.)
Multiclass classification

- Skip-gram prediction
- Consider the following passage:
  - "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."



2) Skip-gram (cont.)
Multiclass classification

- For each word t = 1…T, predict its surrounding context words (next m words and previous m words)
- "m" is the window size



2) Skip-gram (cont.)
Multiclass classification

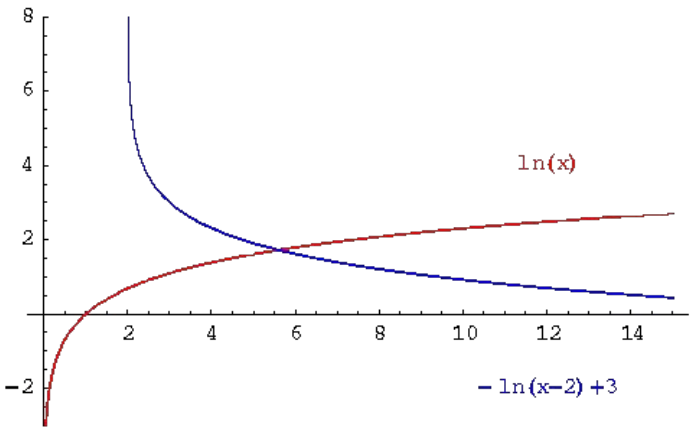
- Likelihood function: Given the target word (aka center word), maximize the probability of each context word

$$J'(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m; j \ne 0} p(w_{t+j} | w_t; \theta)$$

j = -m → previous words
j = +m → next words
j = 0 → the input word (w_t)

- Cost/Loss Function (Negative Log-Likelihood): (minimize)

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

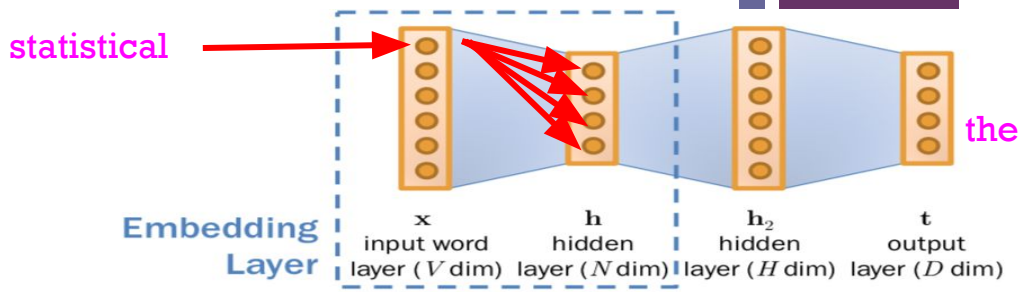


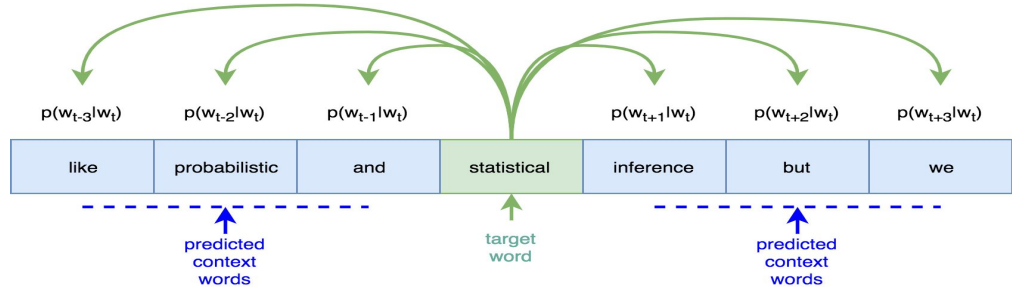
Reference: http://web.stanford.edu/class/cs224n/lectures/cs224n-2017-lecture2.pdf

2) Skip-gram (cont.)

Multiclass classification

- How to calculate $p(w_{t+j}|w_t;\theta)$?
  - for each word w, we will use two vectors
    - 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'

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

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

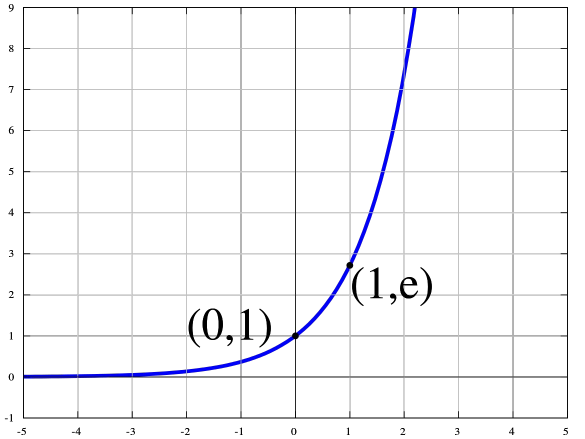
After taking exponent, normalize over entire vocabulary

Reference: http://web.stanford.edu/class/cs224n/lectures/lecture2.pdf

2) Skip-gram (cont.)

Multiclass classification: Softmax

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



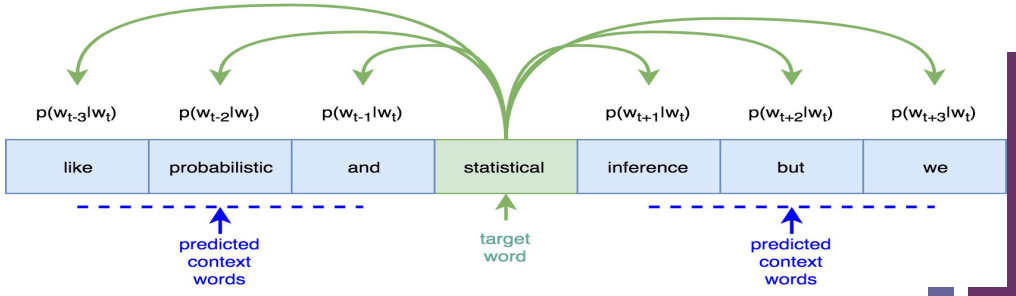
- This is basically a softmax function

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i

Reference: http://web.stanford.edu/class/cs224n/lectures/lecture2.pdf

2) Skip-gram (cont.)
Multiclass classification



- Negative Log-Likelihood of Skip-gram model:

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

- **Notice the difference** between skip-gram's cost function and the cost function of neural language model from the last lecture

Predict context words

VS

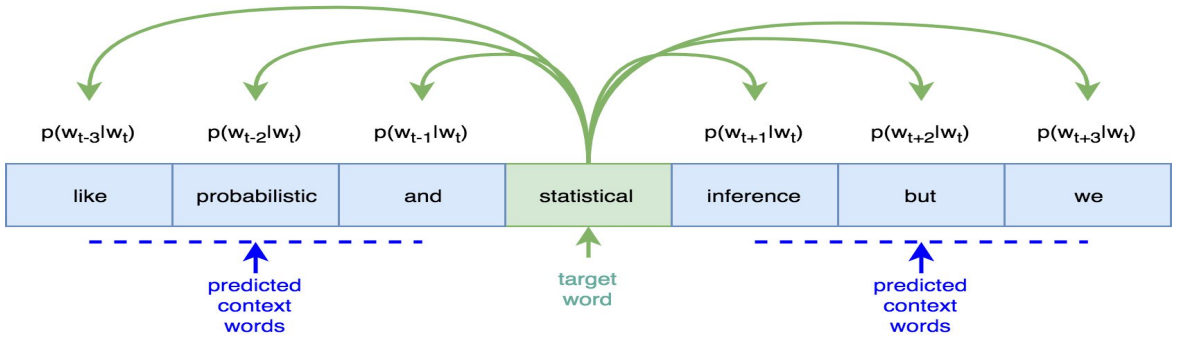
Neural Language Model (cont.)

- Recurrent Neural Network (RNN)
  - Cost function:
    - $$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log \hat{y}_{t,j}$$
  - Where
    - V = Number of unique words in corpus
    - T = Number of total words in corpus
    - y = Target next word
    - $\hat{y}$ = Distribution of predicted next word

Predict the next word

2) Skip-gram (cont.)
Multiclass classification (Steps)

- 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*m score vectors (where m is the window size)
  4. Turn the score vectors into probabilities
  5. We desire our probability vector generated to match the true probabilities which are the one hot vectors of the actual output.

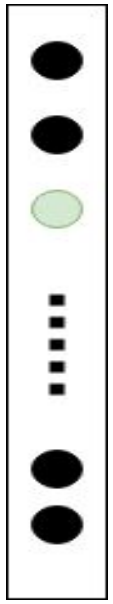


Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf

2) Skip-gram (cont.)
Multiclass classification (Steps)

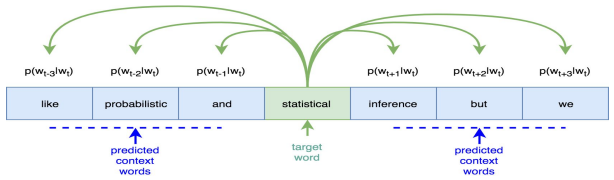
- 1. Generate a one hot input vector ***x*** for of the target word (center word)

statistical = [0 0 1 … 0 0]



$$x \in \mathbb{R}^{|V|}$$

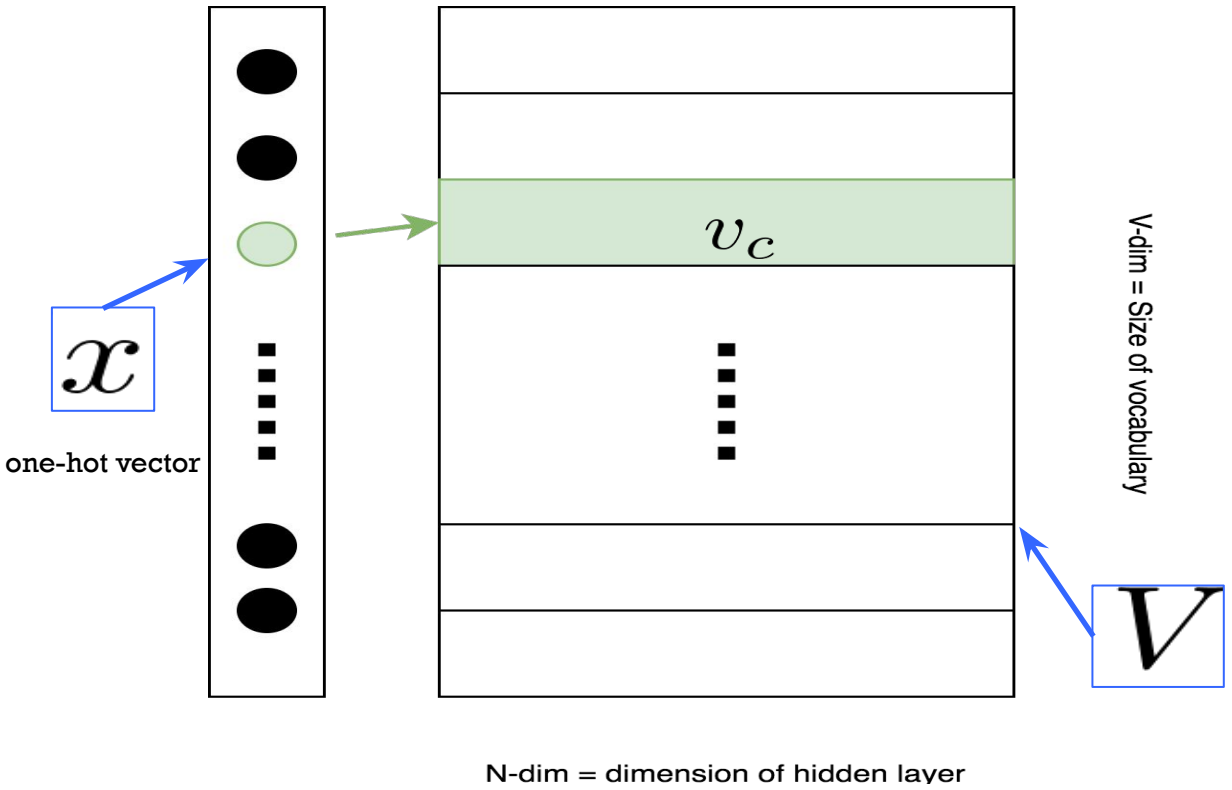
V-Dim = size of the vocabulary



Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf

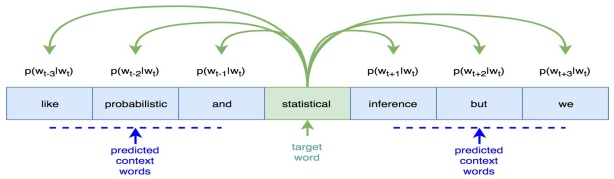
2) Skip-gram (cont.)
Multiclass classification (Steps)

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



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

Note that the weight matrix in the embedding layer can be think of as a look-up table



Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf

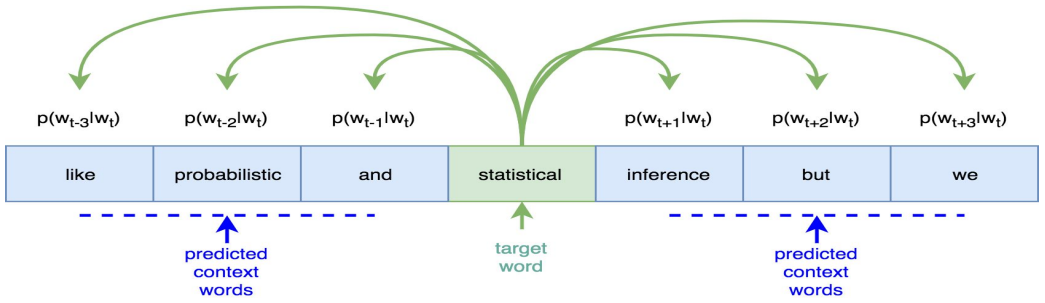
2) Skip-gram (cont.)

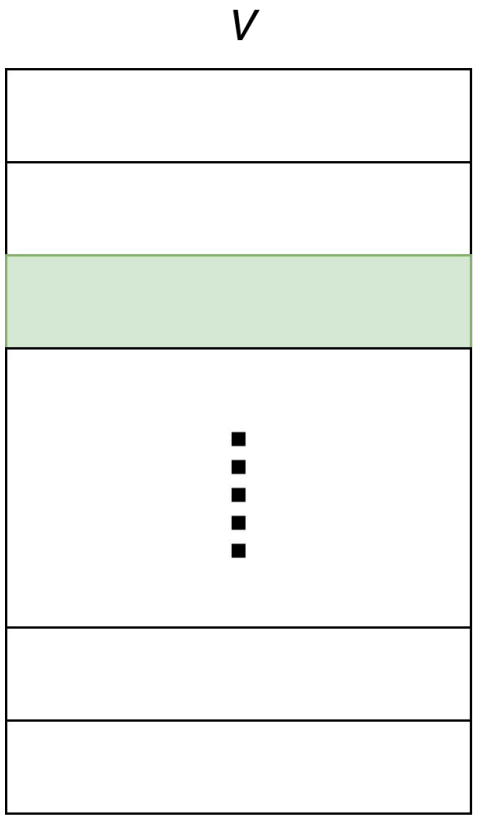
Multiclass classification (Steps)

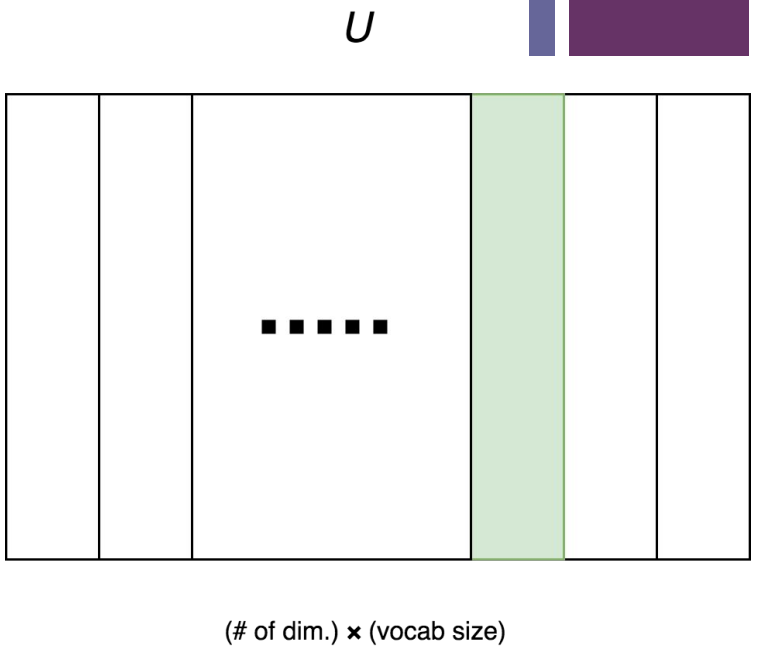
- 3. Generate a score vector

$$z = Uv_c$$

- Note that similarity(v_c,u_o) = $v_c \cdot u_o$
  - v_c = center (target), u_o=other (context)







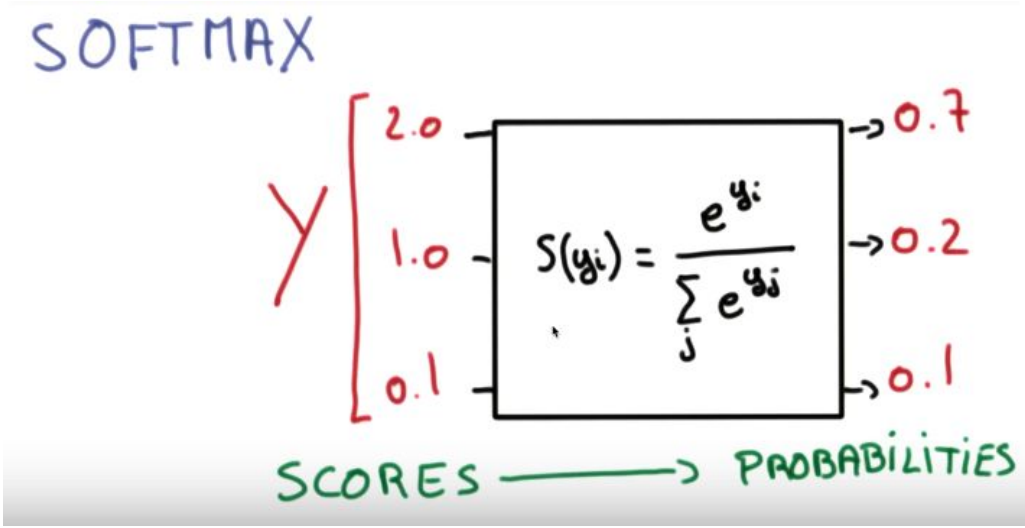
Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf

2) Skip-gram (cont.)
Multiclass classification (Steps)

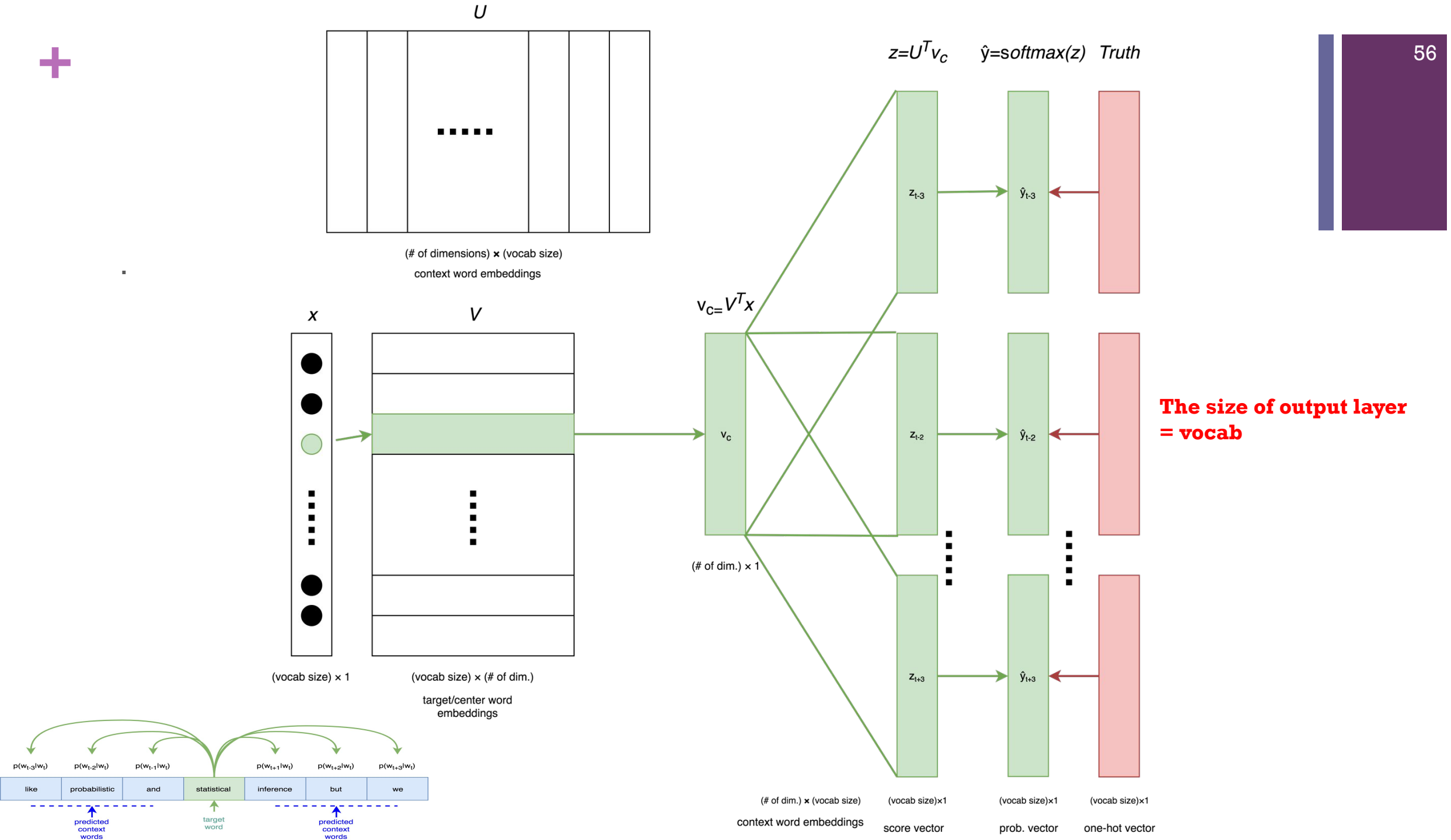
- 4. Turn score into probabilities

$$\hat{y} = softmax(z)$$

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



Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf
Image reference:https://www.udacity.com/course/deep-learning--ud730



The size of output layer = vocab

2) Skip-gram (cont.)
Multiclass classification (Steps)

- **Word2Vec training methods**
- Softmax is not very efficient (slow)
- Computational Cost : O(|V|)
- Solution: Two efficient training methods
  - Hierarchical Softmax: O(log(|V|)) – not covered
  - Negative Sampling

Word2Vec training methods : Solution 1 : Hierarchical Softmax

- Softmax as tree traversal
- Hierarchical softmax uses a binary tree to represent words
- 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

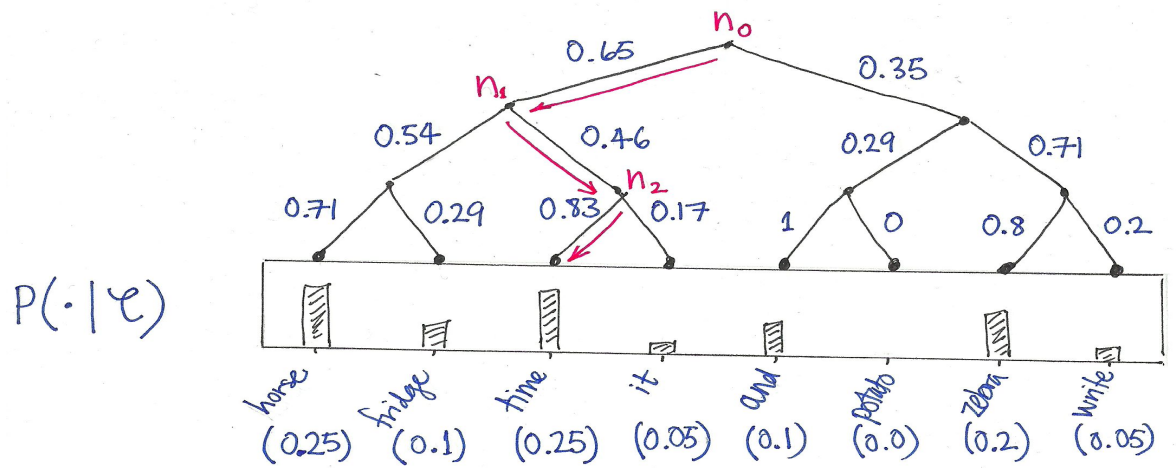
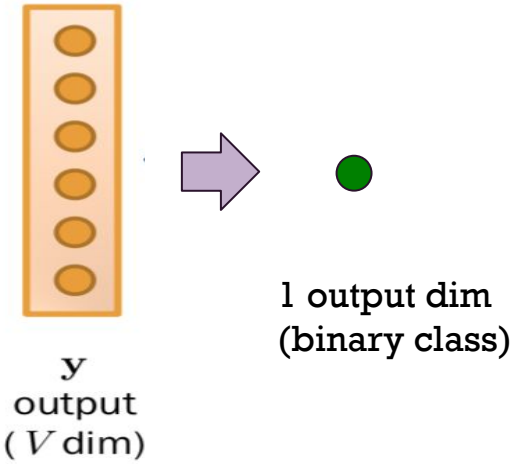


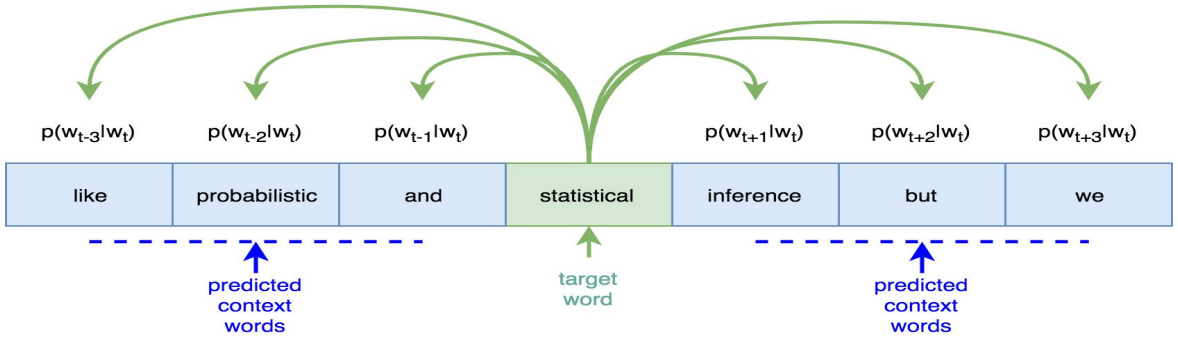
Image reference:
http://building-babylon.net/2017/08/01/hierarchical-softmax/

Multiclass → Binary Classification
Solution2: Negative Sampling (1)

- We assume that the dataset is noisy containing:
  - Positive examples: correct output words
  - Negative examples: incorrect output words
- How to reduce computational cost?
  - The positive examples should be kept
  - Only k negative examples are sampled from a distribution



- **V (target), U (context), label**
- 6 pairs of positive samples
  - statistical, and, 1
  - statistical, inference, 1
  - …
- 6 pairs of negative samples
  - statistical, cat, 0
  - statistical, dog, 0
  - …



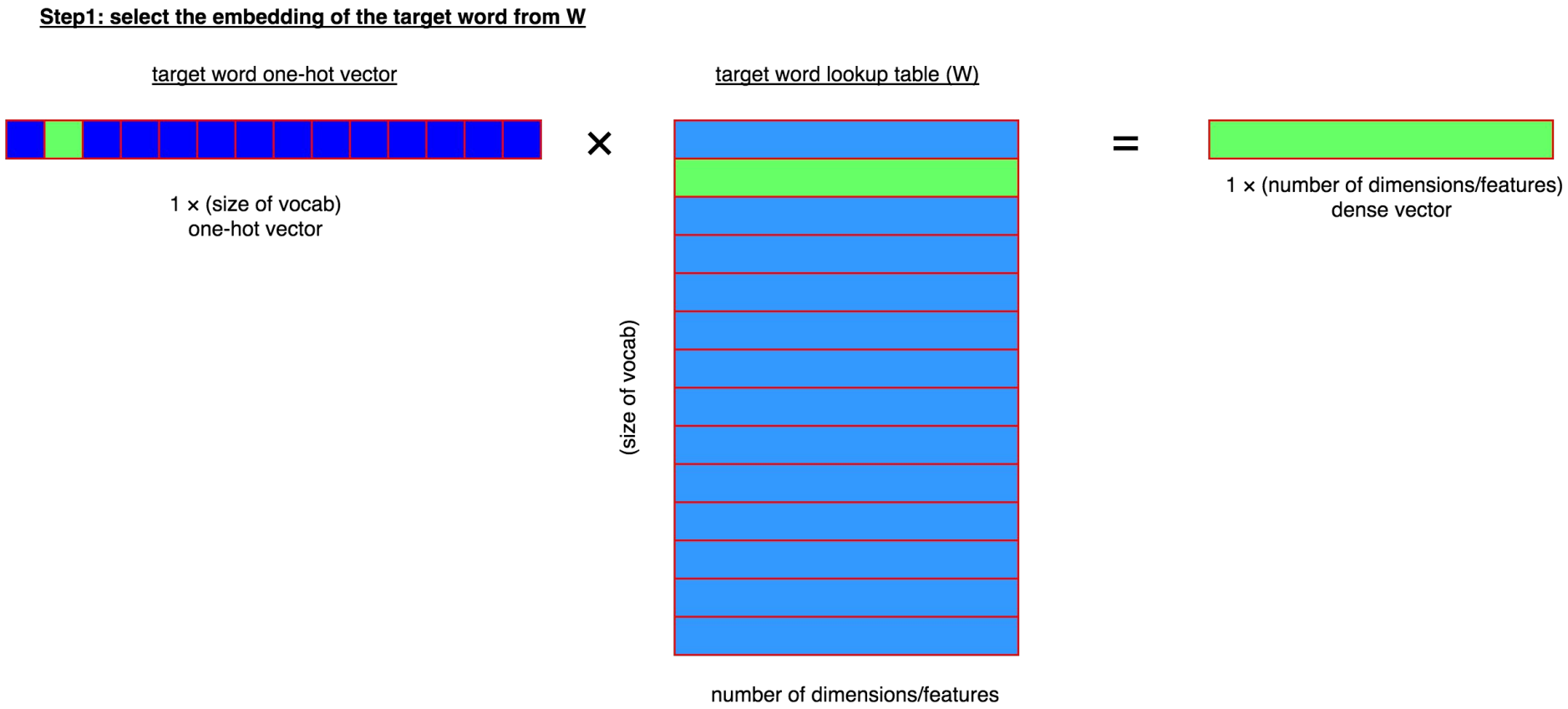
Reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017

Word2Vec training methods : Solution2: Negative Sampling (2)

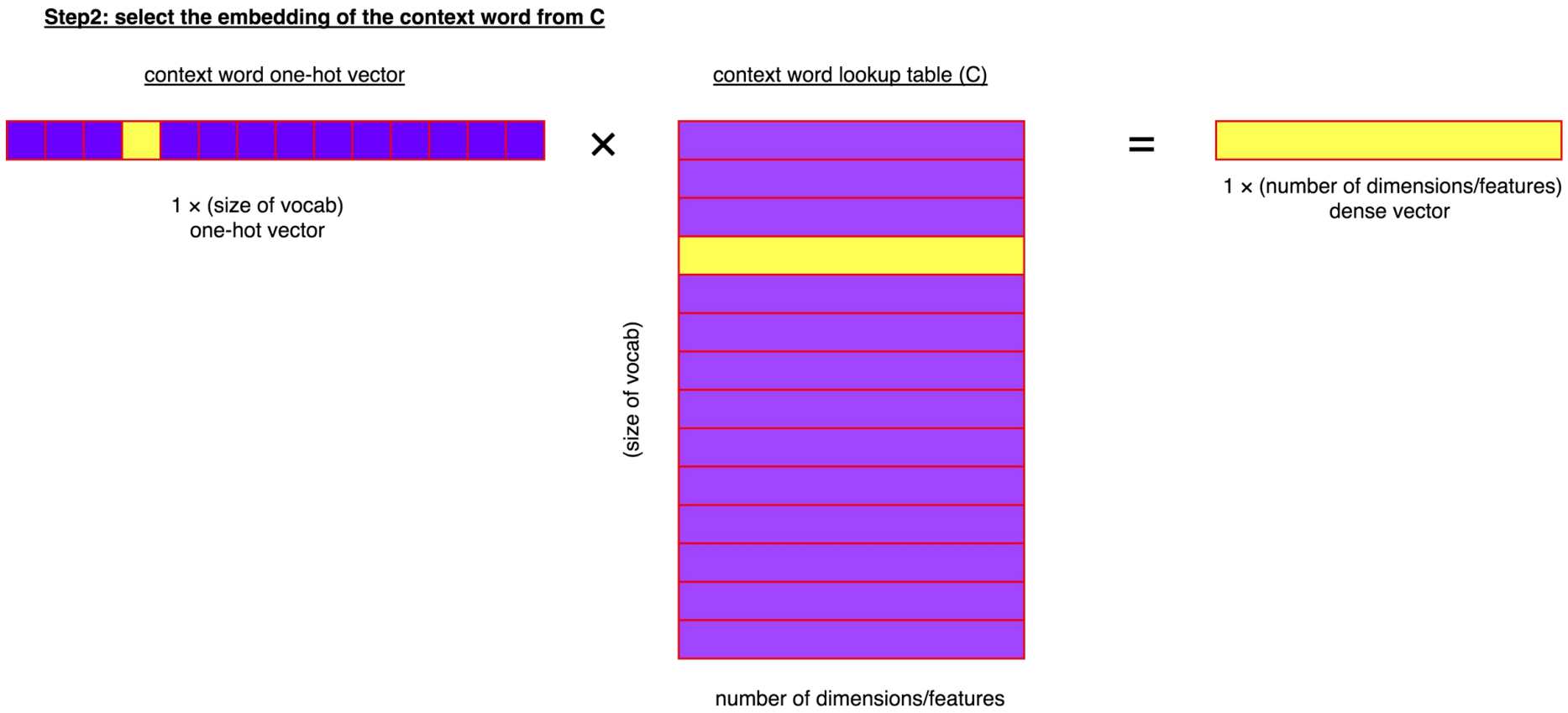
- Why ¾?
  - Chosen based on empirical experiments
- Intuition:
  - ที่: $0.9^{3/4} = 0.92$
  - ชาวนา: $0.09^{3/4} = 0.16$
  - ตถาตา: $0.01^{3/4} = 0.032$
  - A rare word such as 'ตถาตา' is now 3x more likely to be sampled
  - While the probability of a frequent word ที่ only went up marginally

$$P^{\frac{3}{4}}(w) = \frac{(w)^{\frac{3}{4}}}{\sum_{w'}(w')^{\frac{3}{4}}}$$

2) Skip-gram (cont.)
Negative Sampling (Binary Classification)

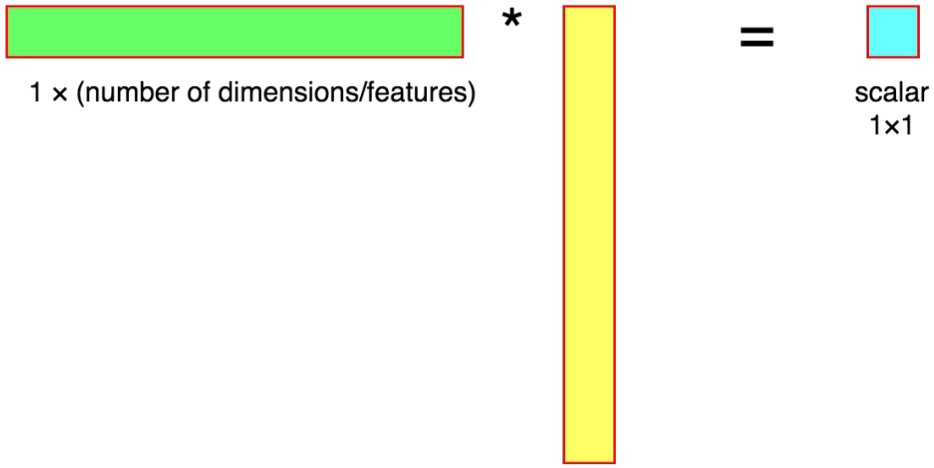


2) Skip-gram (cont.)
Negative Sampling (Binary Classification)

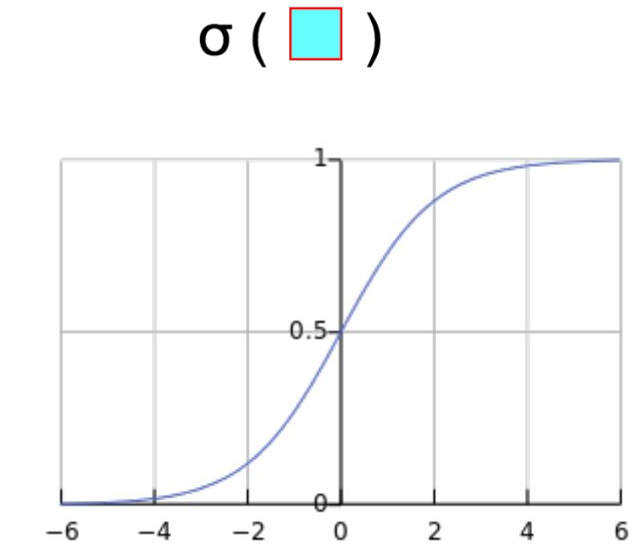


2) Skip-gram (cont.) Negative Sampling (Binary Classification)

Step3: compute the dot product:w*c



Step4: normailize dot products into probability



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

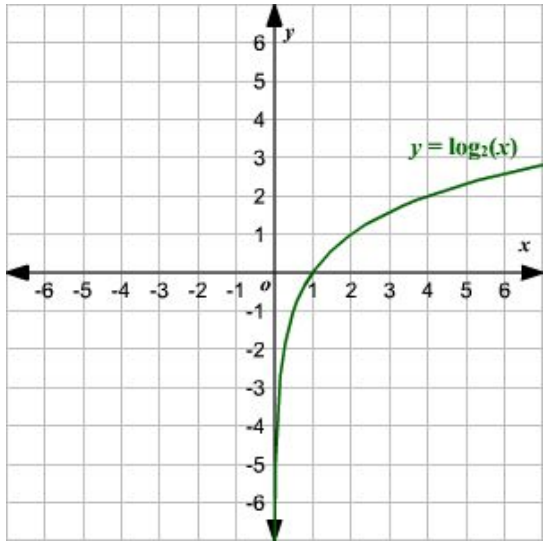
2) Skip-gram (cont.) Negative Sampling (Binary Classification)

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

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

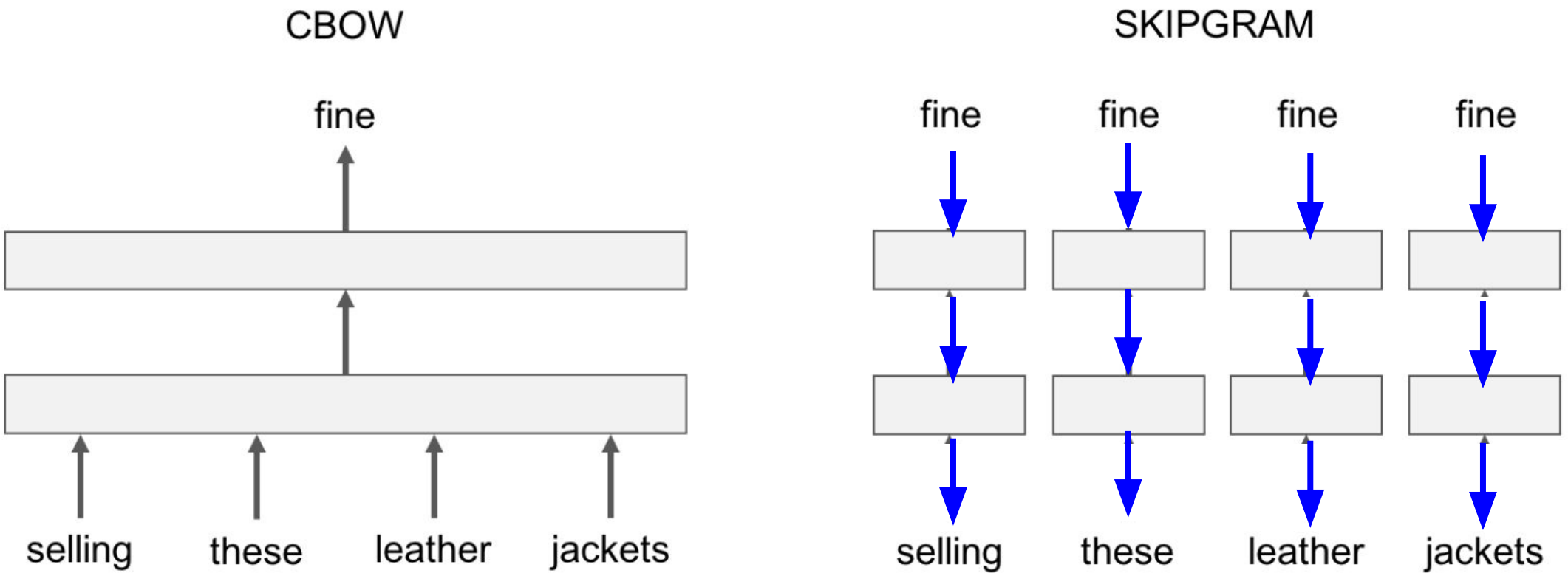
Context word (positive, +1)

Negative samples (negative, -1)



σ = sigmoid

Summary: CBOW vs Skip-gram (recap)

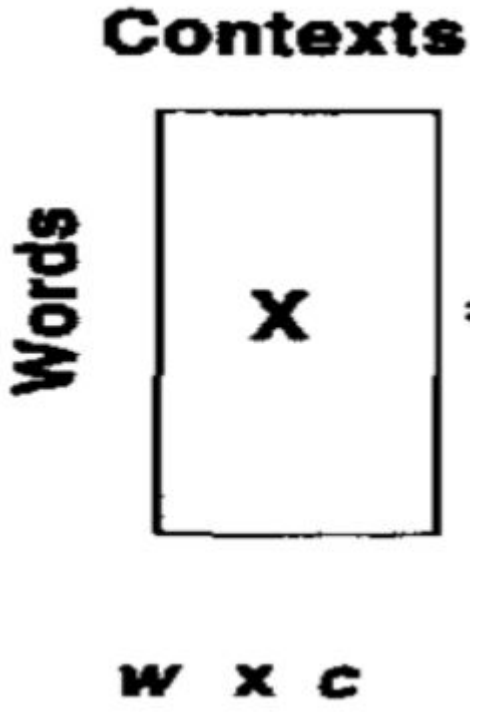


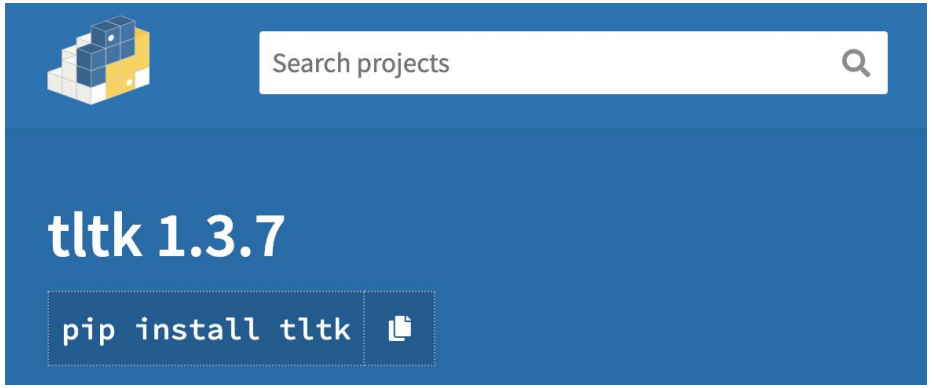
I am selling these fine leather jackets

https://fasttext.cc/docs/en/unsupervised-tutorial.html

Pre-trained Word2Vec

- **1) Non-contextualized Word Embeddings (fixed vector)**
- 1.1) GloVe (Stanford) [not support Thai language]
  - https://nlp.stanford.edu/projects/glove/
- 1.2) fastText [Available in Thai language] (Facebook)
  - https://github.com/facebookresearch/fastText
- 1.3) Word2Vec in TLTK (Aj.Wirote)
  - tltk.corpus.w2v(w)
- 1.4) Large Thai Word2Vec (LTWV): CBOW
- **2) Contextualized Word Embeddings**
- 2.1) thai2fit
  - ULMFit
  - https://github.com/cstorm125/thai2fit/
- 2.2) BERT [Available in Thai language] (Google)
  - https://github.com/google-research/bert
- 2.3) A lot more, i.e., ELMO, FLAIR, BERT variations





Pre-trained Word2Vec: GloVe

Global Vector for Word Representation

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

- GloVe is an unsupervised learning algorithm for obtaining vector representations for words.
- Training is performed on aggregated global **word-word co-occurrence** statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space
- Pre-trained word vectors from different domains are available (Wikipedia+Gigaword, Common Crawl, Twitter)
- 822MB; 50-dimensional, 100-dimensional, 200-dimensional, 300-dimensional

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning
Computer Science Department, Stanford University, Stanford, CA 94305
jpennin@stanford.edu, richard@socher.org, manning@stanford.edu

Reference: Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

Pre-trained Word2Vec: GloVe (cont.)

https://www.aclweb.org/anthology/D14-1162

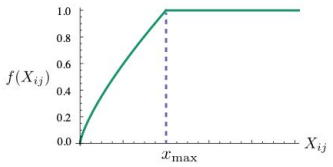
- Main Ideas
  - capture **meaning** in vector space
  - Takes **advantage of co-occurrence statistics** instead of only local information

1 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}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

2 $$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

3 $$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$



weighting function prediction term

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

4) weight = average of all *k* words

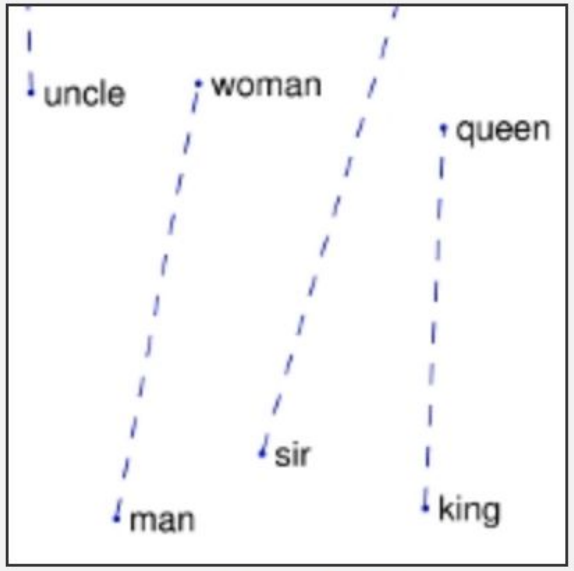
word co-occurrence matrix

	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

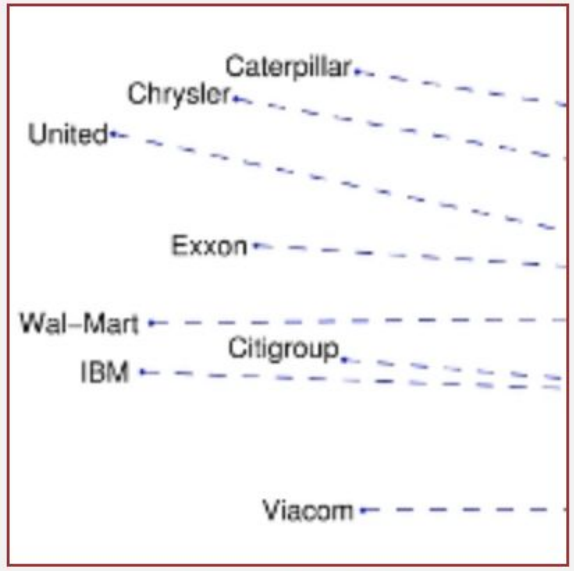
Figure 15.1 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

Pre-trained Word2Vec: GloVe (cont.)

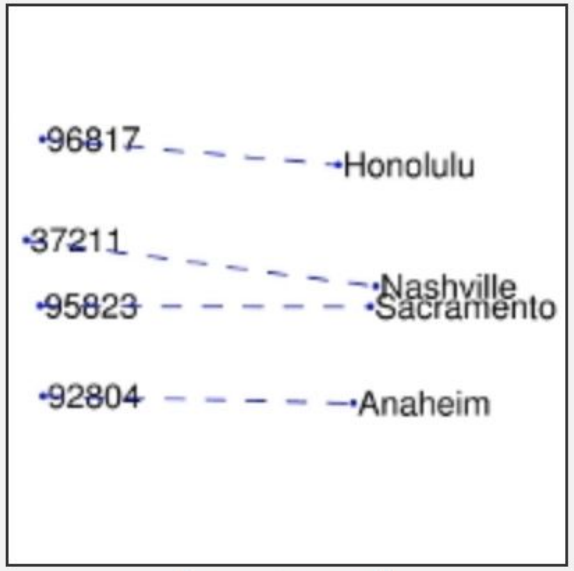
- Linear substructures



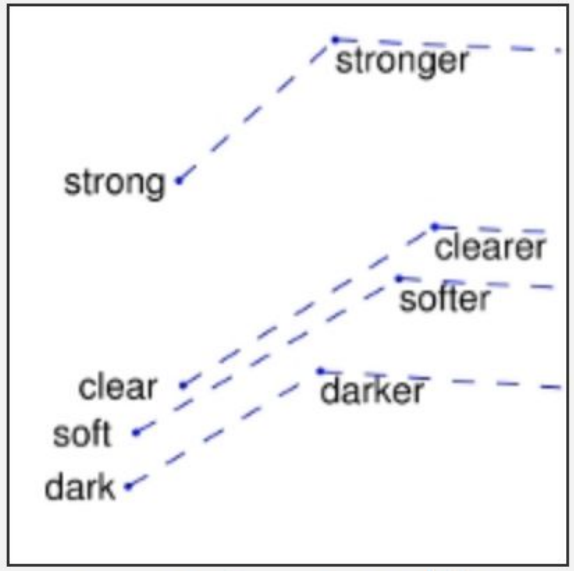
man - woman



company - ceo



city - zip code



comparative - superlative

Pre-trained Word2Vec: fastText

Skip-Gram (or CBOW) of "sum of subwords (char n-grams)"

- fastText is a library for efficient learning of word representations and sentence classification.
- **Character** n-grams as additional features to capture some partial information about the local word order.
  - Good for **rare** words, since rare words can share these n-grams with common words
- Pre-trained word vectors for 157 languages (including Thai) trained on Wikipedia.

<where>. <wh, whe, her, ere, re>

Reference: Bojanowski, Piotr, et al. "Enriching word vectors with subword information." arXiv preprint arXiv:1607.04606 (2016).

Joulin, Armand, et al. "Bag of tricks for efficient text classification." arXiv preprint arXiv:1607.01759 (2016).

https://cai.tools.sap/blog/glove-and-fasttext-two-popular-word-vector-models-in-nlp/

Word vectors for 157 languages

We distribute pre-trained word vectors for 157 languages, trained on *Common Crawl* and *Wikipedia* using fastText. These models were trained using CBOW with position-weights, in dimension 300, with character n-grams of length 5, a window of size 5 and 10 negatives. We also distribute three new word analogy datasets, for French, Hindi and Polish.

Download directly with command line or from python

In order to download with command line or from python code, you must have installed the python package as described here.

Command line | Python

```
$ ./download_model.py en     # English
Downloading https://dl.fbaipublicfiles.com/fasttext/vectors-crawl/cc.en.300.bin.gz
 (19.78%) [==========>                                          ]
```

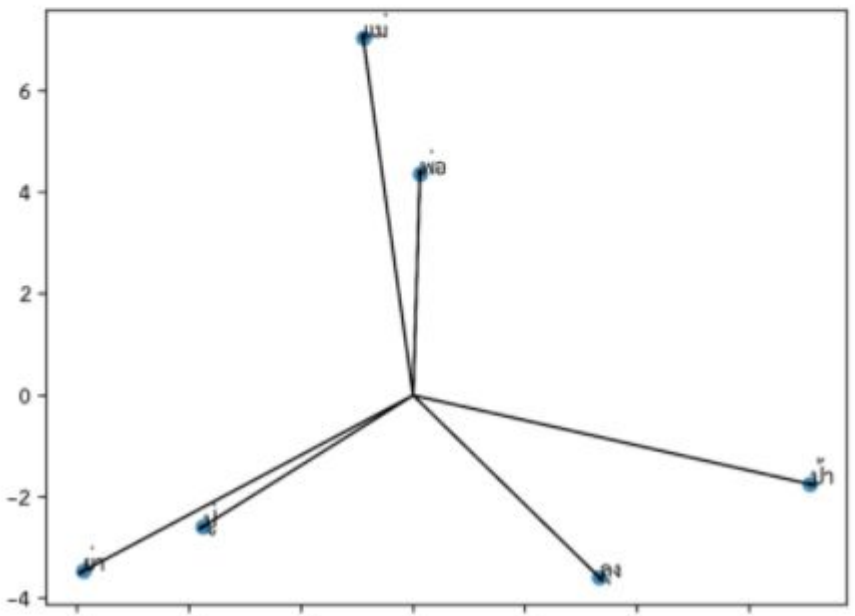
https://fasttext.cc/docs/en/crawl-vectors.html

Thai word2vec (Aj.Wirote)

เราเรียนรู้อะไรจาก word2vec

Wirote Aroonmanakun Nov 21, 2018 · 4 min read

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



Search projects

Help Sponsor Log in Register

tltk 1.3.7

`pip install tltk`

Latest version

Released: Jul 30, 2020

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.

```
>>> 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']
```

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

https://pythainlp.github.io/pythainlp-corpus/ltw2v.html

+

Large Thai Word2Vec

CBOW
Skip-gram

Bases: `object`

Iterate over sentences from the "text8" corpus, unzipped from http://mattmahoney.net/dc/text8.zip.

class gensim.models.word2vec.Word2Vec(sentences=None, corpus_file=None, vector_size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75, cbow_mean=1, hashfxn=<built-in function hash>, epochs=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False, callbacks=(), comment=None, max_final_vocab=None, shrink_windows=True)

Bases: `SaveLoad`

lower than this.

- **workers** (*int, optional*) – Use these many worker threads to train the model (=faster training with multicore machines).
- **sg** (*{0, 1}, optional*) – Training algorithm: 1 for skip-gram; otherwise CBOW.
- **hs** (*{0, 1}, optional*) – If 1, hierarchical softmax will be used for model training. If 0, and *negative* is non-zero, negative sampling will be used.

ltw2v

latest version: 0.1

Description : LTW2V: The Large Thai Word2Vec

Long Description : LTW2V is The large Thai Word2Vec. It built with oxidized-thainlp from OSCAR Corpus (Open Super-large Crawled Aggregated coRpus).

HomePage : https://github.com/PyThaiNLP/large-thaiword2vec

Authors : Wannaphong Phatthiyaphaibun

OSCAR

OSCAR or **O**pen **S**uper-large **C**rawled **A**ggregated co**R**pus is a huge multilingual corpus obtained by language classification and filtering of the Common Crawl corpus using the Ungoliant architecture.

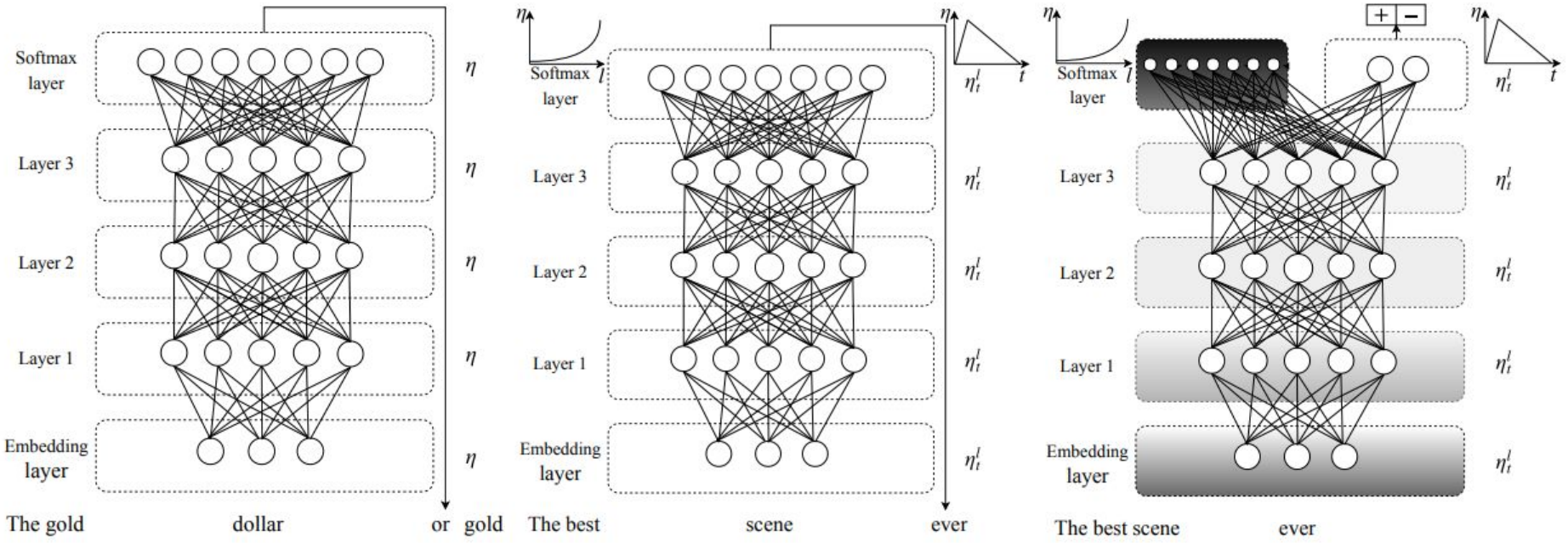
Number

- line: 3,749,826
- Words: 1,739,705,916
- size: 400

embedding dimensions

https://colab.research.google.com/drive/1XSoEIRWz6l9w7Ebg9vfYEueFLdpZDvdo?usp=sharing

Pre-trained Word2Vec: Universal Language Model Fine-tuning (ULMFit)



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Figure 1: ULMFiT consists of three stages: a) The LM is trained on a general-domain corpus to capture general features of the language in different layers. b) The full LM is fine-tuned on target task data using discriminative fine-tuning ('*Discr*') and slanted triangular learning rates (STLR) to learn task-specific features. c) The classifier is fine-tuned on the target task using gradual unfreezing, '*Discr*', and STLR to preserve low-level representations and adapt high-level ones (shaded: unfreezing stages; black: frozen).

https://medium.com/airesearch-in-th/wangchanberta-%E0%B9%82%E0%B8%A1%E0%B9%80%E0%B8%94%E0%B8%A5%E0%B8%9B%E0%B8%A3%E0%B8%B0%E0%B8%A1%E0%B8%A7%E0%B8%A5%E0%B8%9C%E0%B8%A5%E0%B8%A0%E0%B8%B2%E0%B8%A9%E0%B8%B2%E0%B9%84%E0%B8%97%E0%B8%A2%E0%B8%97%E0%B8%B5%E0%B9%88%E0%B9%83%E0%B8%AB%E0%B8%8D%E0%B9%88%E0%B9%81%E0%B8%A5%E0%B8%B0%E0%B8%81%E0%B9%89%E0%B8%B2%E0%B8%A7%E0%B8%AB%E0%B8%99%E0%B9%89%E0%B8%B2%E0%B8%97%E0%B8%B5%E0%B9%88%E0%B8%AA%E0%B8%B8%E0%B8%94%E0%B9%83%E0%B8%99%E0%B8%82%E0%B8%93%E0%B8%B0%E0%B8%99%E0%B8%B5%E0%B9%89-d920c27cd433

Image by Phannisa Nirattiwongsakorn

WangchanBERTa โมเดลประมวลผลภาษาไทยที่ใหญ่และก้าวหน้าที่สุดในขณะนี้

VISTEC-depa AI Research Institute of Thailand Follow

Jan 24 · 5 min read

เราใช้เวลากว่า 3 เดือนในการเทรนโมเดลให้ loss ลดลงมาในระดับที่ 2.592 (perplexity = 13.356) ณ step ที่ 360,000 จากทั้งหมด 500,000 steps ณ วันนี้โมเดลก็ยังถูกเทรนอย่างต่อเนื่องในศูนย์วิจัยที่วังจันทร์ จึงเป็นไปได้ว่าเราจะได้โมเดลที่มีประสิทธิภาพดียิ่งกว่ามาใช้ในอนาคต

Benefit of Pre-trained Word Embedding: Adaptation (1) on parsing (blue vs red)

- Weights from word embedding layer can be used as pre-trained weights in other NLP applications

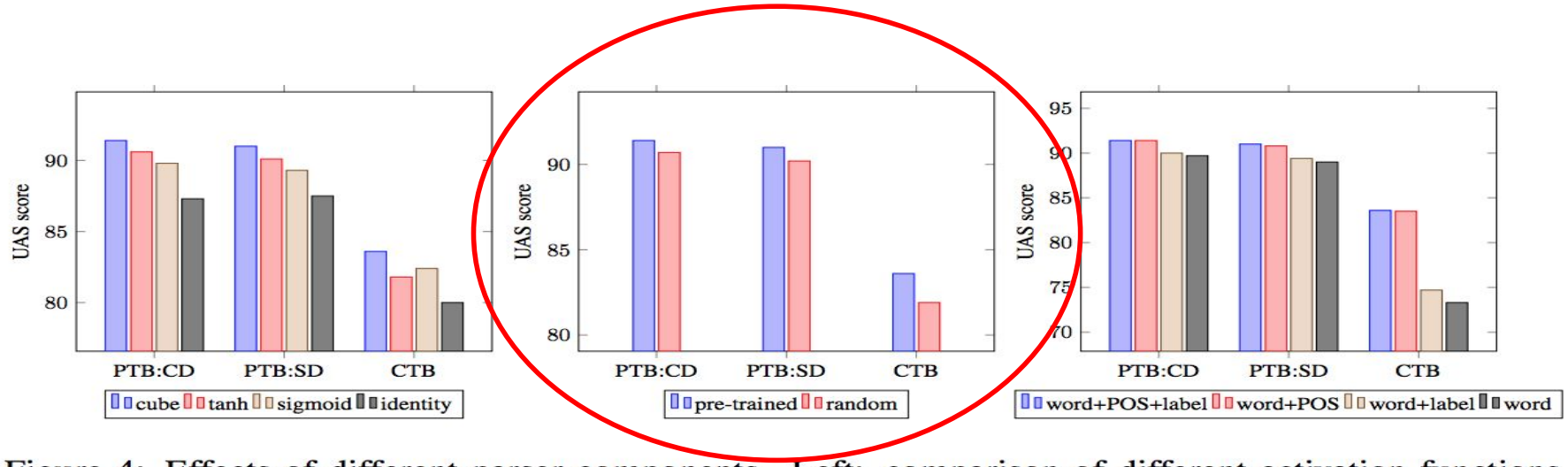
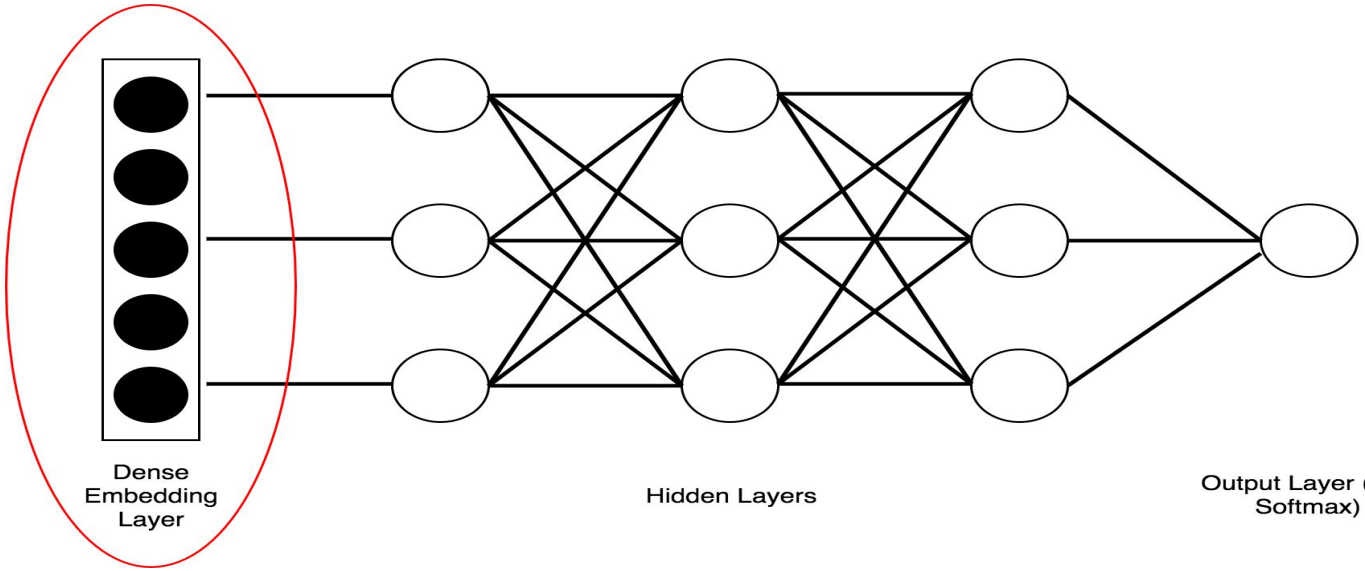


Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

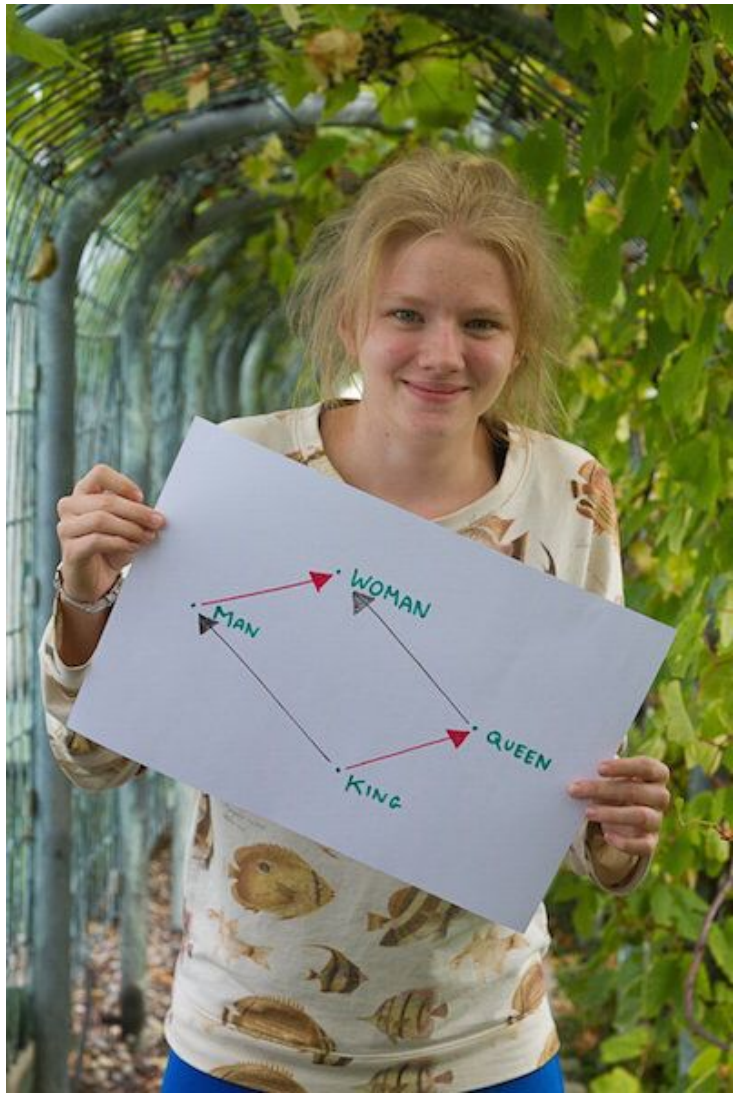
Image reference: Chen, Danqi, and Christopher Manning. "A fast and accurate dependency parser using neural networks." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014. APA

Adaptation (2)

- Classification model with pre-trained word embedding
- In your homework ☺, you will be asked to create a classification model using pre-trained weights from the skip-gram model



Learned Semantic-Syntactic Relationships



- Word Embedding can also capture semantic & syntactic relationships between words
- king – queen = man – woman
- We can ask "what is the word that is similar to **king** in the same sense as **woman** is similar to **man** ?"
- X = king-man+woman
- We then search for word that is closest to X (cosine distance)

Image reference: http://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html

Compositionality

- Now, we know how to create a dense vector representation for a word
  - What about larger linguistic units? (e.g. phrase, sentence)
- We can combine smaller units into a larger unit

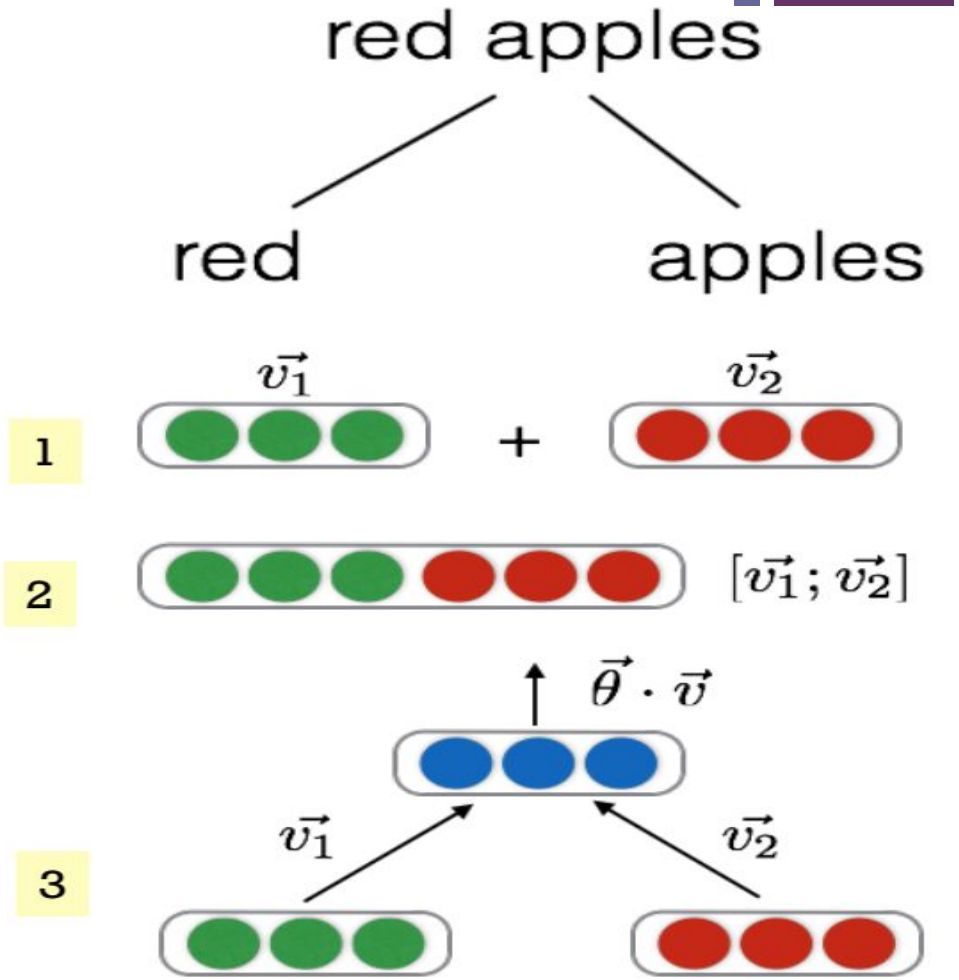


Image ref: Prof. Regina Barzilay , NLP@MIT

Part 4: Word2Vec Evaluation

Extrinsic

Intrinsic

Evaluation for word vectors (1)

- Extrinsic Evaluation:
  - Use pre-trained word vectors to initialize or concatenate as extra features
  - Then evaluate on real tasks (**other** tasks) (e.g. Part-of-speech tagging, Named entity recognition, sentimental analysis, etc.)

Evaluation for word vectors (2)

- 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

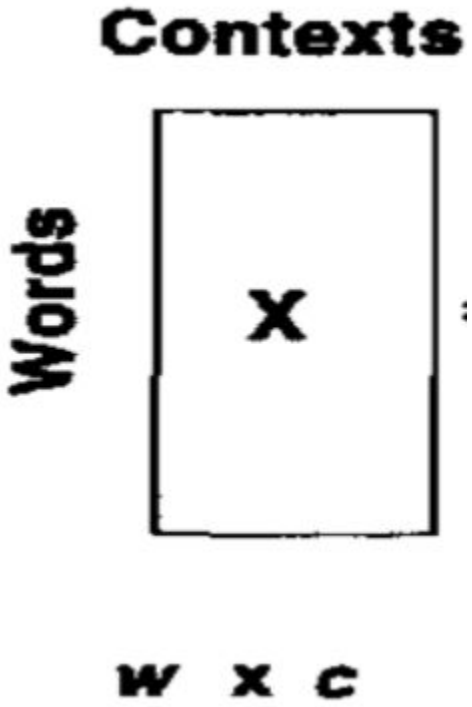
Evaluation for word vectors (3)

- Intrinsic Evaluation
- Tasks:
  - **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 vectors are clustered, then measure the purity of cluster based on the labeled dataset
  - **Selectional Preference (word function)**: The goal is to determine how typical a noun is for a verb either as a subject or as an object (e.g., people eat, but we rarely eat people)

Reference: Schnabel, Tobias, et al. "Evaluation methods for unsupervised word embeddings." EMNLP. 2015

Summary

- One-hot vectors do not represent similarity between words
- Distributional representations can capture similarity between words
- Word2Vec is faster than SVD; pre-trained Word2Vec parameters are available online.
- Two main Word2Vec models:
  - Skip-gram
  - CBOW
- Efficient training methods
  - Hierarchical Softmax
  - Negative Sampling
- Benefit
  - Adaptation
  - Learned Semantic-Syntactic Relationships
  - Compositionality
- Word2Vec evaluation: extrinsic evaluation and intrinsic evaluation



Part 5: Advanced Topics

Doc2Vec

Limitations of word embeddings and possible improvements

Doc2Vec (Le and Mikolov 2014)
For information retrieval (IR)

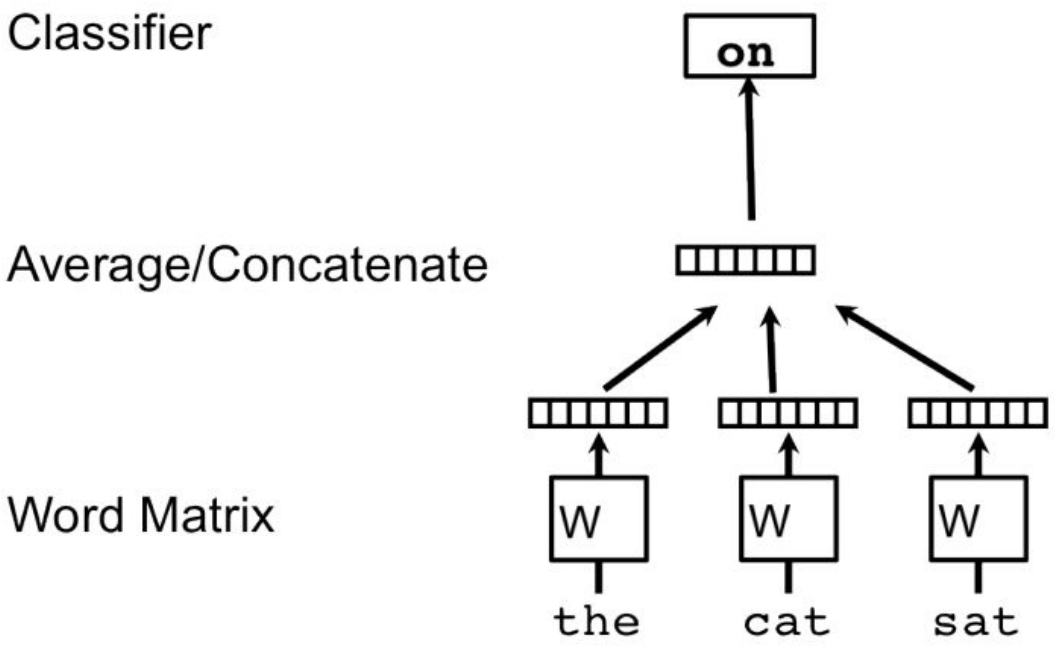
- Document representation
  - also work with variable-length pieces of texts, such as sentences, paragraphs, documents, etc.

Distributed Representations of Sentences and Documents

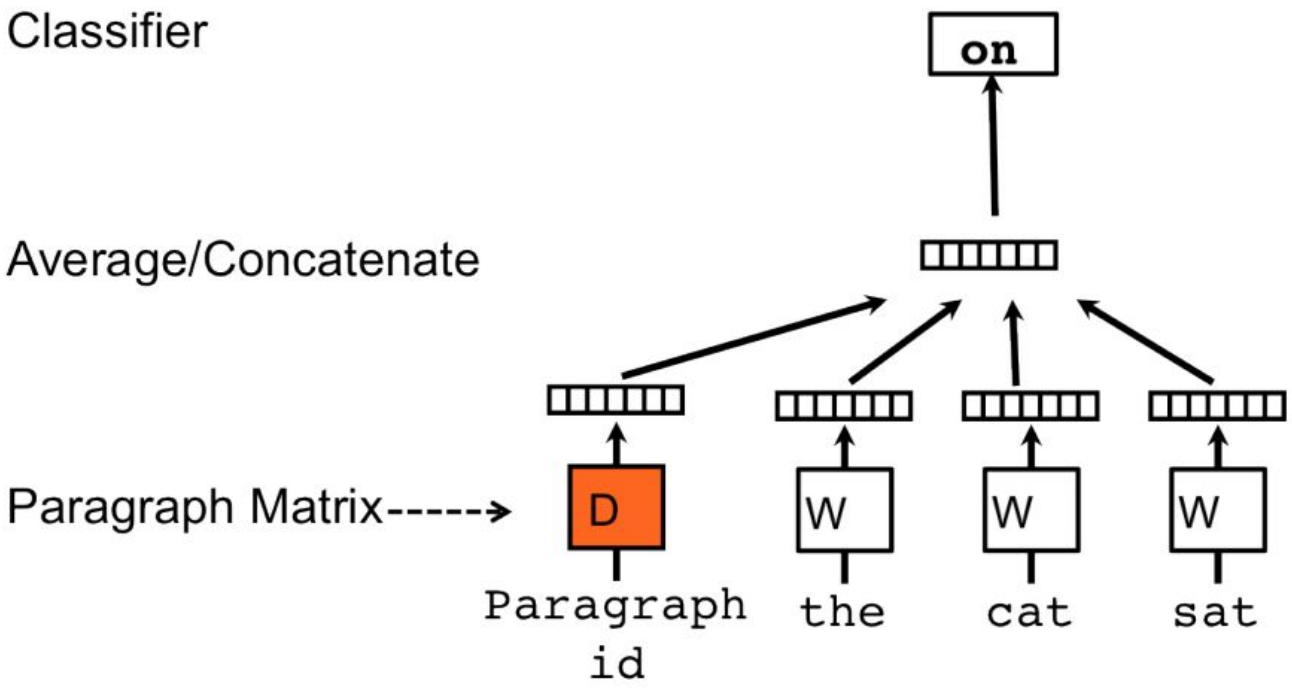
Quoc Le QVL@GOOGLE.COM
Tomas Mikolov TMIKOLOV@GOOGLE.COM
Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043

Doc2Vec (cont.)

A framework for learning word vector

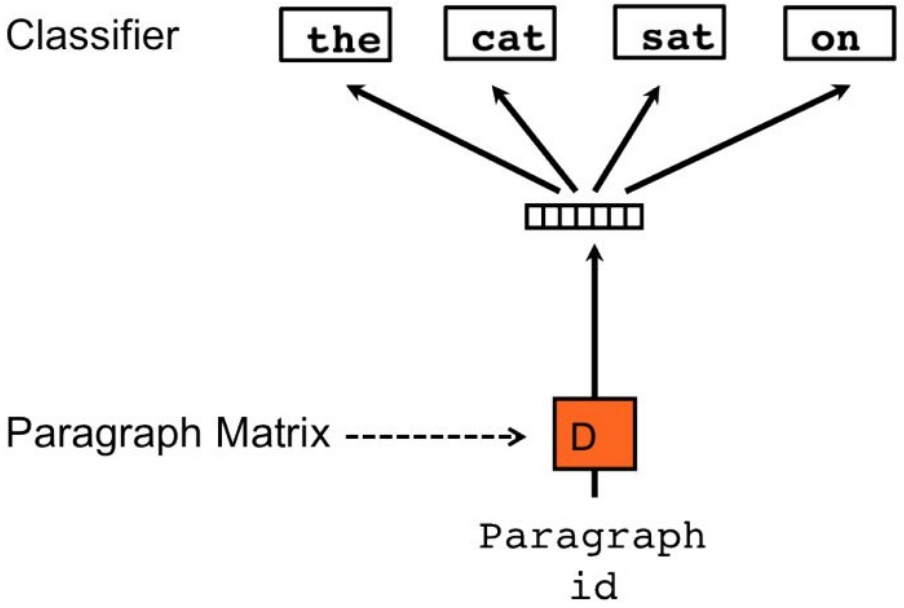


A framework for learning document vector



LM (similar to **CBOW**)

Doc2Vec (cont.)



similar to **skip-gram**

Table 1. The performance of our method compared to other approaches on the Stanford Sentiment Treebank dataset. The error rates of other methods are reported in (Socher et al., 2013b).

Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	**12.2%**	**51.3%**

Unknown words (Problem: OOV)

- Why do we need a UNK token?
  - Not all words are available in training data
  - Larger vocab size = more memory/computation time

- Common ways:
  - Frequency threshold
    - 1) if word count <= 1, then UNK
    - 2) Rank threshold: only include top 50,000 words, the rest are UNK

Limitations of word embeddings

- Problem1) Sensitive to subtle/superficial morphological differences → Solution: Subwords
  - student vs students
- Problem2) Only has one representation for all unknown words → Solution: Subwords

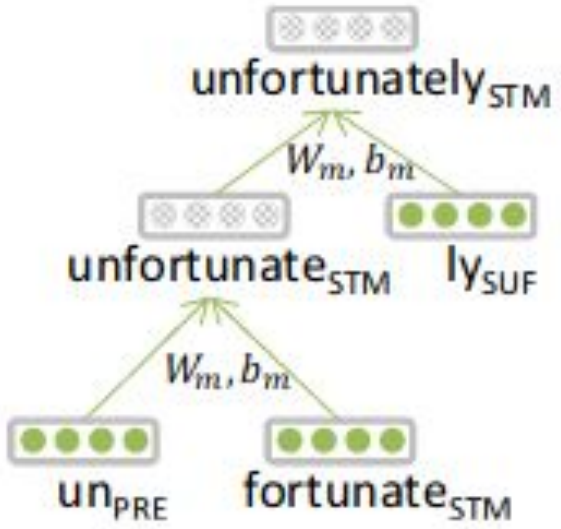
- Problem3) Insensitive to context → Solution: Contextualized word representations
  - baseball bat  vs bat can fly 

- Problem4) Interpretability → Solution: Sparse Embeddings
  - what does each dimension of a word embeddings represent?
- Problem5) Bias → Solution: debias algorithm
  - racial discrimination and gender bias

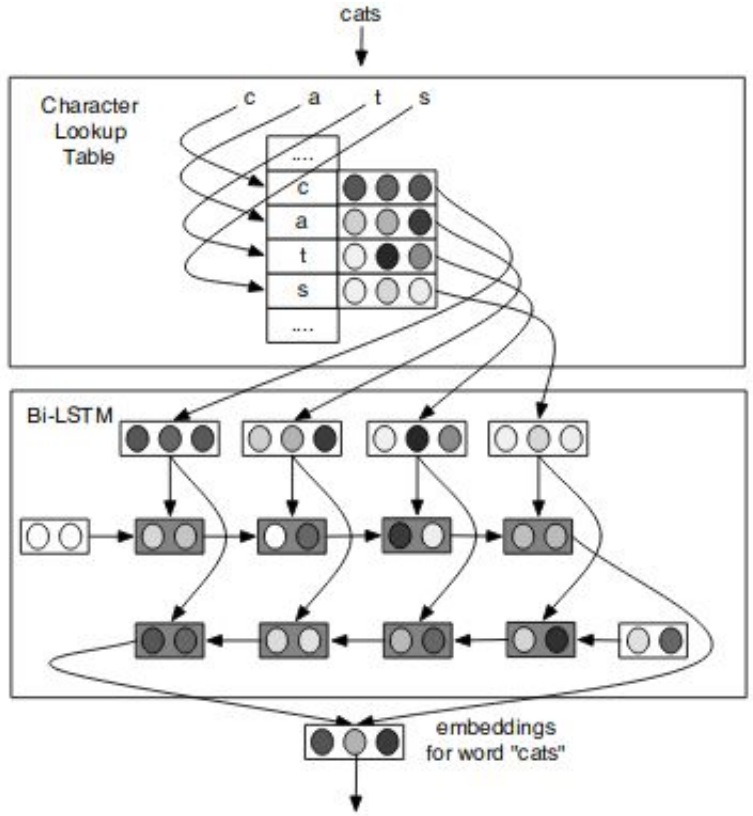
Solutions 1,2: Subwords Embeddings

- Morpheme-based (Luong et al. 2013)

recursive neural net!



- Character-based (Ling et al. 2015)
  - compositional character to word model



Solutions 1,2: Subwords Embeddings (cont.)

- Character n-grams (Wieting et al. 2016)
- add vectors of character n-grams together followed by an elementwise nonlinearity
- Best version uses <2,3,4,5,6>-grams

sweet → <sw, swe, we, wee, ee, eet, et >

$$g_{\text{CHAR}}(x) = h\left(\boldsymbol{b} + \sum_{i=1}^{m+1} \sum_{j=1+i-k}^{i} \mathbb{I}\left[x_j^i \in V\right] W^{x_j^i}\right)$$

- 1) Byte-Pair Encoding (BPE)
  - Byte-Pair Encoding (BPE) was introduced in Neural Machine Translation of Rare Words with Subword Units (Sennrich et al., 2015).
  - Used in GPT-2, Roberta
  - For example, aaabdaaabac
  - **Z**abd**Z**abac
    - Z=aa
  - Z**Y**dZ**Y**ac
    - Y=ab
    - Z=aa
  - **X**d**X**ac
    - X=ZY
    - Y=ab
    - Z=aa
  - reference: https://en.wikipedia.org/wiki/Byte_pair_encoding

1) BPE example

พราว และ ขาว นั่ง บน ราว ดู ข่าว คราว บน ดาว

พรx และ ขx นั่ง บน รx ดู ข่xครx บน ดx

x=าว

พy และ ขx นั่ง บน y ดู ข่x คy บน ดx

x=าว

y=รx

พy และ ขx นั่ง z y ดู ข่x คy z ดx

x=าว

y=รx

z=บน

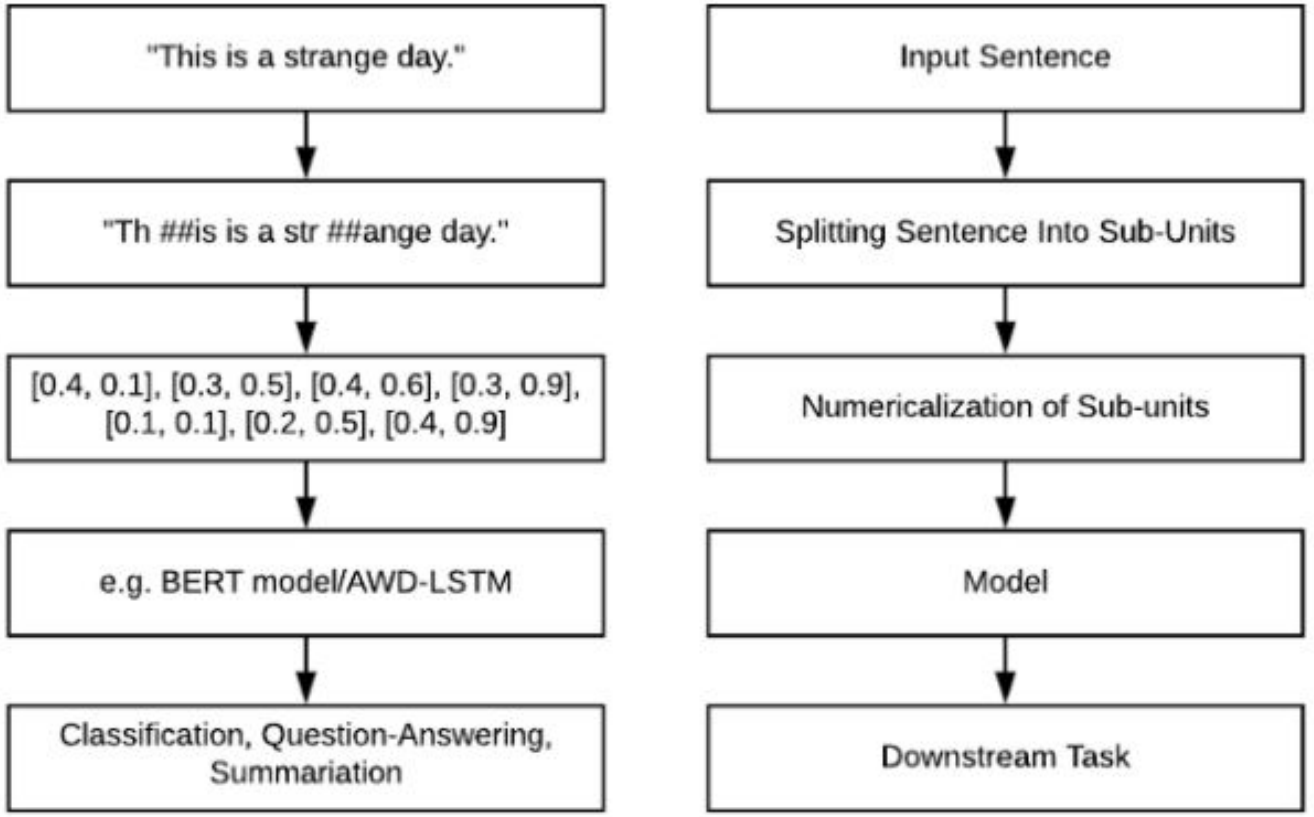
2) WordPiece/SentencePiece

- Google NMT(GNMT)uses a variant of this
  - V1: wordpiece model
  - V2: sentencepiece model

- Rather than char n-gram count, uses a greedy approximation to maximizing language model log likelihood to choose the pieces
  - Add n-gram that maximally reduces perplexity

2) WordPiece

- WordPiece is the subword tokenization algorithm used for BERT, DistilBERT, and Electra.
- There are 2 types of tokens: start token (no ##), and continuing token (##)



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

2) WordPiece (cont.)

- The algorithm was outlined in Japanese and Korean Voice Search (Schuster et al., 2012) and is **very similar to BPE.**
- In contrast to BPE, WordPiece does not choose the most frequent symbol pair, but the one that maximizes the likelihood of the training data once added to the vocabulary.
- E.g. "u", followed by "g" would have only been merged if the probability of "ug" divided by "u", "g" would have been greater than for any other symbol pair. Intuitively, WordPiece is slightly different to BPE in that it evaluates what it loses by merging two symbols to make ensure it's worth it. (calculation with normalization)

https://huggingface.co/transformers/tokenizer_summary.html

3) SentencePiece

- SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing (Kudo et al., 2018)
- It aims to solve 2 issues.
- Issue1: Which one should a correct denomalization?
  - Tokenize("World.") == Tokenize("World .")
- Issue2: End-to-End to avoid the need of language-specific tokenization.

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

Table 3: WangchanBERTa model names

3) SentencePiece (cont.)

- Wordpiece model tokenizes inside words
- Sentencepiece model works from raw text (across words)
  - White space is retained as special token (_) and grouped normally
  - You can reverse things at end by joining pieces and recoding them to spaces

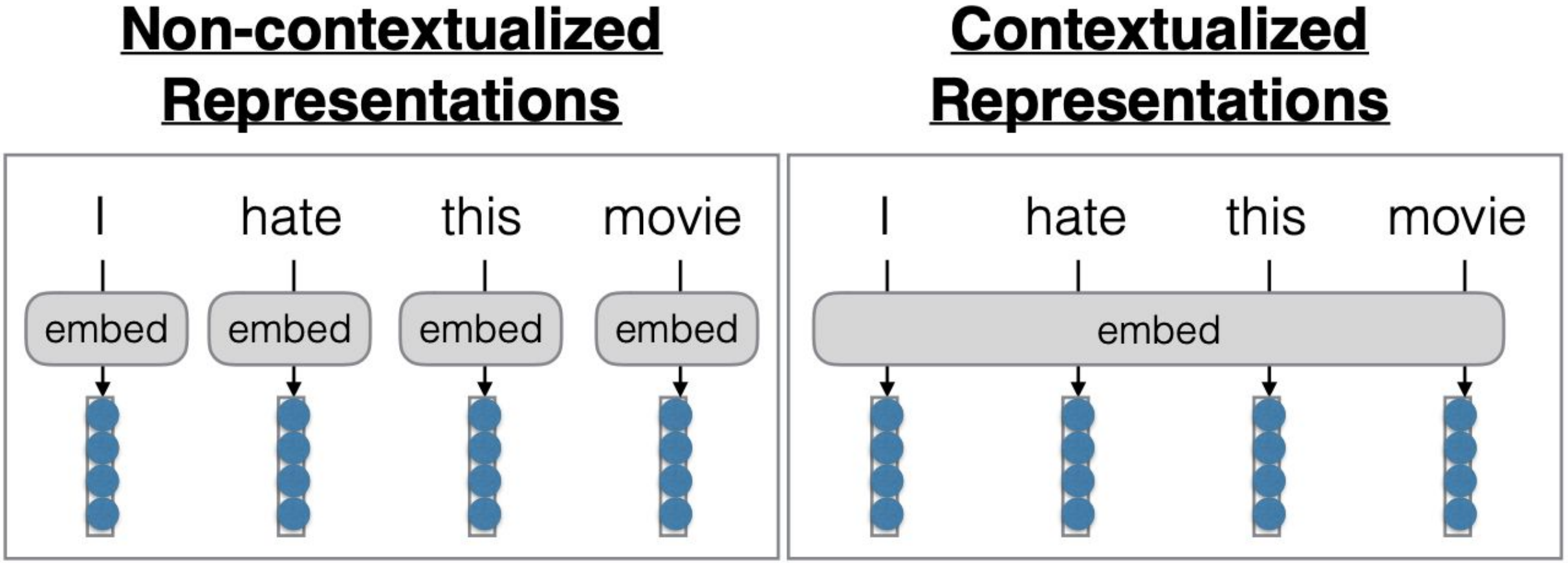
[Hello] [World] [.]

Hello_World.

```
detokenized = ''.join(pieces).replace('_', ' ')
```

[Hello] [_Wor] [ld] [.]

Solution3: Contextualized word representation



reference: Graham Neubig, http://phontron.com/class/nn4nlp2020/assets/slides/nn4nlp-08-wordemb.pdf

Solution3: Contextualized word representation (cont.)

- The representation for each word depends on the entire context in which it is used: (these are state-of-the-art models)
  - ELMo[pretrained weights and code are available]
  - BERT [pretrained weights and code are available]

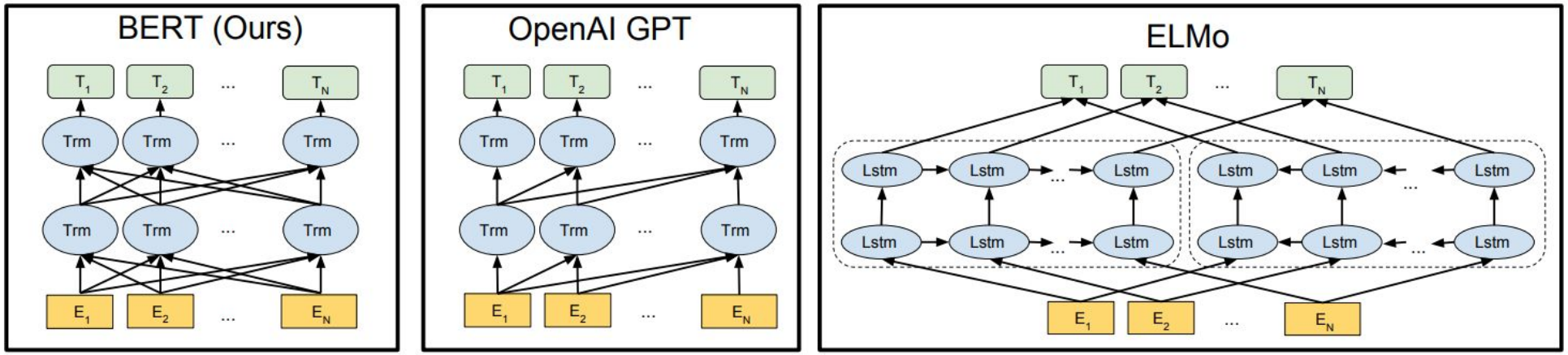


Figure 1: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTM to generate features for downstream tasks. Among three, only BERT representations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

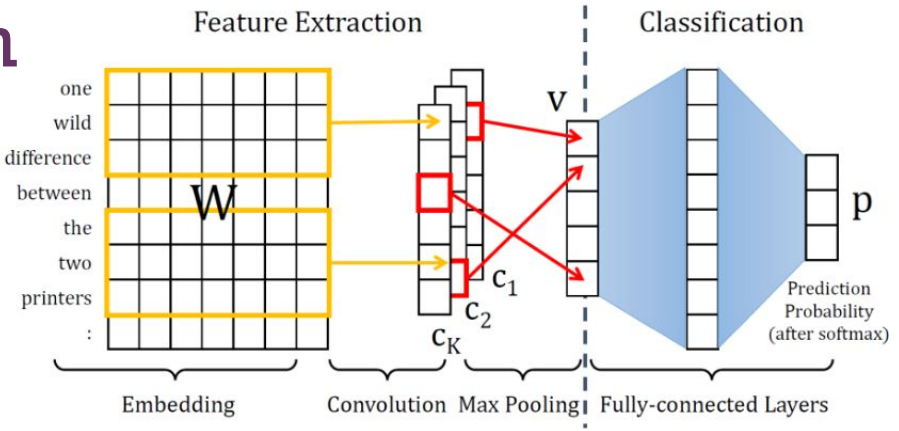
Solution4: Sparse Embeddings (Murphy et al. 2012) [interpretation issue]

- Increase the **interpretability of each dimension** of a word embedding
  - constrained matrix factorization technique
    - Non-Negative Sparse Embedding (NNSE)
  - add sparsity constraint
    - most of the dimensions are zeros
    - therefore increase the information content of non-zero dimensions for each word

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

Neural based interpretation

Gradient Analysis



- Sensitivity Analysis (Arras et al., 2016)
  - The effect of w_i towards the prediction of class j (E_{j,w_i})

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

- Grad-CAM-Text (Lertvittayakumjorn & Toni, 2019)
  - $E_{j,N(k)}$: the effect of an n-gram selected by the k^{th} filter towards the prediction of class j: $E_{j,N(k)} = \left|\max\left(\frac{\partial FC(\boldsymbol{v})_j}{\partial \boldsymbol{v}_k}, 0\right)\right| \times \boldsymbol{v}_k$
  - $E_{j,w_i} = \sum_k (\mathbb{I}[w_i \epsilon N(k)] \times E_{j,N(k)})$

Lertvittayakumjorn, Piyawat, and Francesca Toni. "Human-grounded Evaluations of Explanation Methods for Text Classification." *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019.

Solution5: Debiasing word embeddings (Bolukbasi et al. 2016) [gender bias issue]

- What if we want to use ML model to make a very important decision?
- Word embeddings reflect undesirable biases of text in the training corpus
  - Man:Woman as King:Queen (we want this) ✅
  - Man:Programmer as Woman:Homemaker (we don't want this) ❌
  - He:She as Professor:Associate_Professor (we don't want this) ❌

- Transform the space while preserving useful relationships
- Steps:
  - 1. Identify bias direction
  - 2. Neutralize: For every word that is not definitional, project to get rid of biases
  - 3. Equalize pairs

reference/suggested video: https://www.youtube.com/watch?v=fg8ijSPHyx0