Word Representations

Slides adapted from Assoc. Prof.Peerapon Vateekul, Ph.D.

Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

Outline

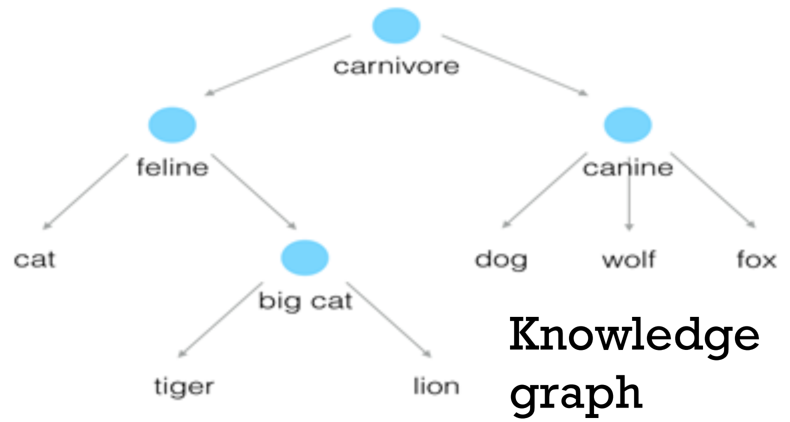
- **1) How to represent words?**
  - Symbolic vs distributional word representations
- **2) Distributional: Sparse vector** representations (discrete representation)
  - Term-document matrix
  - TF-IDF
- **3) Distributional: Dense vector representations**
  - Word2Vec
    - CBOW
  - Word2Vec training methods
  - Pre-trained vector representations
    - Adaptation
  - Compositionality

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

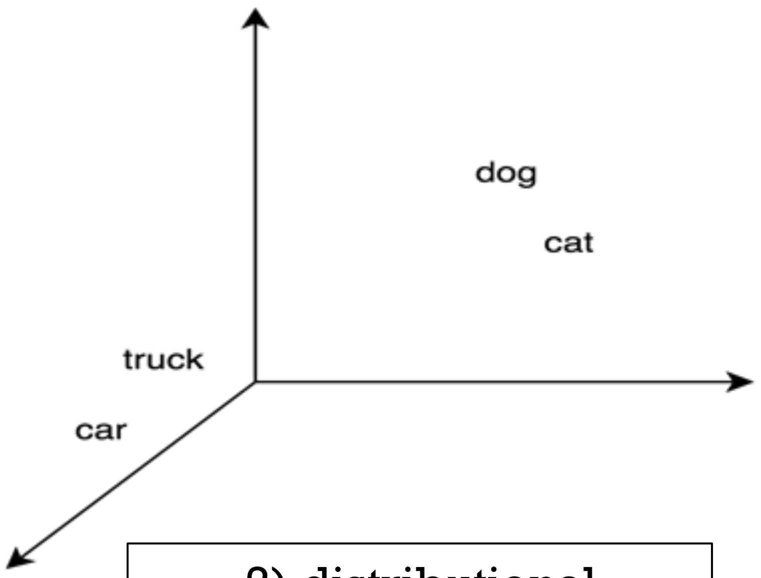


Knowledge graph

เสือ = [0 1 0 0 0 … 0 0]

One-hot model

1) symbolic

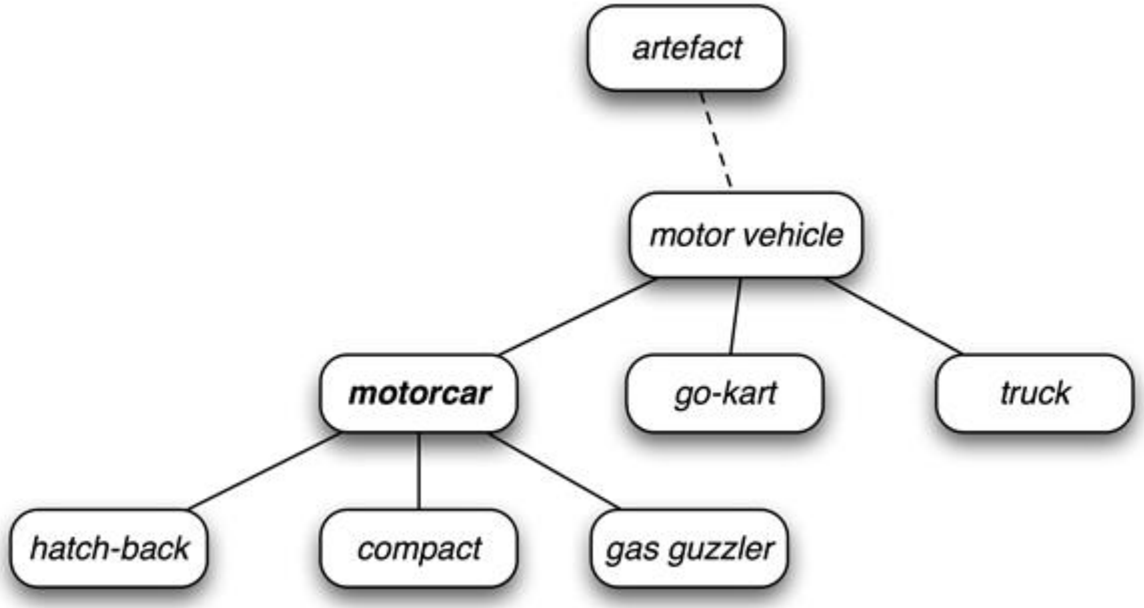


2) distributional

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.

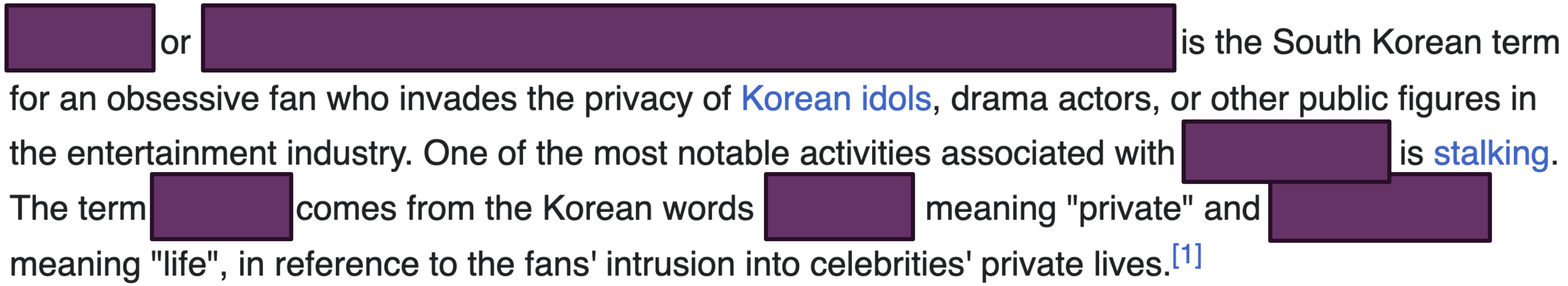
  - Drawback:
    - Cannot capture similarity between words

+

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!**



[redacted] or [redacted] is the South Korean term for an obsessive fan who invades the privacy of Korean idols, drama actors, or other public figures in the entertainment industry. One of the most notable activities associated with [redacted] is stalking. The term [redacted] comes from the Korean words [redacted] meaning "private" and [redacted] meaning "life", in reference to the fans' intrusion into celebrities' private lives.[1]

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
TF-IDF

Sparse vector representations in Distributional Representations

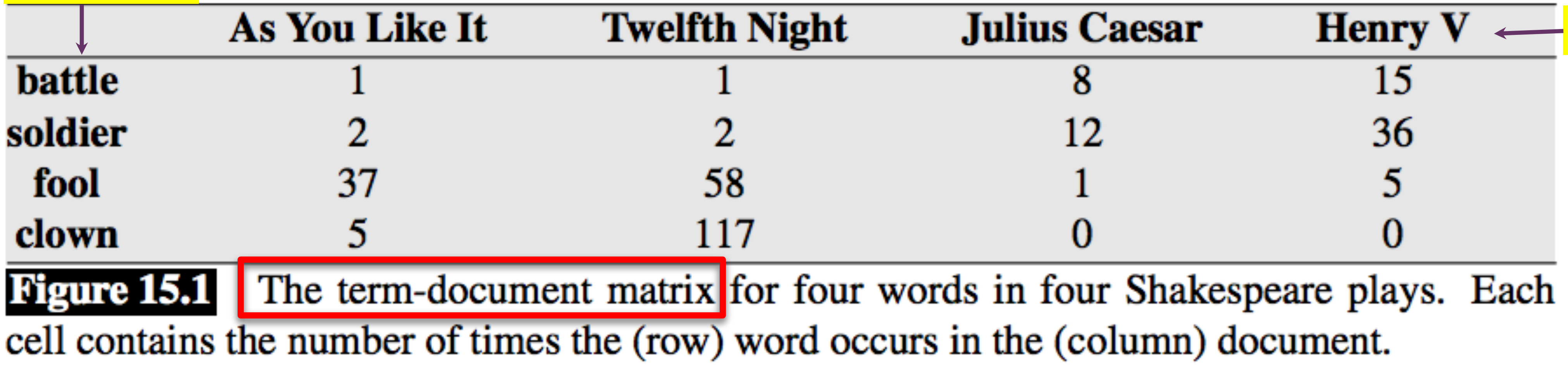
1. Term-Frequency (Raw Frequency)
2. Co-occurrence matrix
3. PPMI
4. TF-IDF

Sparse vector representations: 1 term-document matrix

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

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.

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)

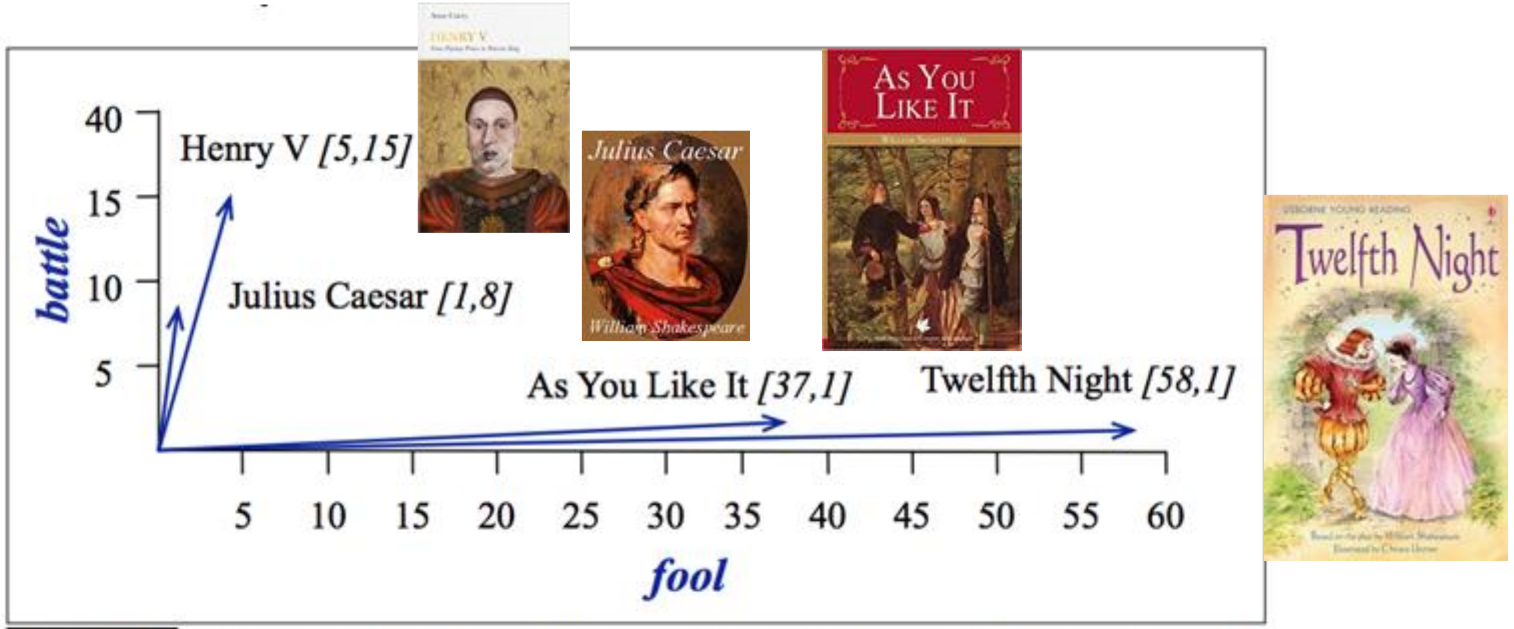
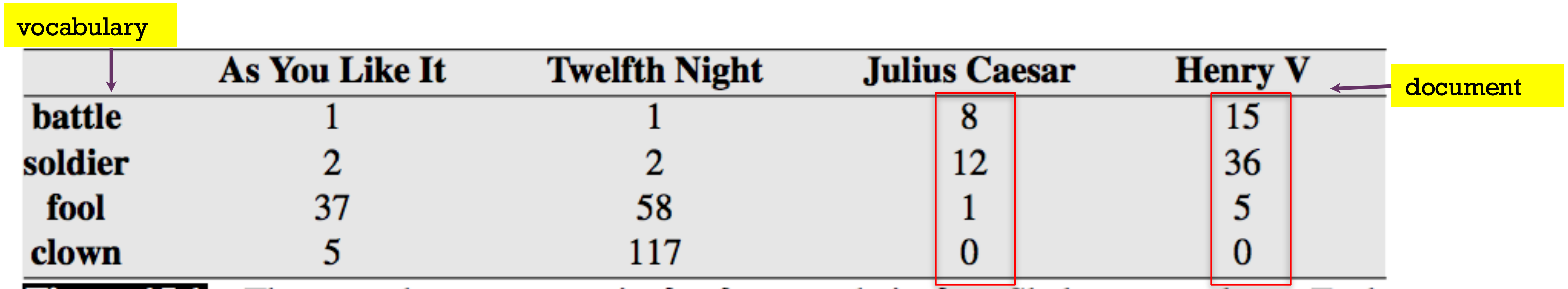


Figure 15.3 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

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
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.

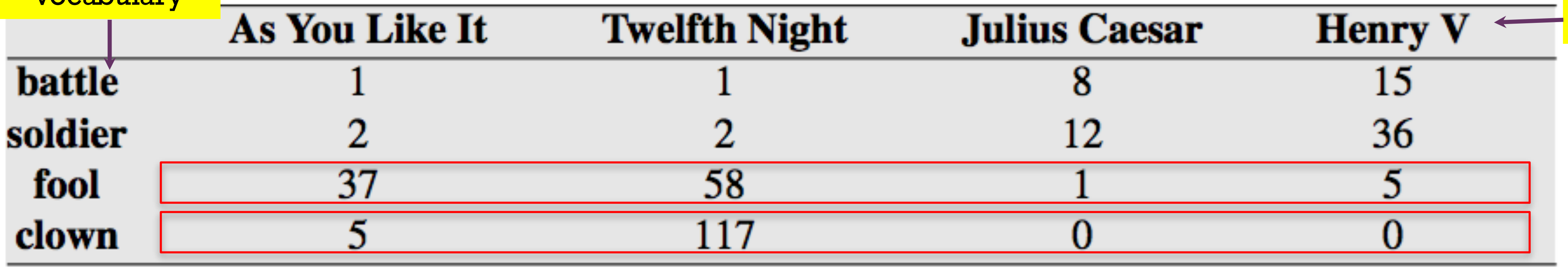
Reference: Jurafsky, Dan, and James H. Martin. Speech and language processing. 3rd edition draft, https://web.stanford.edu/~jurafsky/slp3/, August 2017

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

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

vocabulary

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.

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

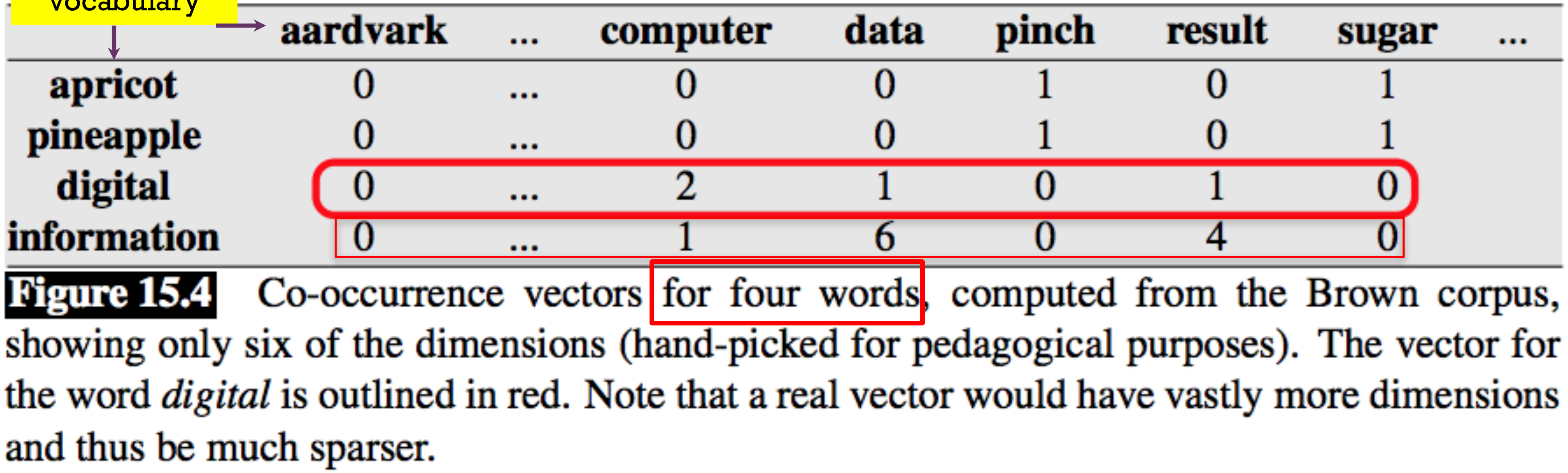
2

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

window = 4

vocabulary

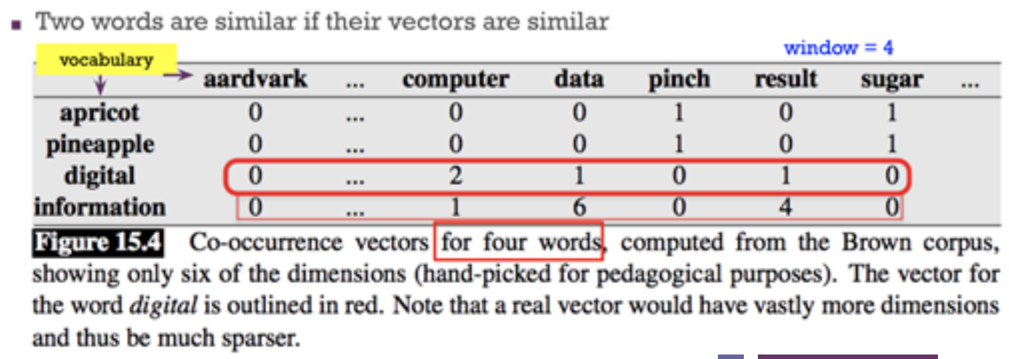


	aardvark	...	computer	data	pinch	result	sugar	...
apricot	0	...	0	0	1	0	1	
pineapple	0	...	0	0	1	0	1	
digital	0	...	2	1	0	1	0	
information	0	...	1	6	0	4	0	

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.

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)

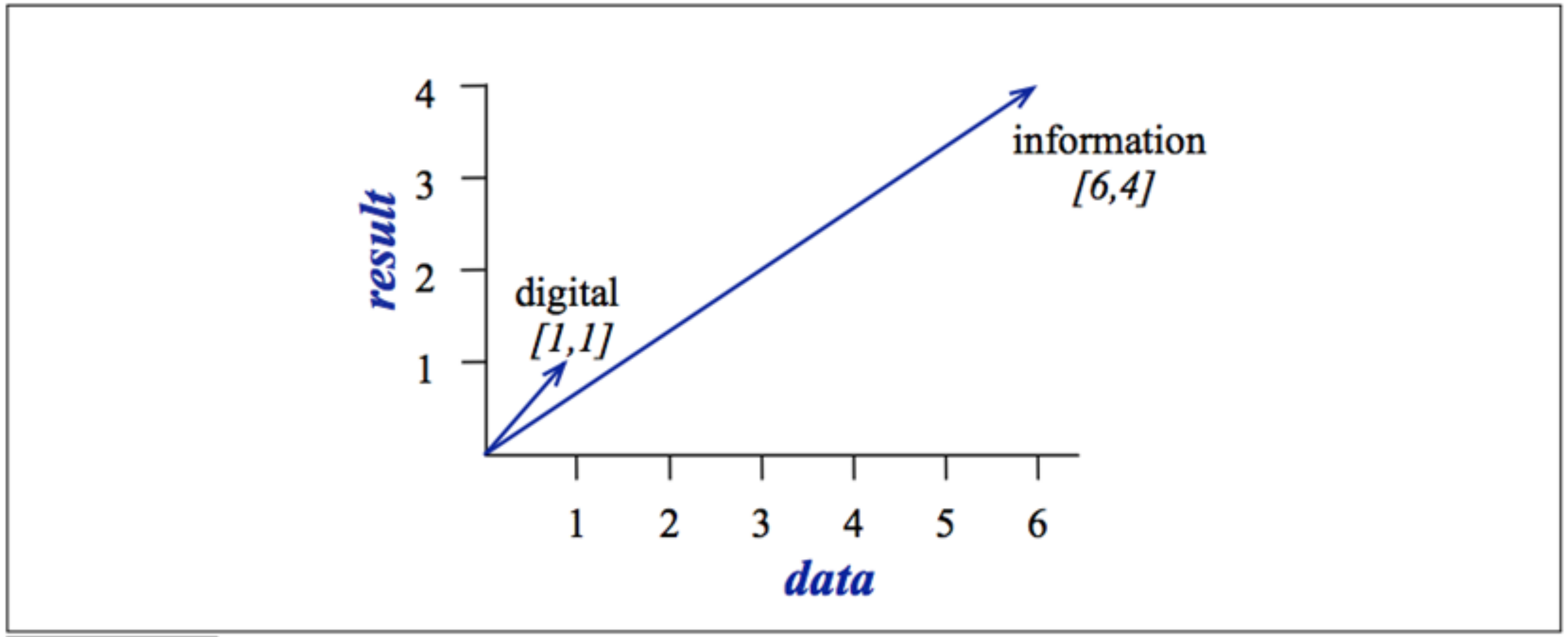


Figure 15.5 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *result*.

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

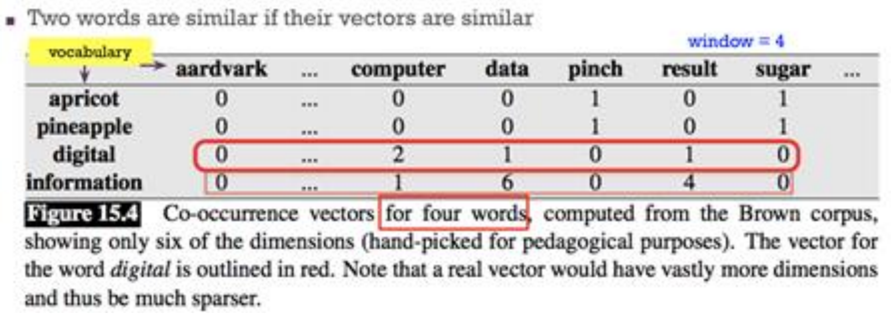
+

Sparse vector representations: 3 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
    - , 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



- 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	

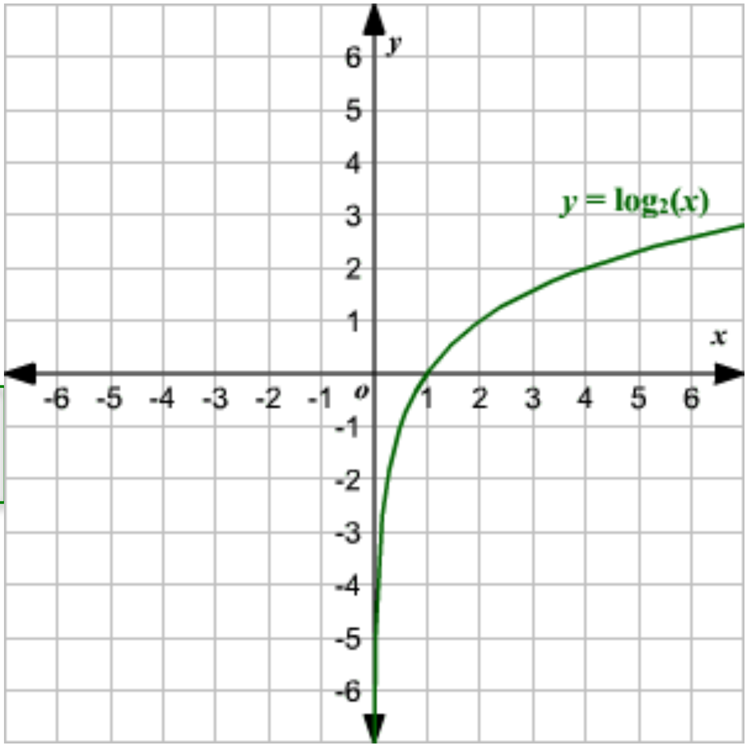
Figure 15.4 Co-occurrence vectors for four words, computed from the Brown corpus, showing only six of the dimensions (hand-picked for pedagogical purposes). The vector for the word *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Sparse vector representations:
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)



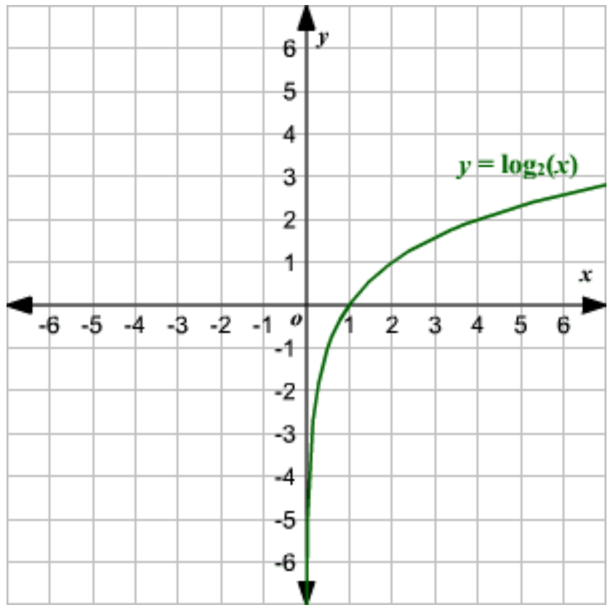
w – target word
c – context word
Do words "w" and "c" co-occur more than if they were independent?

+ : 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

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



word co-occurrence matrix

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	

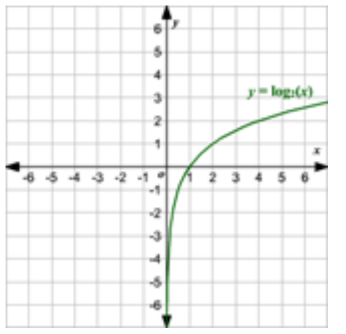
PPMI matrix

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

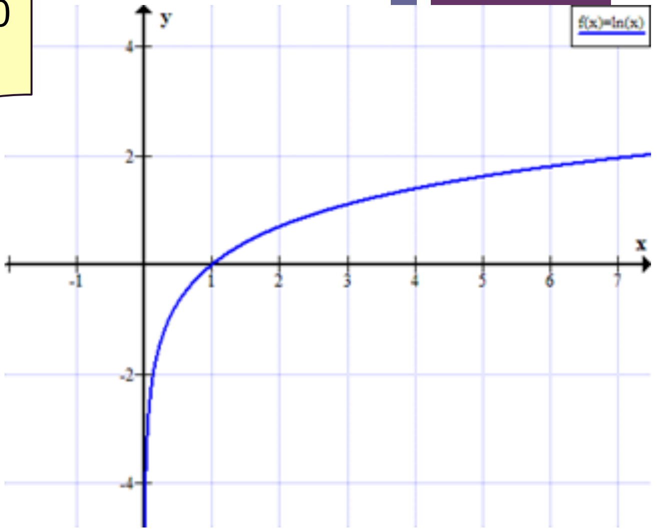
Need for normalization in TF

Doc1
cat = 5/10

Doc2
cat = 50/1000

- Term Frequency (TF) – per each document

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



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

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

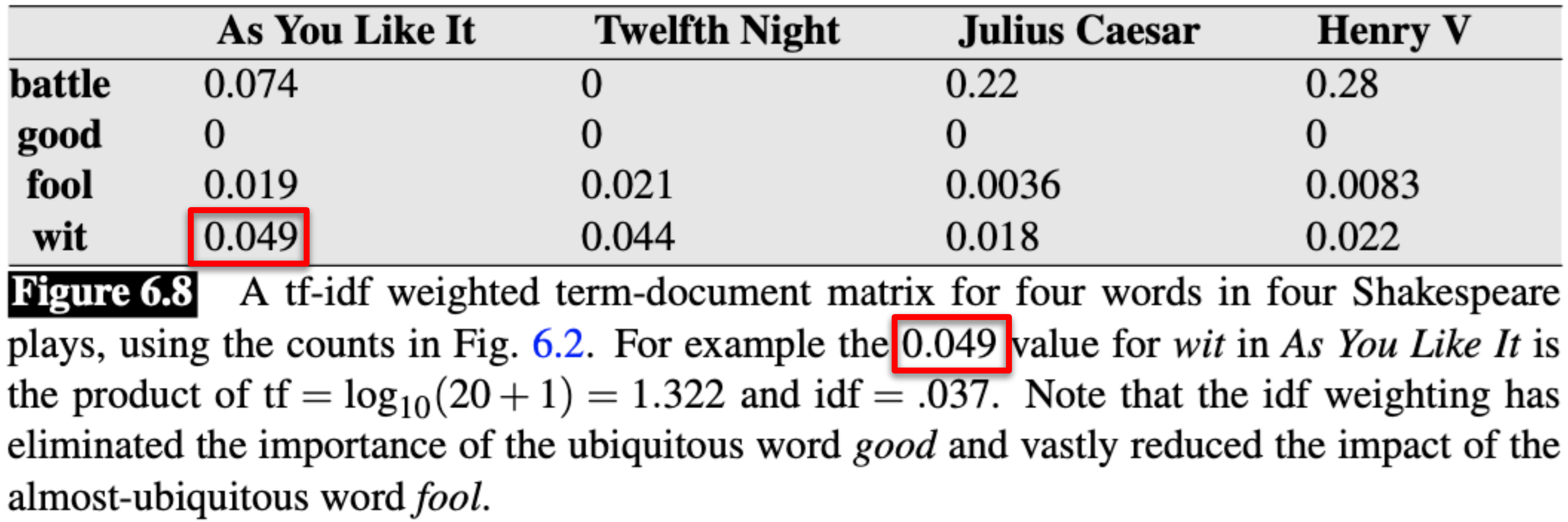
penalty score
i.e., a, an, the

- TF-IDF

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

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 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 (Best Matching)

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

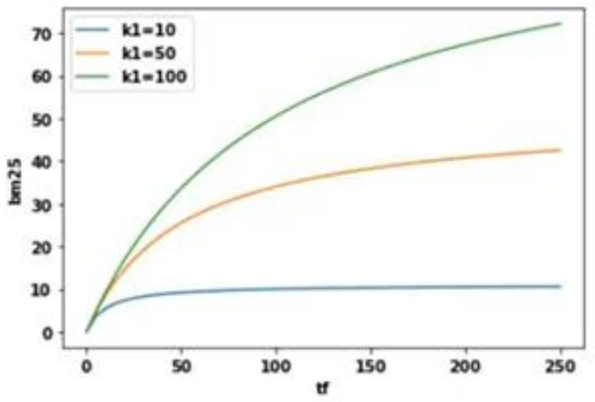
- k_1, b – parameters
- $dl(d)$ – length of document d
- dl_{avg} – average document length

RAG performance using different document representation

Model	MRR	R@1	R@5	R@10
ReQA SQuAD				
BM25	0.670	0.591	0.759	0.814
USE$_{qa}$	0.665	0.561	0.793	0.854
MPNET$_{qa}$	0.549	0.399	0.748	0.847
SGPT$_{5.8B}$	**0.783**	**0.699**	**0.887**	**0.926**
ReQA NQ				
BM25	0.529	0.378	0.723	0.797
USE$_{qa}$	0.582	0.447	0.751	0.826
MPNET$_{qa}$	0.628	0.439	**0.902**	**0.950**
SGPT$_{5.8B}$	**0.652**	**0.528**	0.807	0.856

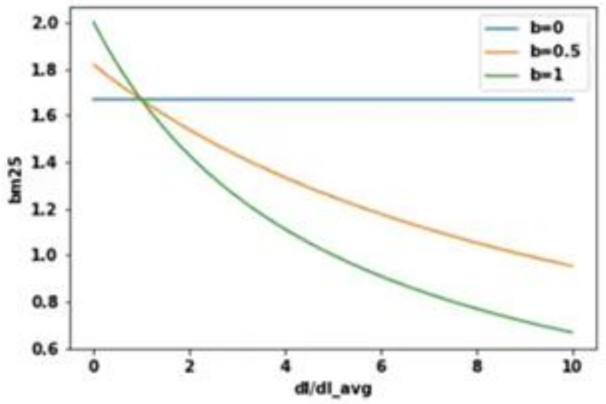
Table 2: The off-the-shelf sentence-level retrieval performance on the ReQA SQuAD and ReQA NQ task.

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?

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.

Part 3: Distributional: Dense vector representations

Dense vector representations

- Sparse vector representations such as Term-Document 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)

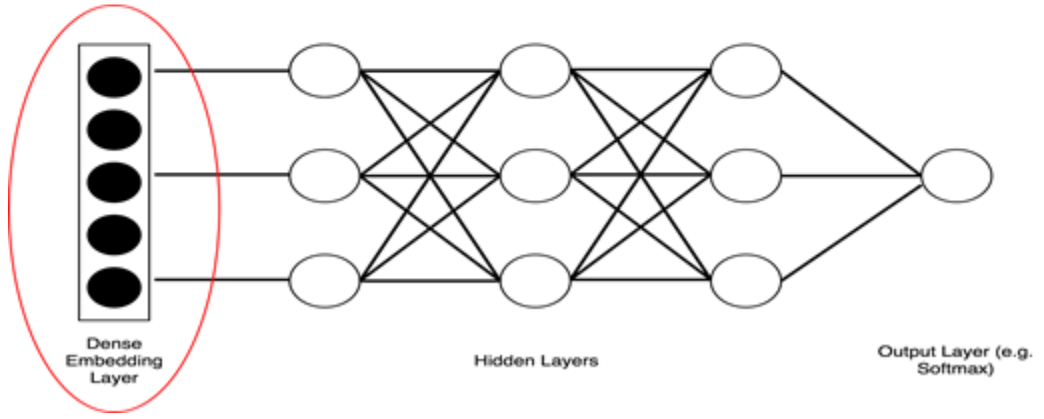
Dense vector representations (cont.)

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

1 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 language modeling task (e.g. next word prediction, cloze (predict missing words))
3. **Pre-train** on supervised task
   - e.g. train on POS tagging task and use it on other tasks

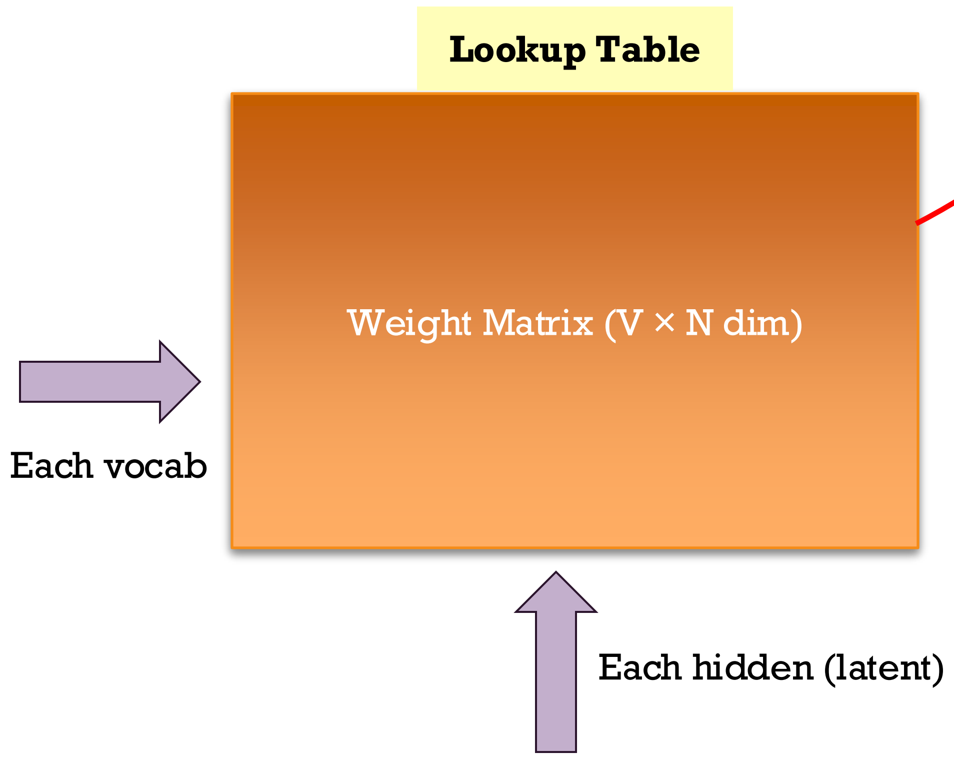


reference: Neubig (2020), https://www.youtube.com/watch?v=RRaU7pz2eT4

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:
  - Fast
  - Pre-trained word representations are available online!

Dense vector representations : neural networks (cont.)



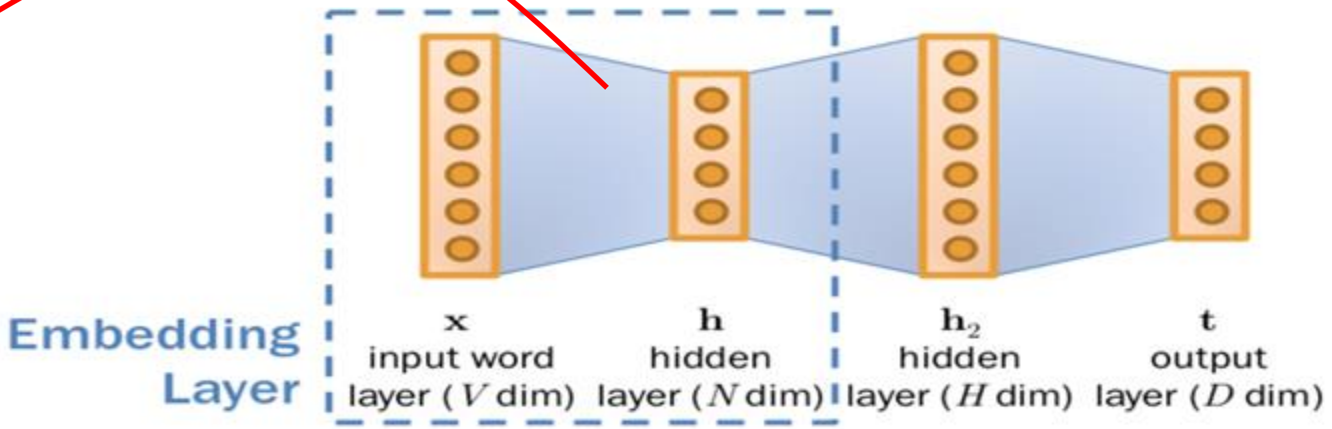
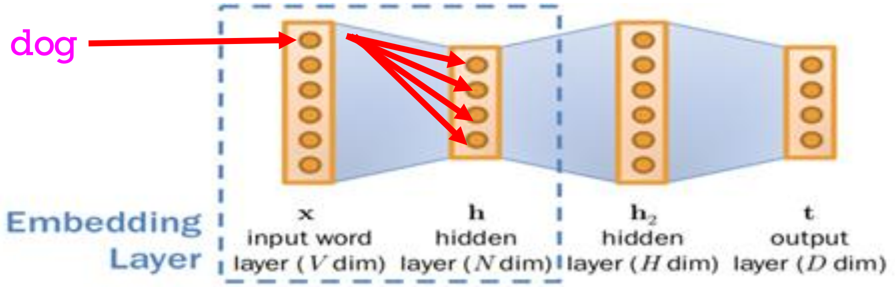


Image reference: Boonkwan, Prachya. “Word2Vec: When Language Meets Number Crunching”, https://goo.gl/hhA3hO, Feb 2017



Dense vector representations : neural networks (cont.)

- **CBOW** and **Skip-Gram** (not covered) 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.

negative words

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.

positive words

target word

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

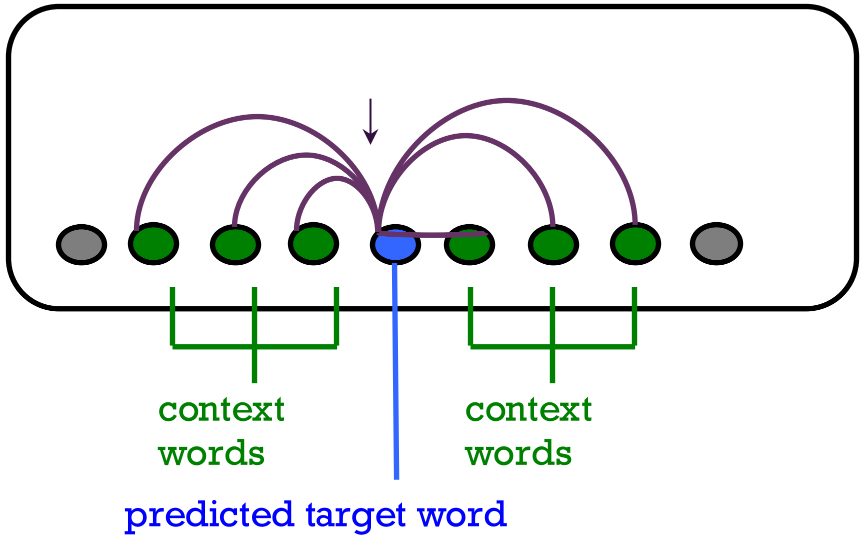
https://www.bbc.com/news/world-asia-china-51519055

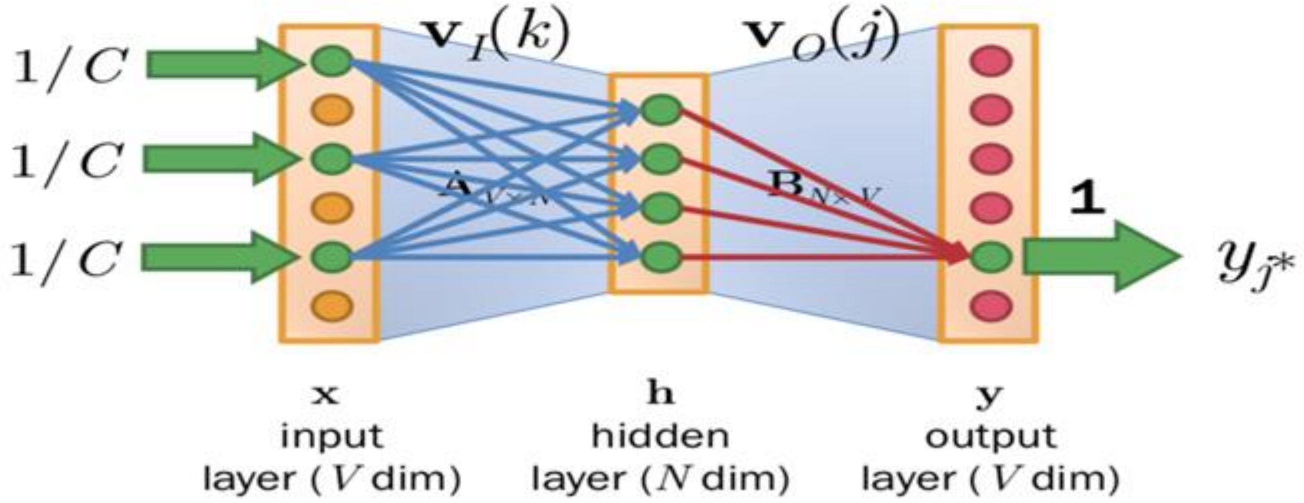
+

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.





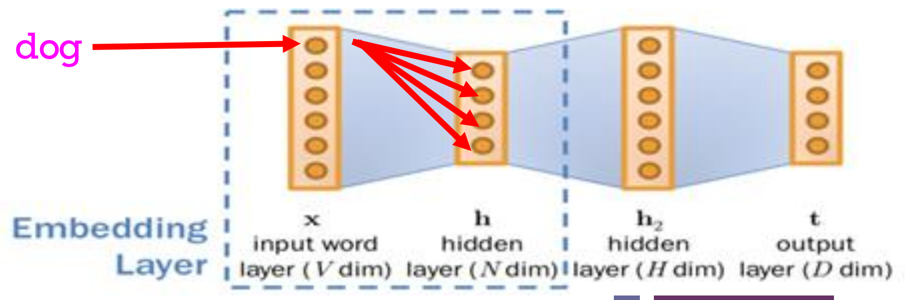
$$\mathbf{h}_{\text{ctx}} = \frac{1}{C}\sum_{q=1}^{C}\mathbf{v}_I(q)$$

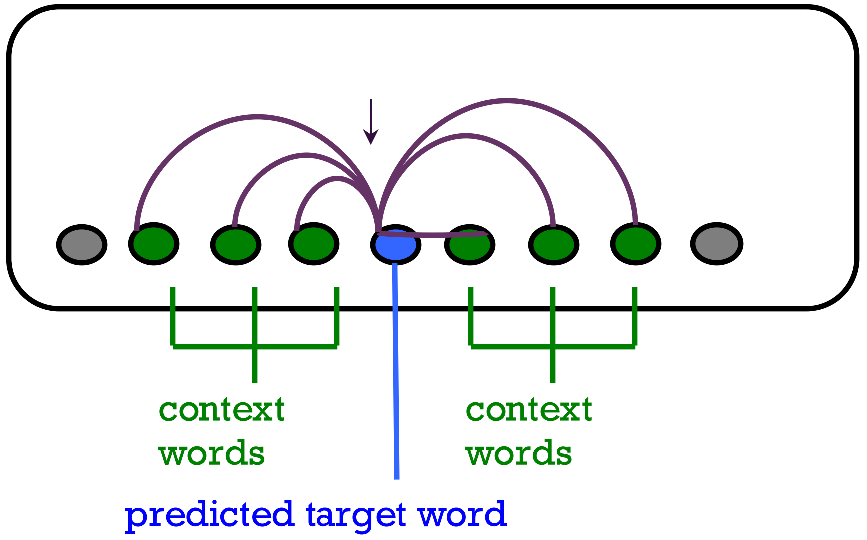
Image reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017

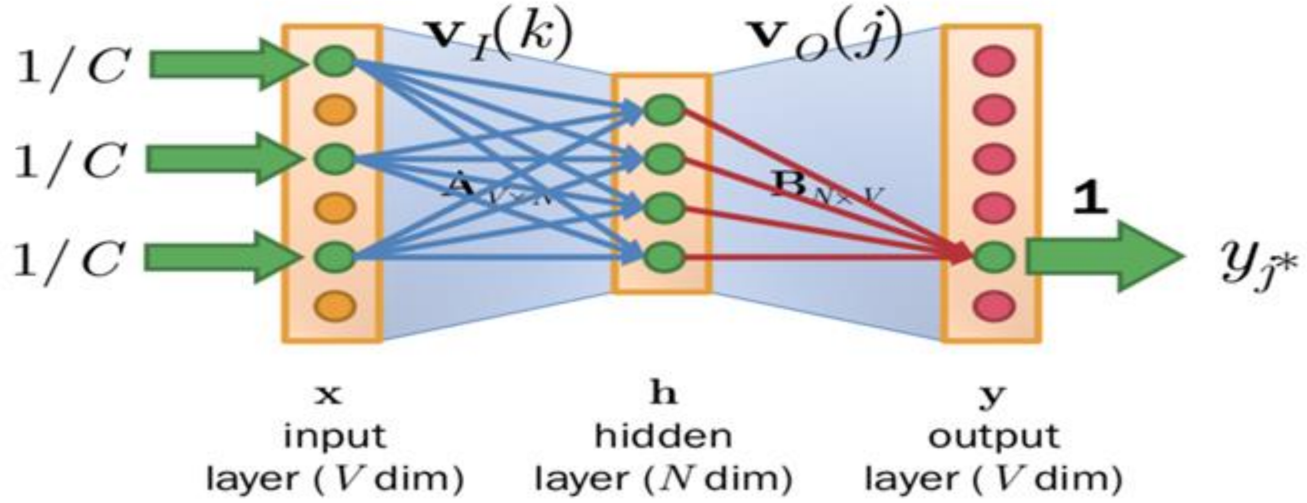
1) CBOW (cont.)

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







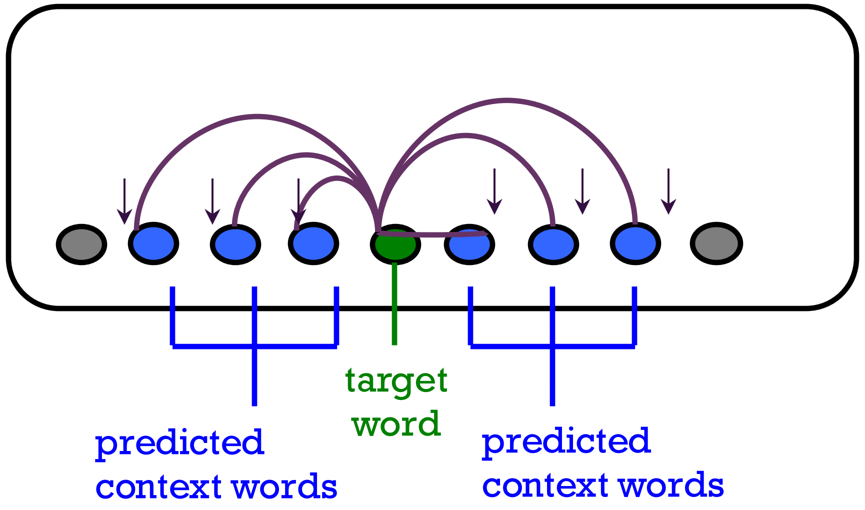
$$\mathbf{h}_{ctx} = \frac{1}{C}\sum_{q=1}^{C}\mathbf{v}_I(q)$$

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

2) Skip-gram

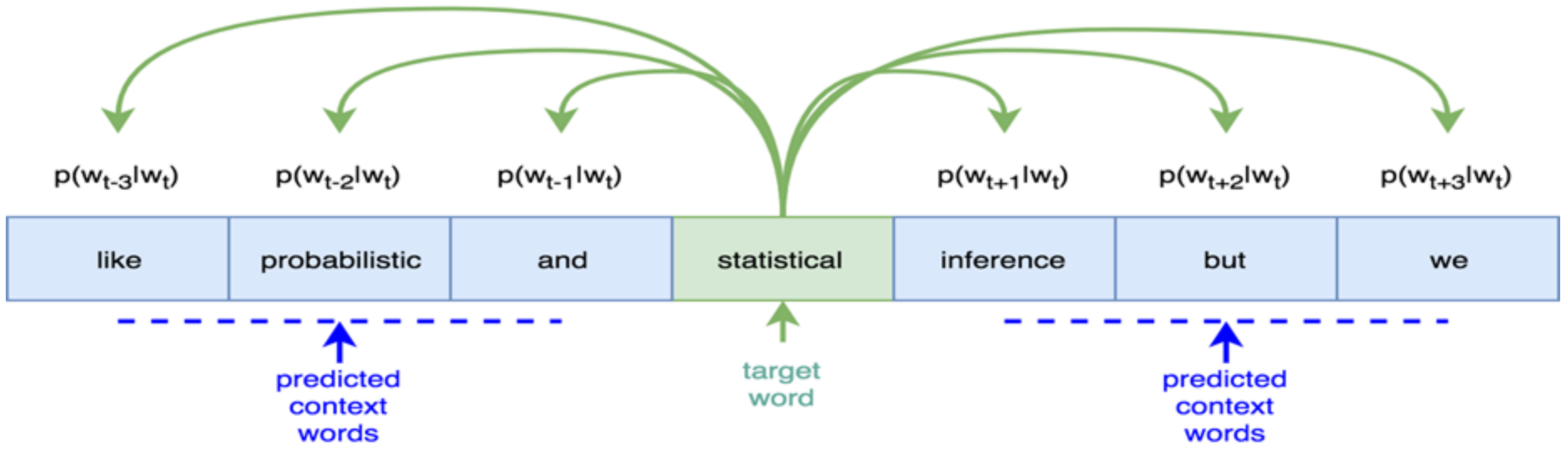
Outside of China there have been more than 500 cases in nearly 30 countries. Four people have died - in **France**, Hong Kong, the Philippines and Japan.

- 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



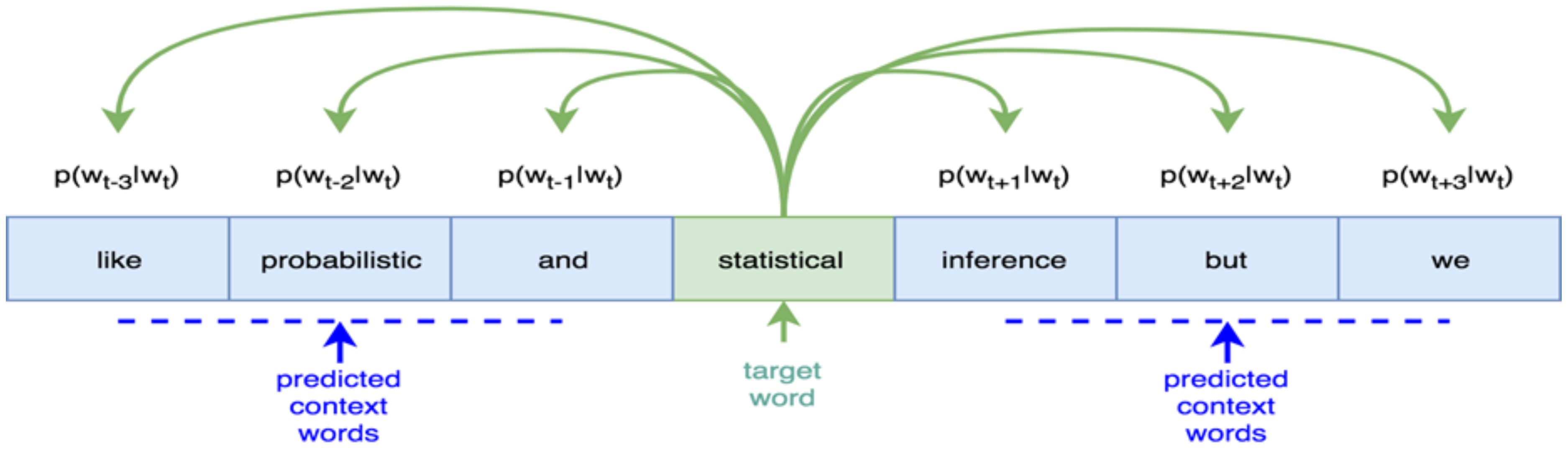
2) Skip-gram (cont.)

- 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.)

- 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.)

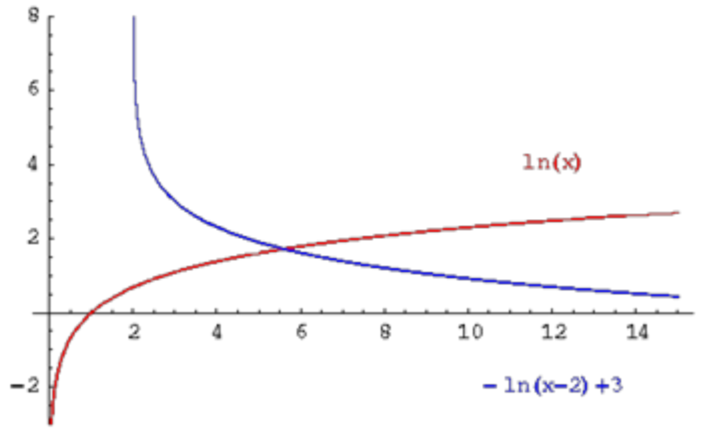
- Likelihood function: Given the target word (aka center word), maximize the probability of each context word

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

j = -m $\rightarrow$ previous words
j = +m $\rightarrow$ next words
j = 0 $\rightarrow$ the input word (w_t)

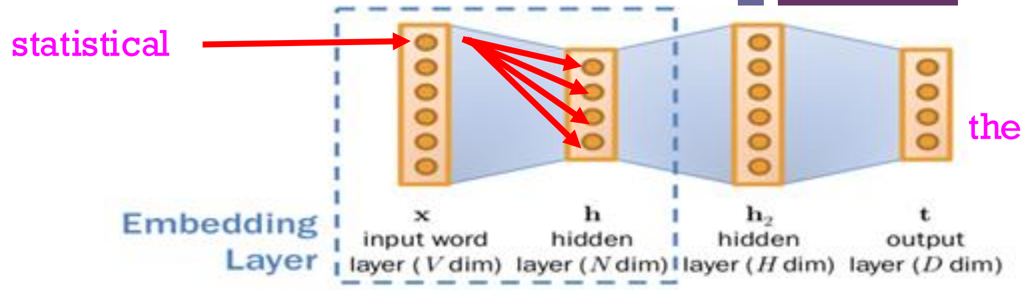
- Cost/Loss Function (Negative Log-Likelihood): (minimize)

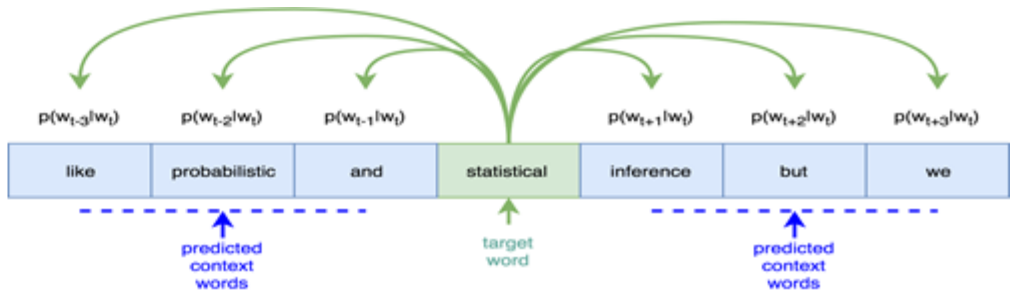
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m; j \ne 0} \log p(w_{t+j} | w_t; \theta)$$



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

2) Skip-gram (cont.)
Softmax





- 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'

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

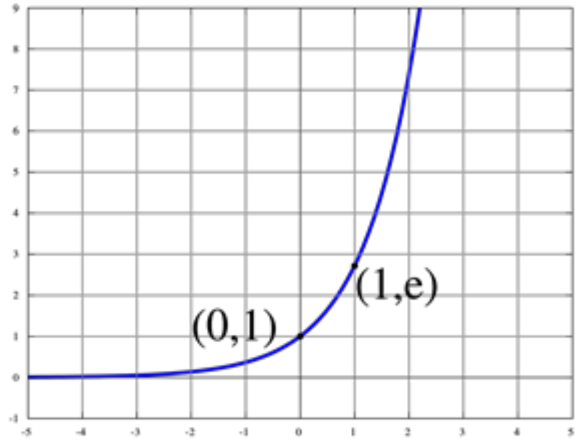
Dot product compares similarity of o and c.
Larger dot product = larger probability

After taking exponent, normalize over entire vocabulary

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

2) Skip-gram (cont.)
Softmax

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



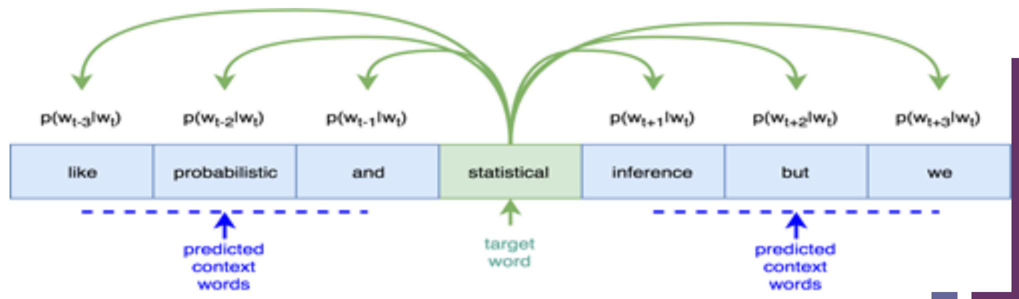
- This is basically a softmax function

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

- 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

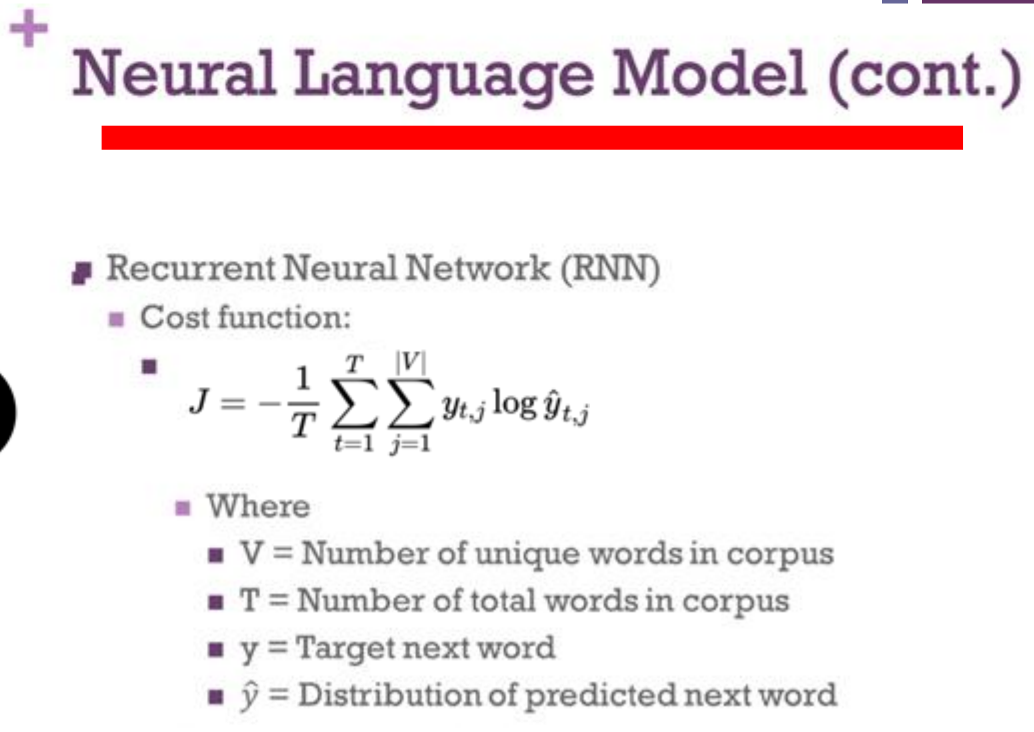
2) Skip-gram (cont.)



- Negative Log-Likelihood of Skip-gram model:

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

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

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

Predict context words

Predict the next word

+ Dealing with large vocab sizes

- **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 : Solution1: 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

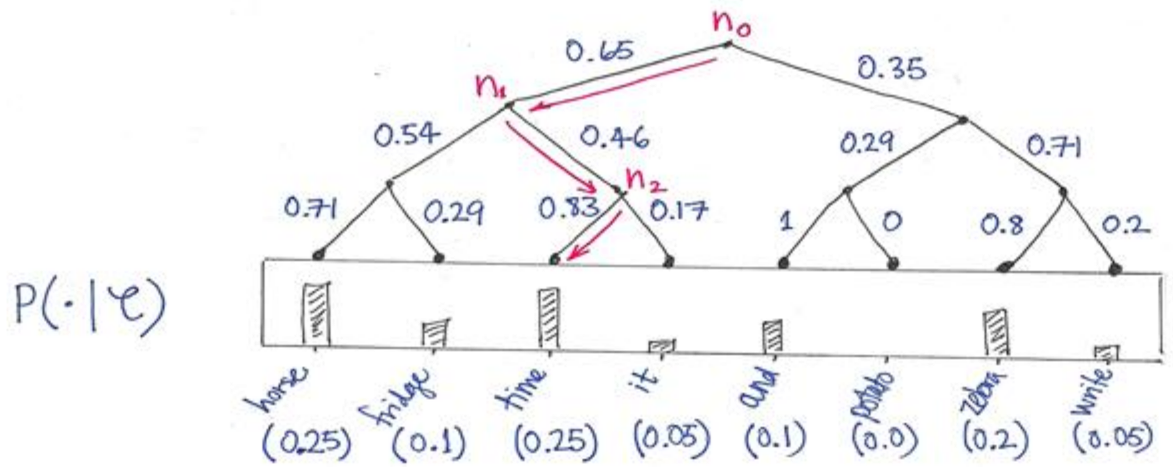
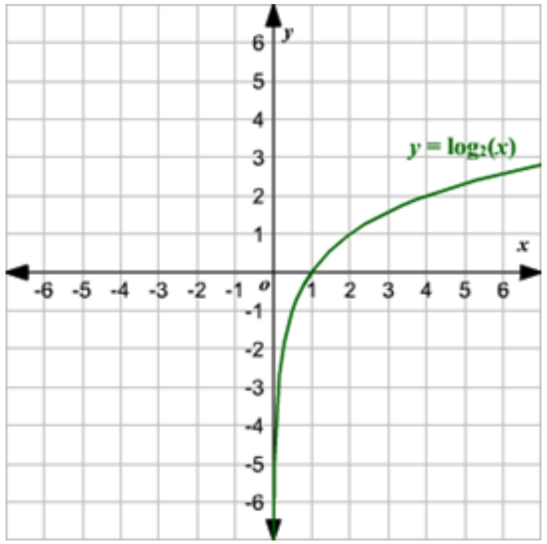


Image reference: http://building-babylon.net/2017/08/01/hierarchical-softmax/

Word2Vec training methods : Solution2: Negative Sampling (2)

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

$$J_t(\theta) = \underbrace{\boxed{\log \sigma\left(u_o^T v_c\right)}}_{\text{Context word (positive, +1)}} + \underbrace{\boxed{\sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]}}_{\text{Negative samples (negative, -1)}}$$



σ = sigmoid

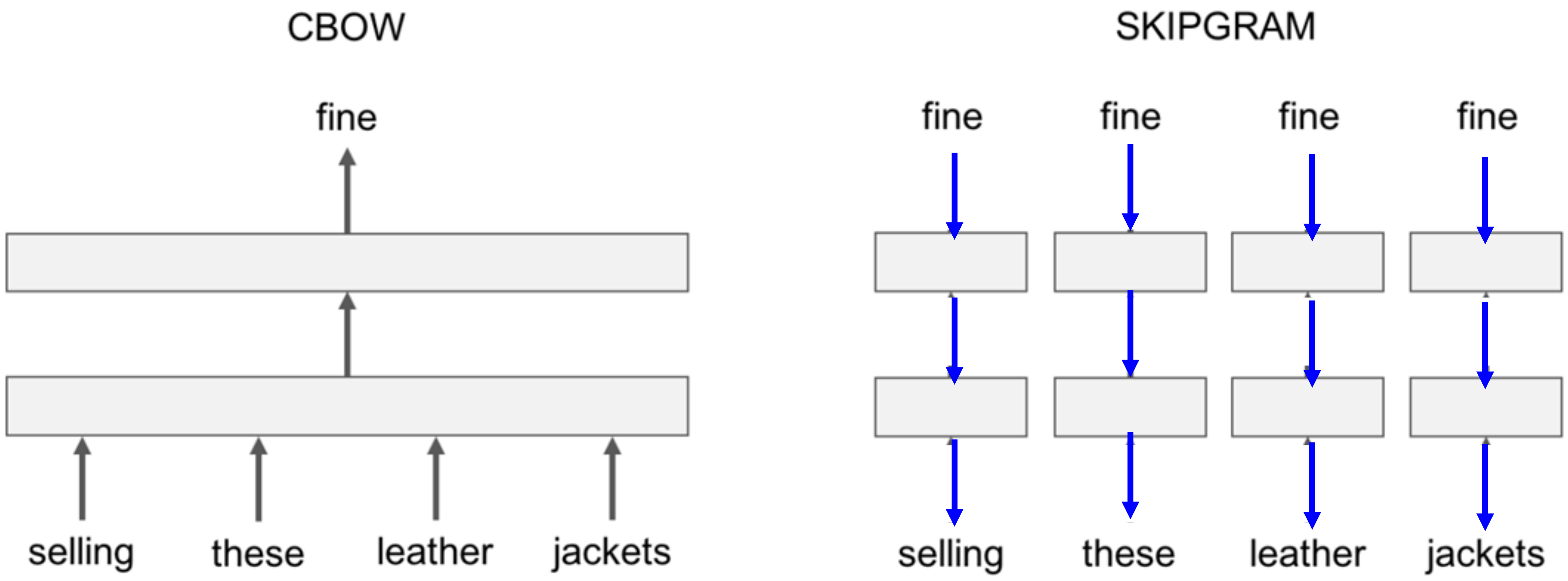
Word2Vec training methods : Solution2: Negative Sampling (2)

- Why ¾?
  - Chosen based on empirical experiments

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

- Intuition:
  - at: $0.9^{3/4} = 0.92$
  - farmer: $0.09^{3/4} = 0.16$
  - superfluous: $0.01^{3/4} = 0.032$
  - A rare word such as ‘superfluous’ is now 3x more likely to be sampled
  - While the probability of a frequent word “at” only went up marginally

Summary: CBOW vs Skip-gram (recap)

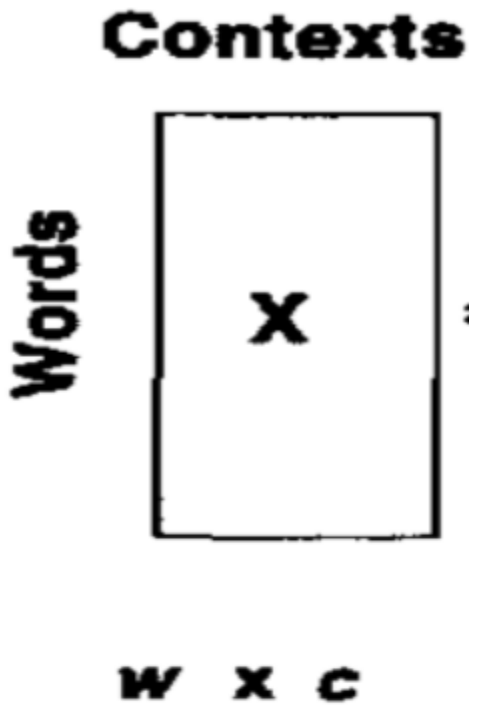


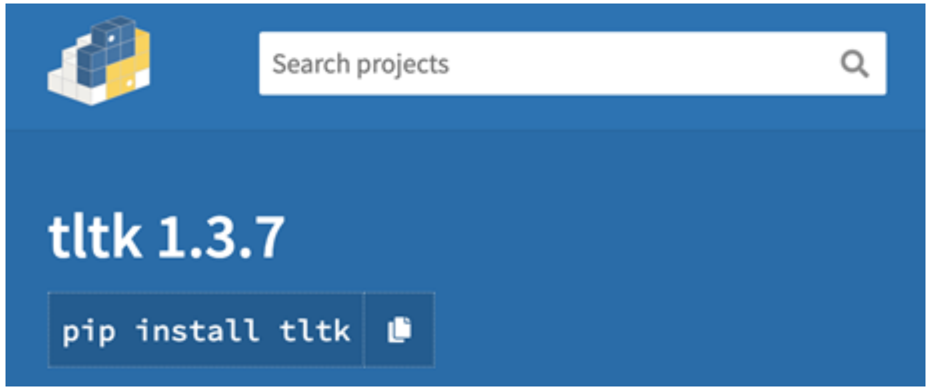
I am selling these fine leather jackets

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 family
- 2.3) A lot more…





tltk 1.3.7

```
pip install tltk
```

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.

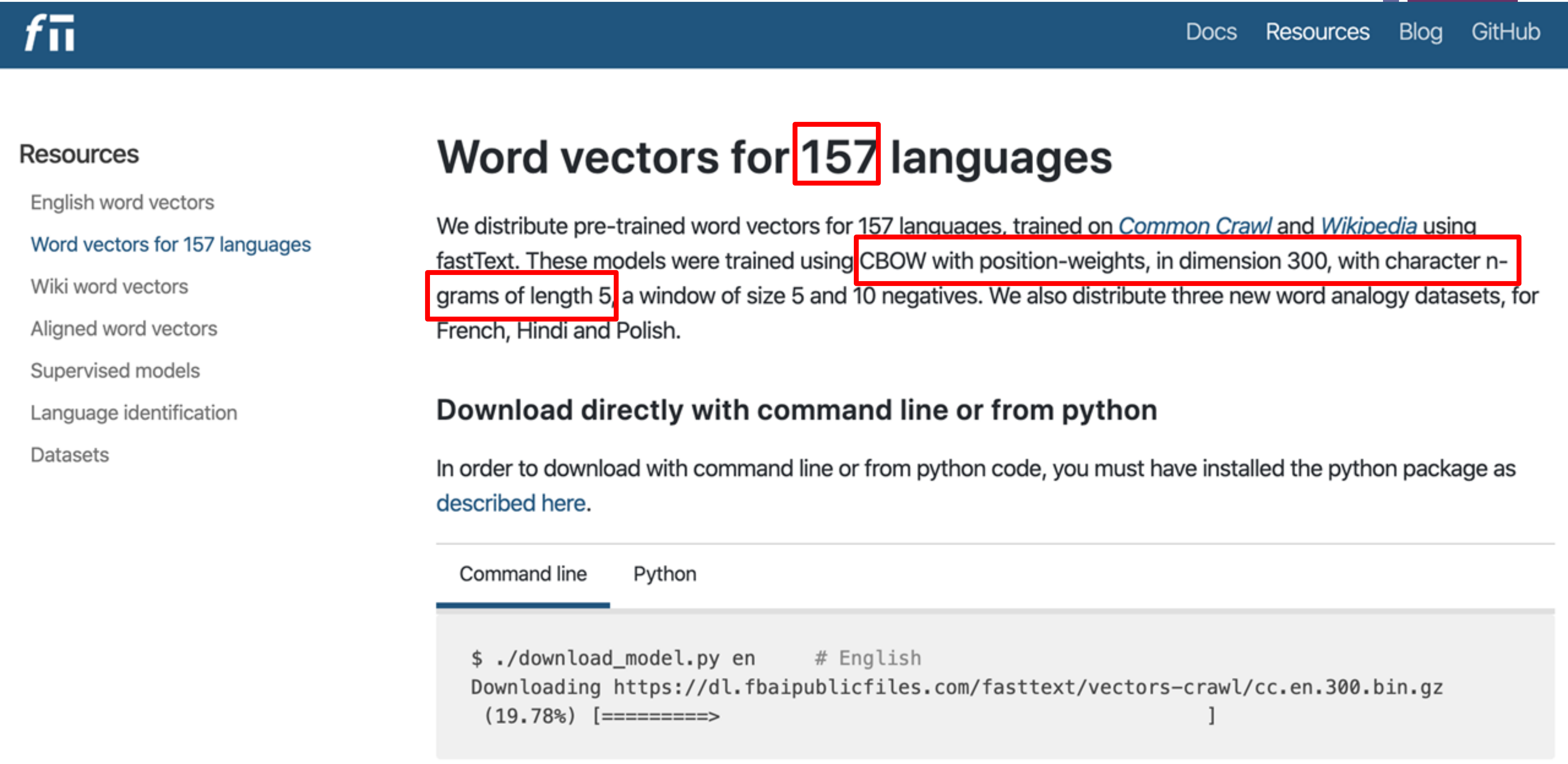
`<where>, <wh, whe, her, ere, re>`

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/

fastText library by Facebook



Resources

- English word vectors
- Word vectors for 157 languages
- Wiki word vectors
- Aligned word vectors
- Supervised models
- Language identification
- Datasets

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

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

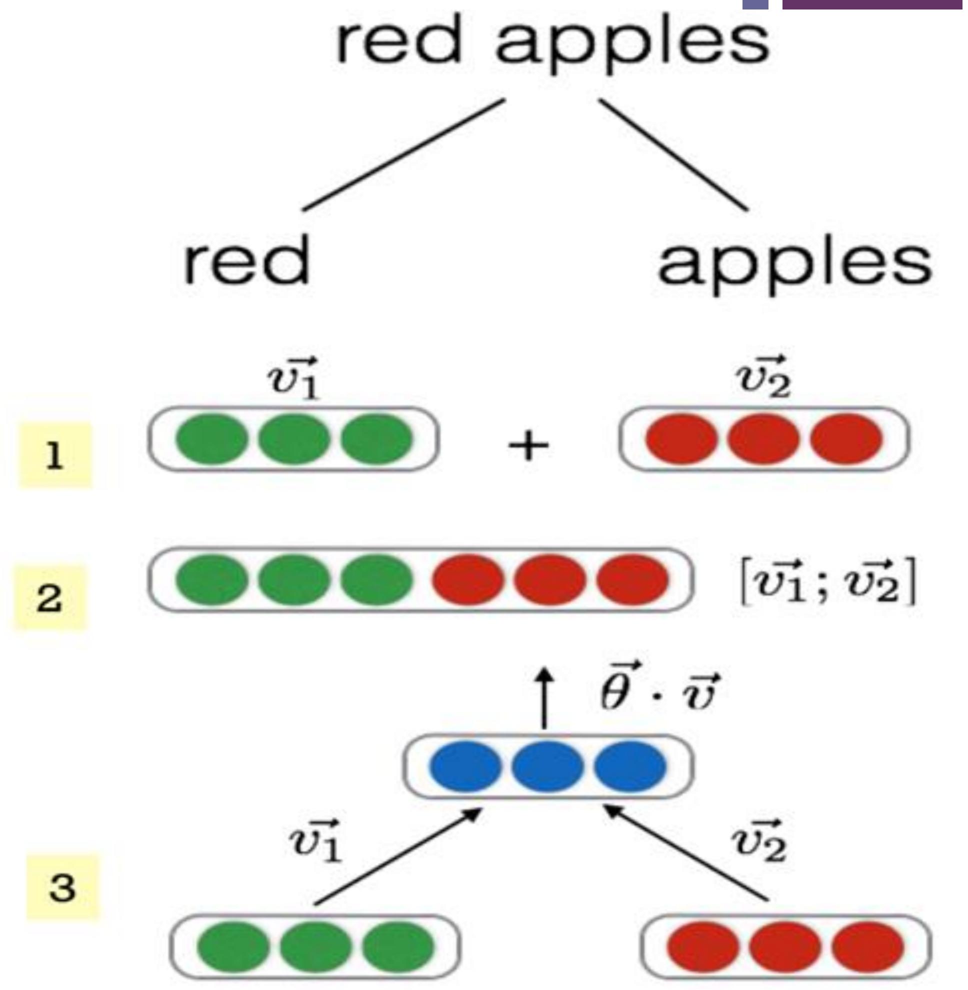


Image ref: Prof. Regina Barzilay , NLP@MIT

Outline

- **1) How to represent words?**
  - Symbolic vs distributional word representations
- **2) Distributional: Sparse vector** representations (discrete representation)
  - Term-document matrix
  - Co-occurence
  - PPMI
  - TF-IDF
- **3) Distributional: Dense vector representations**
  - Word2Vec
    - CBOW
    - Skip-gram
  - Word2Vec training methods
  - Pre-trained vector representations
    - Adaptation
  - Compositionality