CHULA ENGINEERING
Foundation toward Innovation

COMPUTER

Word Representations

2110594: Natural Language Processing (NLP)

Peerapon Vateekul & Ekapol Chuangsuwanich
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

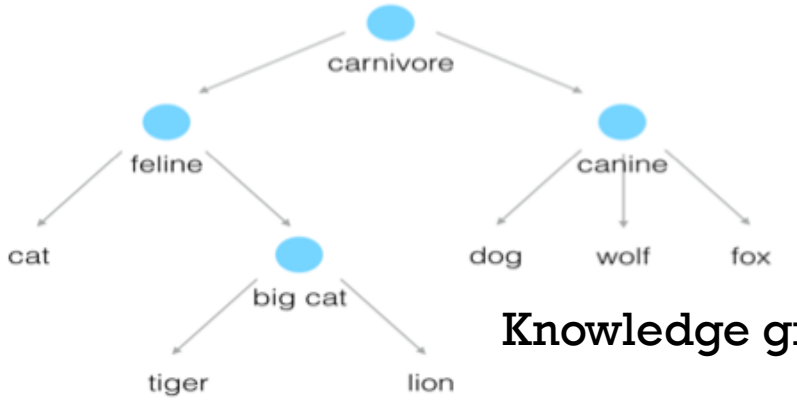
Credit: Can Udomcharoenchaikit & Nattachai Tretasayuth

Outline

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

How to represent words? Symbolic vs Distributional representations (1)

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

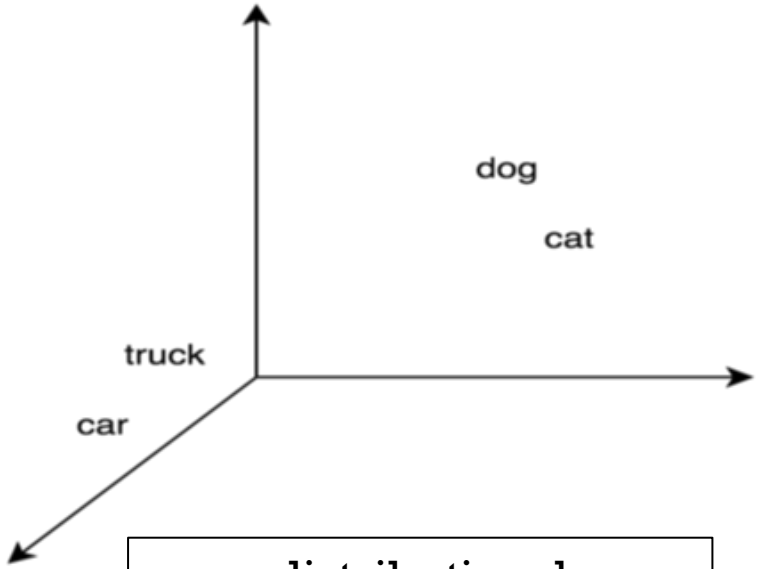


Knowledge graph

เสือ = [0 1 0 0 0 ... 0 0]

One-hot model

symbolic

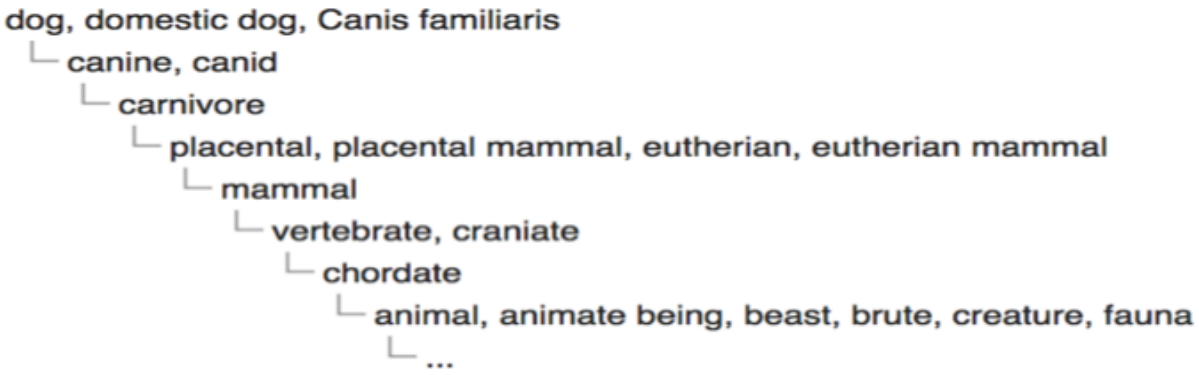


distributional

How to represent words? Symbolic vs Distributional representations (2)

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

How to represent words? Symbolic vs Distributional representations (3)

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]

  - Drawback:
    - Cannot capture similarity between words

How to represent words? Symbolic vs Distributional representations (4)

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

These words represent "เสือ"

เสือ เป็นสัตว์เลี้ยงลูกด้วยนมในวงศ์ฟิลิดีซึ่งเป็นวงศ์เดียวกับแมว

How to represent words?: Symbolic vs Distributional representations (5)

- 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.

Sparse vector representations: term-document matrix (1)

- 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 (2)

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

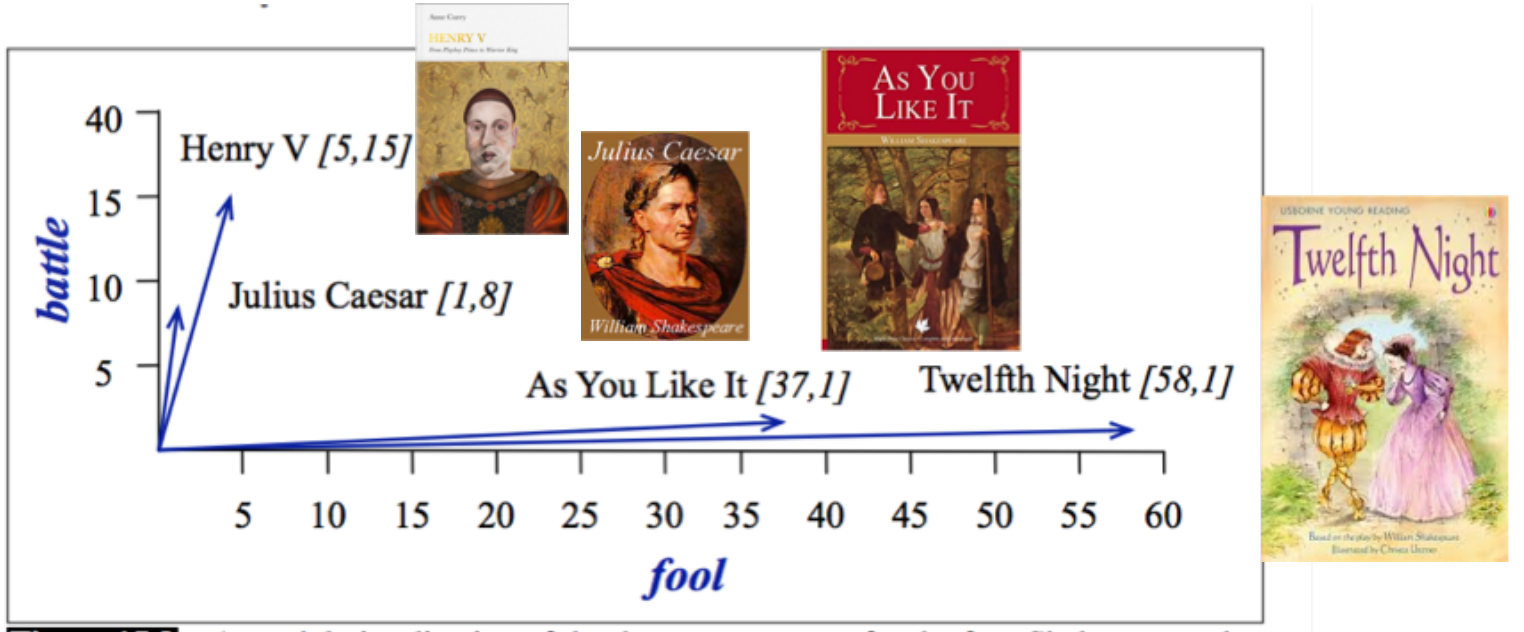


Figure 15.3 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

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 (3)

- 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 (4)

- 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

2.1.2

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 →

	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 similar words tend to have similar vectors (word similarity)

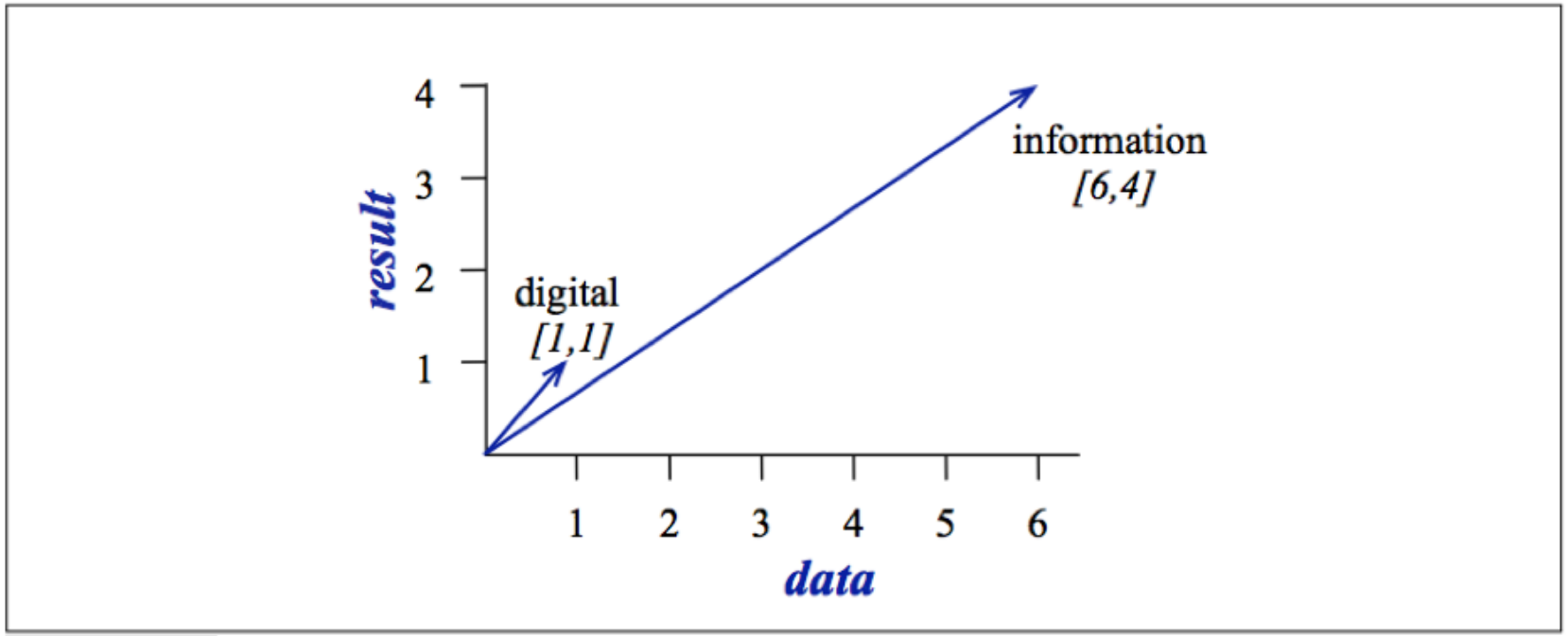


Figure 15.5 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *result*.

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: Positive Pointwise Mutual Information (PPMI) (1)

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

Sparse vector representations: Positive Pointwise Mutual Information (PPMI) (2)

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

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

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

Negative PMI values tend to be unreliable.

It is common to replace all negative PMI values with zero

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

Sparse vector representations: Postive Pointwise Mutual Information (PPMI) (4)

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

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

Sparse vector representations: TF-IDF (1)

- 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)

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

- TF-IDF

$$TFIDF(w) = \frac{TF(w)}{IDF(w)}$$

Sparse vector representations: TF-IDF (2)

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

Dense vector representations (1)

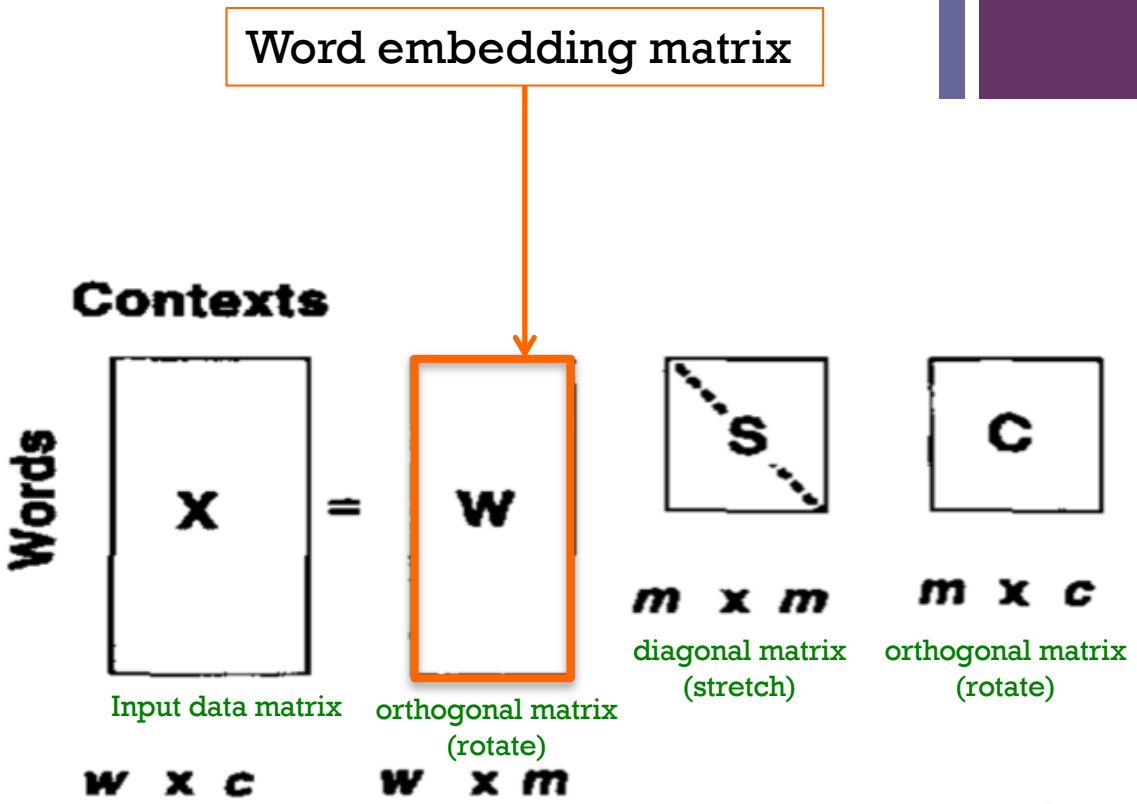
- Sparse vector representations such as PPMI vectors are:
  - Long (length of vector ≈ 20,000 to 50,000)
  - Sparse (most elements are zero)
- Dense vector representations are introduced to:
  - Reduce length of vectors (length of vector ≈ 200 to 1,000)
  - Reduce sparsity (hence the name "dense"; most elements are not zero)

Dense vector representations (2)

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

Dense vector representations: SVD-based method (1)

- Single Value Decomposition (SVD)
  - dimensionality reduction
  - 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
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 (2)

- 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

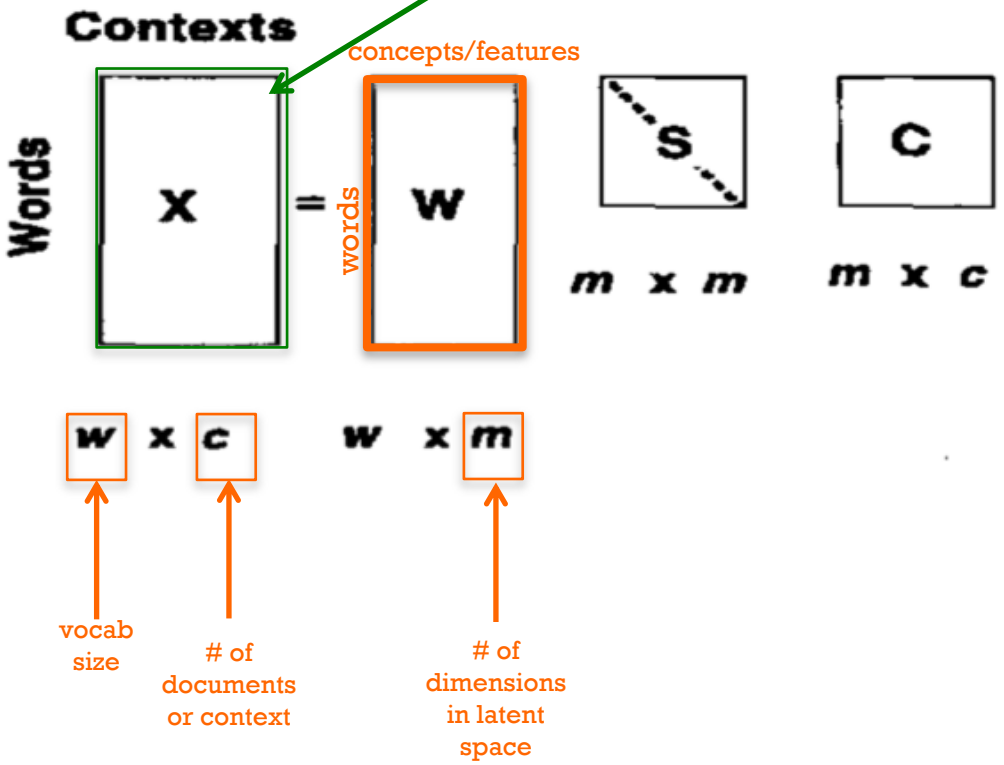
vocabulary

	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

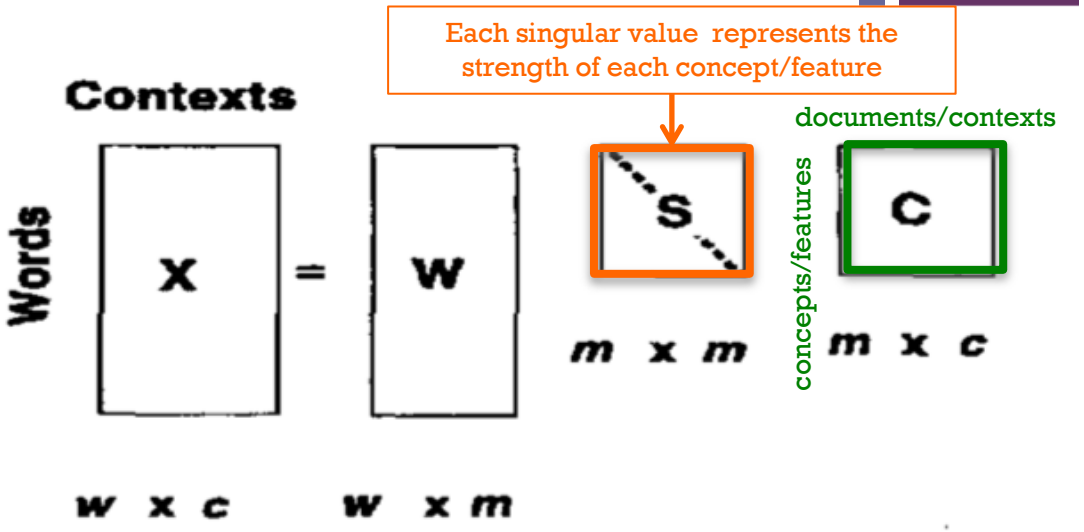


w = total words
c = total documents
m = total latent factors (dimensions)

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 (3)

- 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

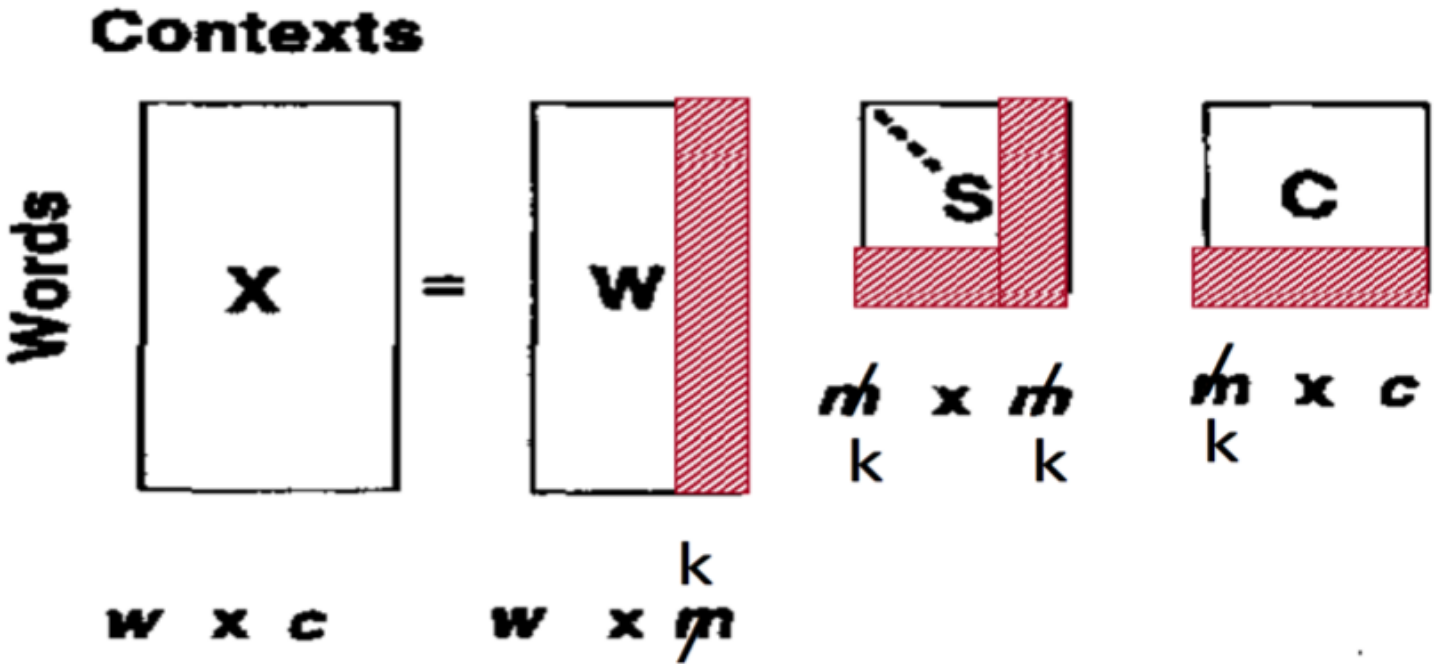


w = total words
c = total documents
m = total latent factors (dimensions)

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 (4)

- 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 (5)

- 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

	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

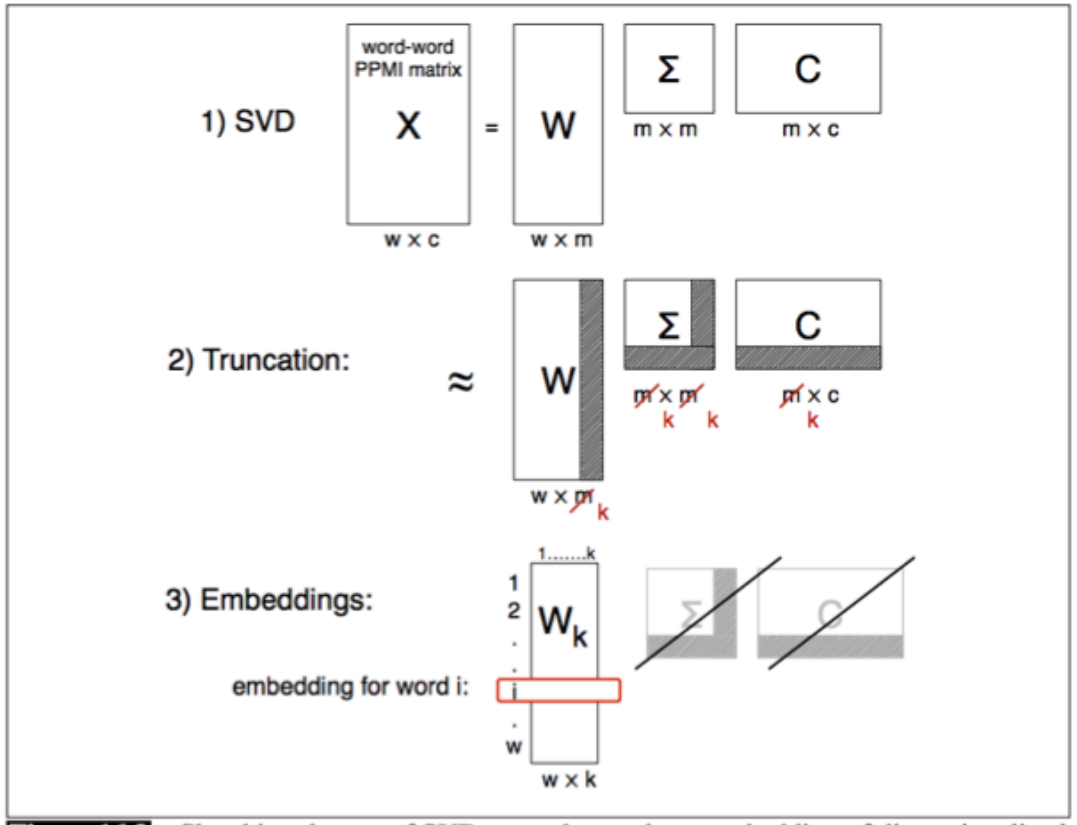


Figure 16.3 Sketching the use of SVD to produce a dense embedding of dimensionality k from a sparse PPMI matrix of dimensionality c. The SVD is used to factorize the word-word PPMI matrix into a W, Σ, and C matrix. The Σ and C matrices are discarded, and the W matrix is truncated giving a matrix of k-dimensionality 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 (6)

- 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

Dense vector representations : neural language models (1)

- Tomas Mikolov introduced **Skip-gram** and **CBOW** in 2013
- 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 language models (2)

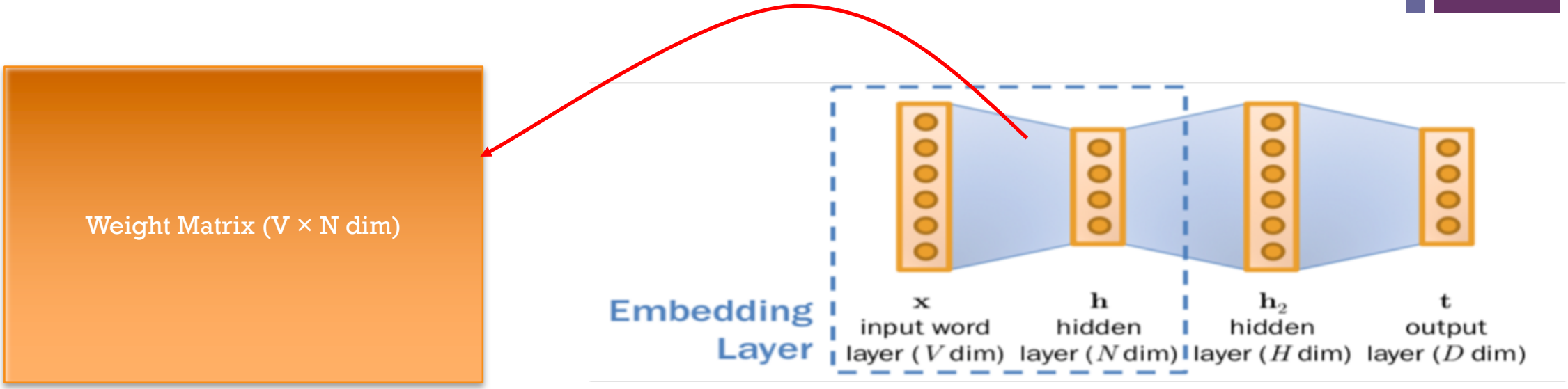


Image reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017

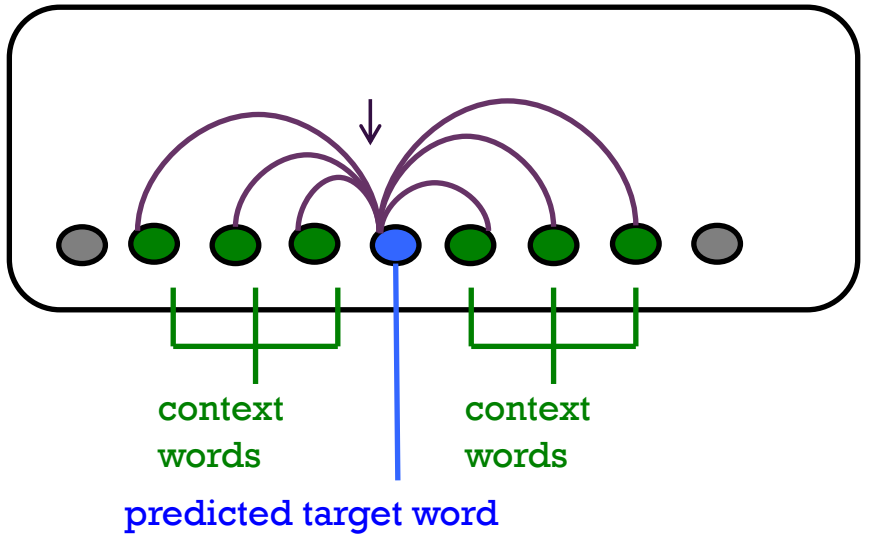
Dense vector representations : neural language models (3)

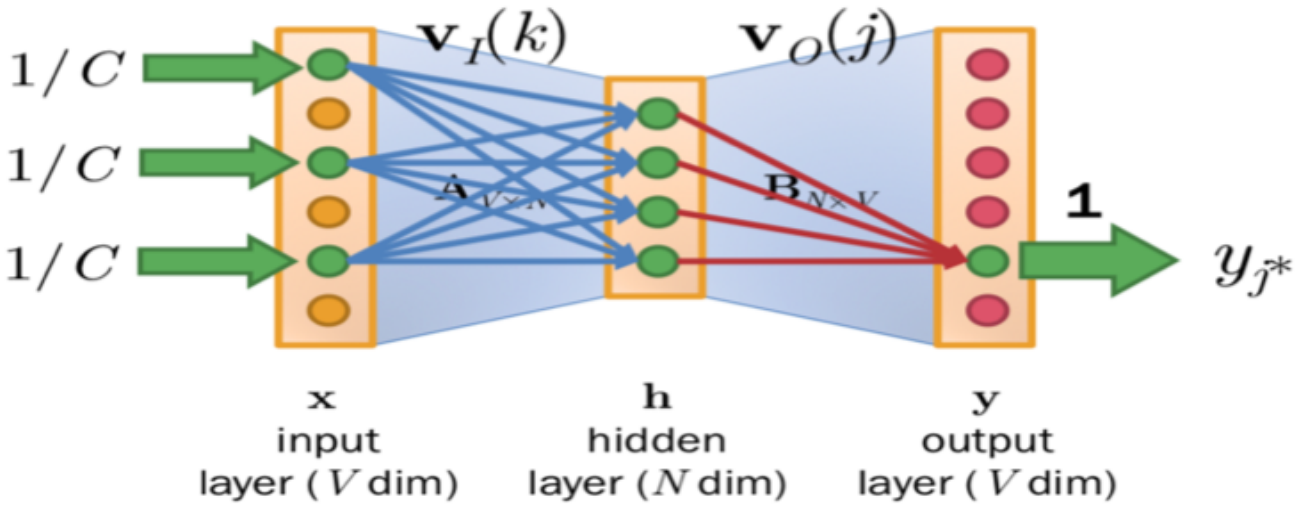
- Intuition: "Iteratively make the embeddings for a word
  - (positive class) more like the embeddings of its neighbors and
  - (negative class) less like the embeddings of other words."

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

Dense vector representations : CBOW (1)

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



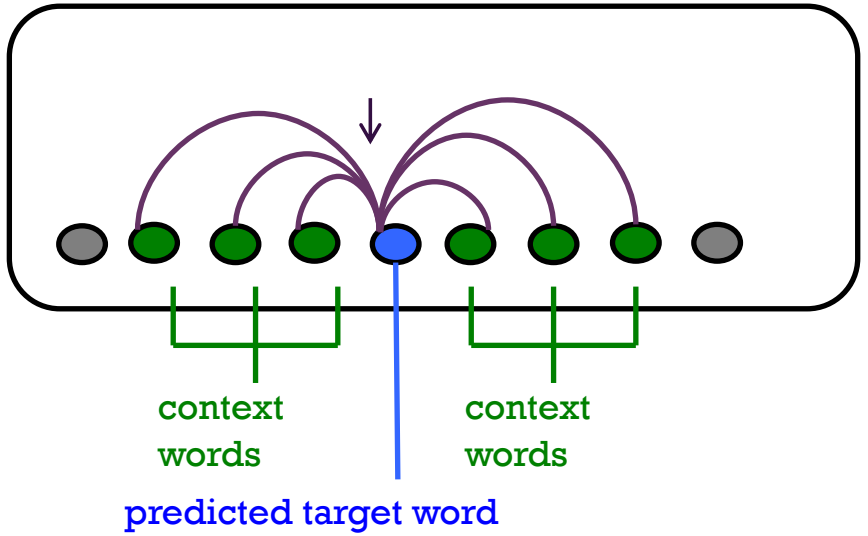


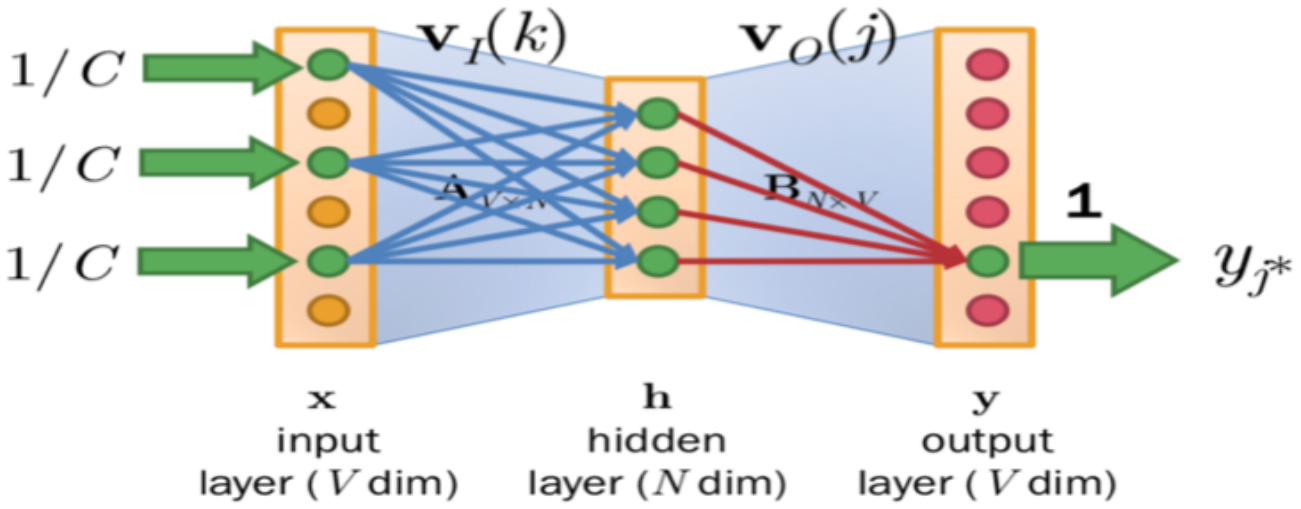
$$\mathbf{h}_{\text{ctx}} = \frac{1}{C}\sum_{q=1}^{C}\mathbf{v}_I(q)$$

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

Dense vector representations : CBOW (2)

- We simply average the encoding vectors



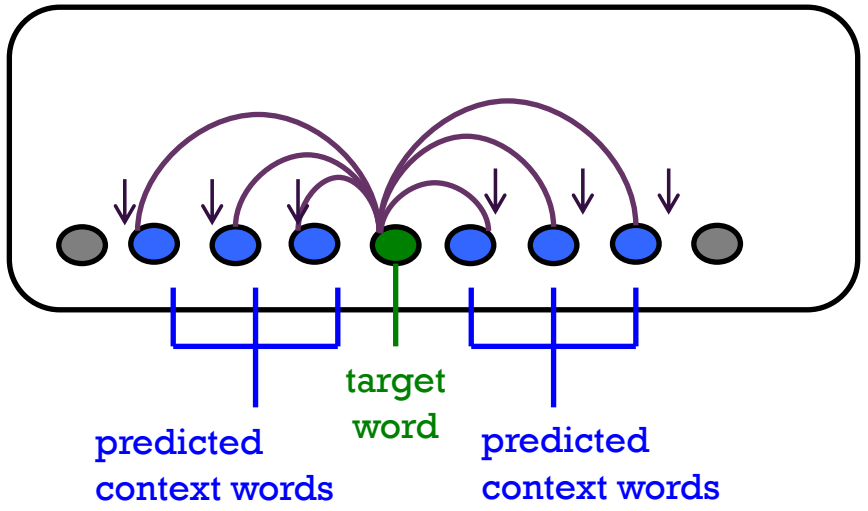


$$\mathbf{h}_{\text{ctx}} = \frac{1}{C}\sum_{q=1}^{C}\mathbf{v}_I(q)$$

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

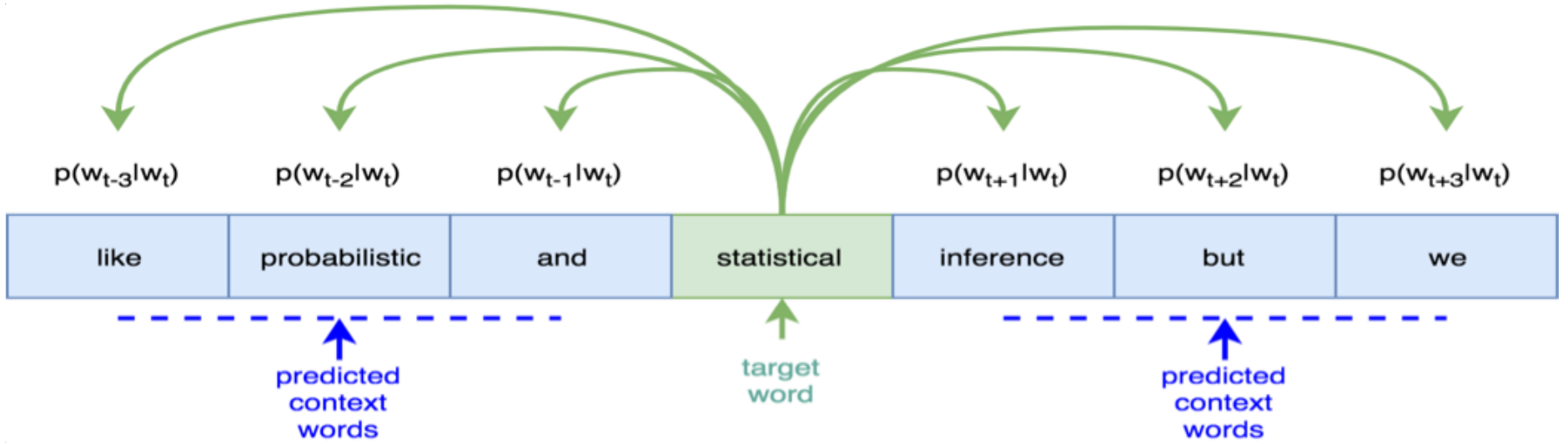
Dense vector representations : Skip-gram (1)

- 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



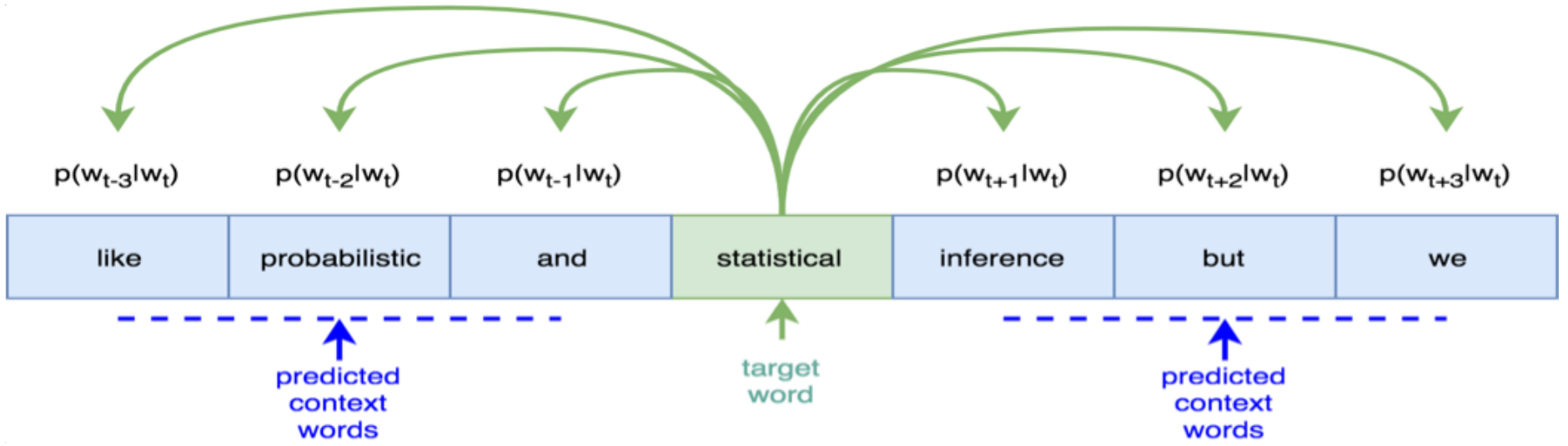
Dense vector representations : Skip-gram (2)

- 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."



Dense vector representations : Skip-gram (3)

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



Dense vector representations : Skip-gram (4)

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

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

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

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

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

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

Dense vector representations : Skip-gram (5)

- How to calculate $p(w_{t+j}|w_t;\theta)$?
  - for each word w, we will use two vectors
    - v_w when w is a target/center word
    - u_w 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

Dense vector representations : Skip-gram (6)

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

- 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

Dense vector representations : Skip-gram (4)

- Negative Log-Likelihood:

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

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

Language Model
Neural Language Model

- 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
    - Actually, we are calculating perplexity
  - Perplexity = 2^J

Dense vector representations : Skip-gram (5)

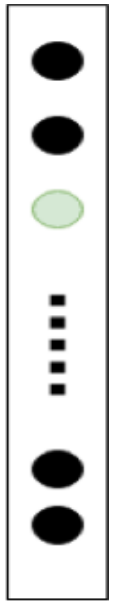
- Skip-gram model step-by-step:
  1. Generate a one hot input vector for of the target word (center word)
  2. Get the embedded vector for the target center word
  3. Generate 2*m score vectors (where m is the window size)
  4. Turn the score vectors into probabilities
  5. We desire our probability 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

Dense vector representations : Skip-gram (6)

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

statistical = [0 0 1 … 0 0]



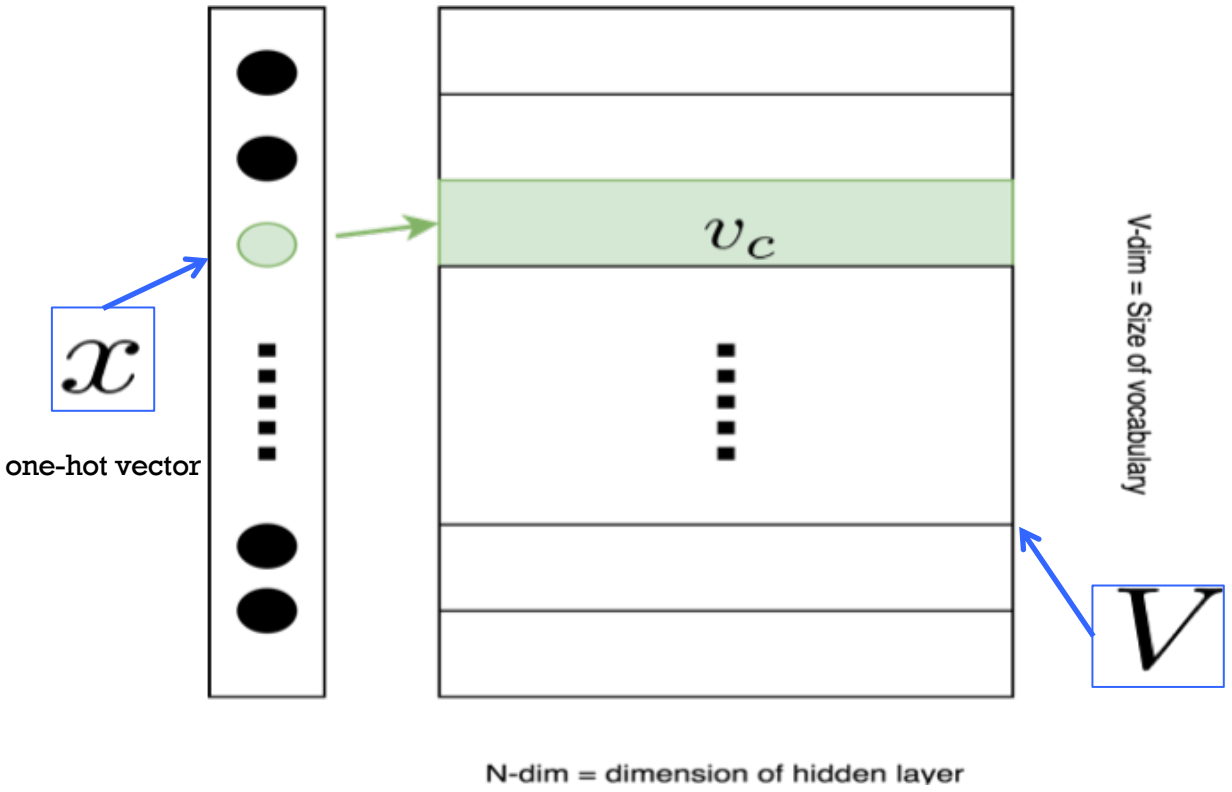
$$x \in \mathbb{R}^{|V|}$$

V-Dim = size of the vocabulary

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

Dense vector representations : Skip-gram (7)

- 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

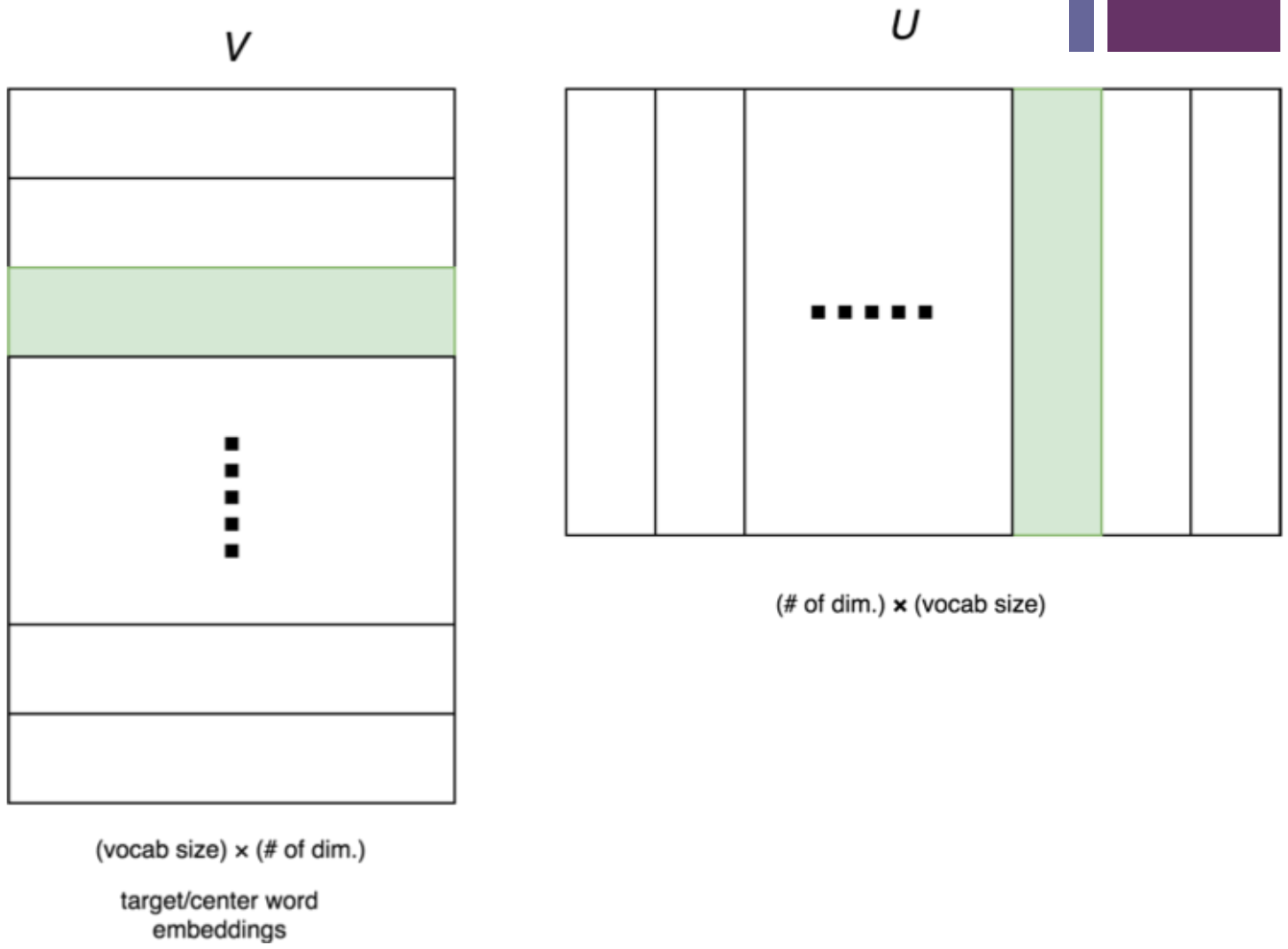
Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf

Dense vector representations : Skip-gram (8)

- 3. Generate a score vector

$$z = Uv_c$$

- Note that similarity(v_c,u_o) = $v_c \cdot u_o$



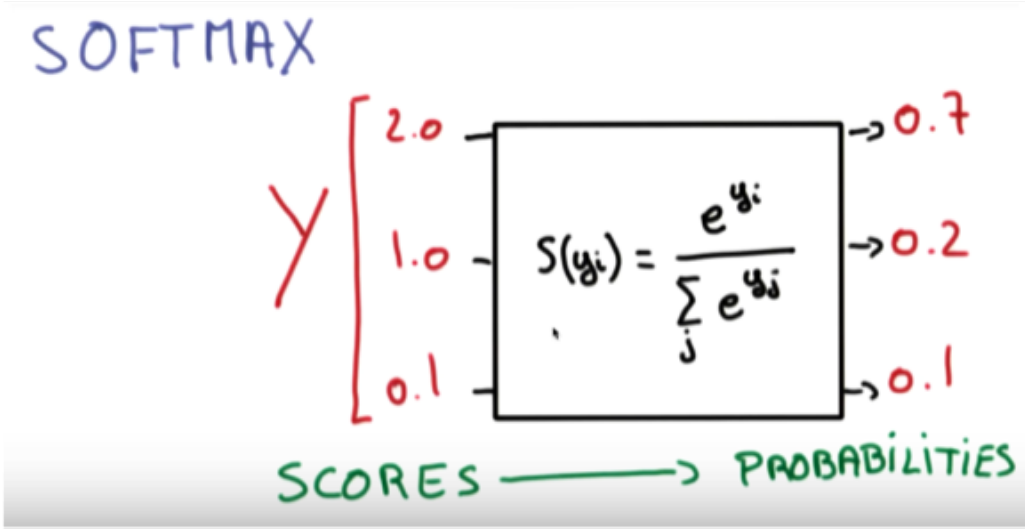
Reference:http://web.stanford.edu/class/cs224n/lecture_notes/cs224n-2017-notes1.pdf

Dense vector representations : Skip-gram (9)

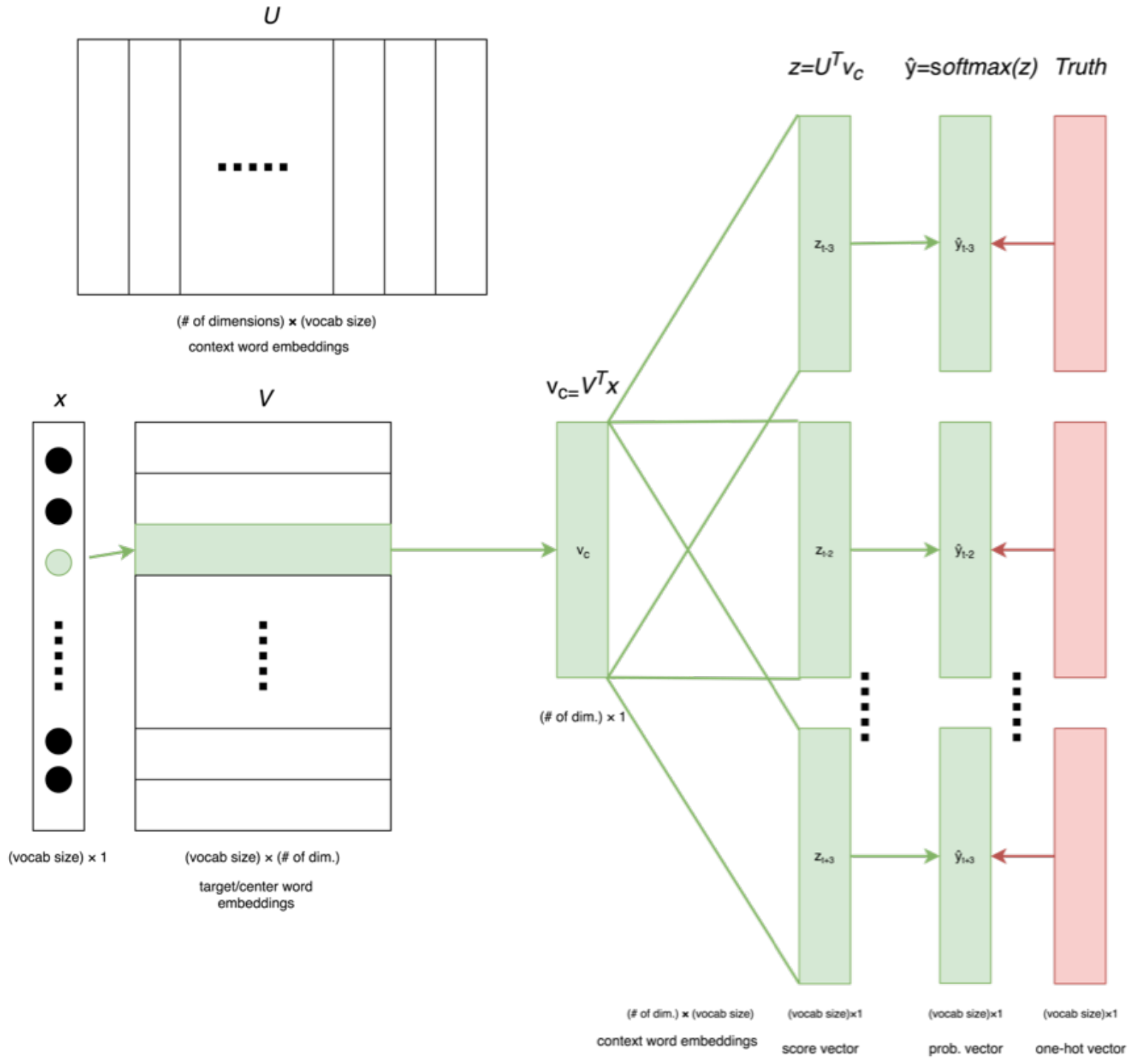
- 4. Turn score into probabilities

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

- 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

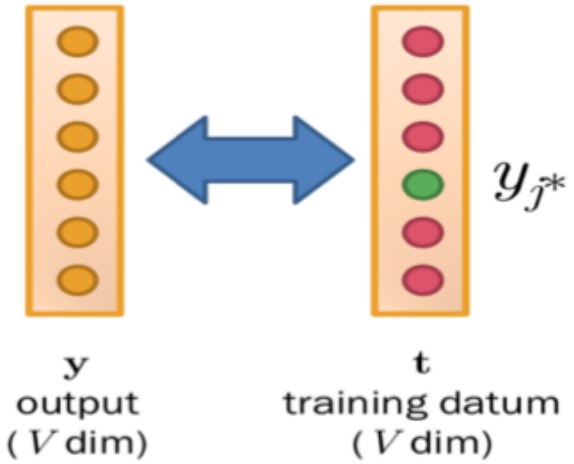

U
(# of dimensions) × (vocab size)
context word embeddings
x
V
(vocab size) × 1
(vocab size) × (# of dim.)
target/center word embeddings
$v_c = V^T x$
v_c
(# of dim.) × 1
$z = U^T v_c$
ŷ=softmax(z)
Truth
z_{t-3}
z_{t-2}
z_{t+3}
$\hat{y}_{t-3}$
$\hat{y}_{t-2}$
$\hat{y}_{t+3}$
(# of dim.) × (vocab size)
context word embeddings
(vocab size)×1
score vector
(vocab size)×1
prob. vector
(vocab size)×1
one-hot vector

Word2Vec training methods

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

Word2Vec training methods : 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



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

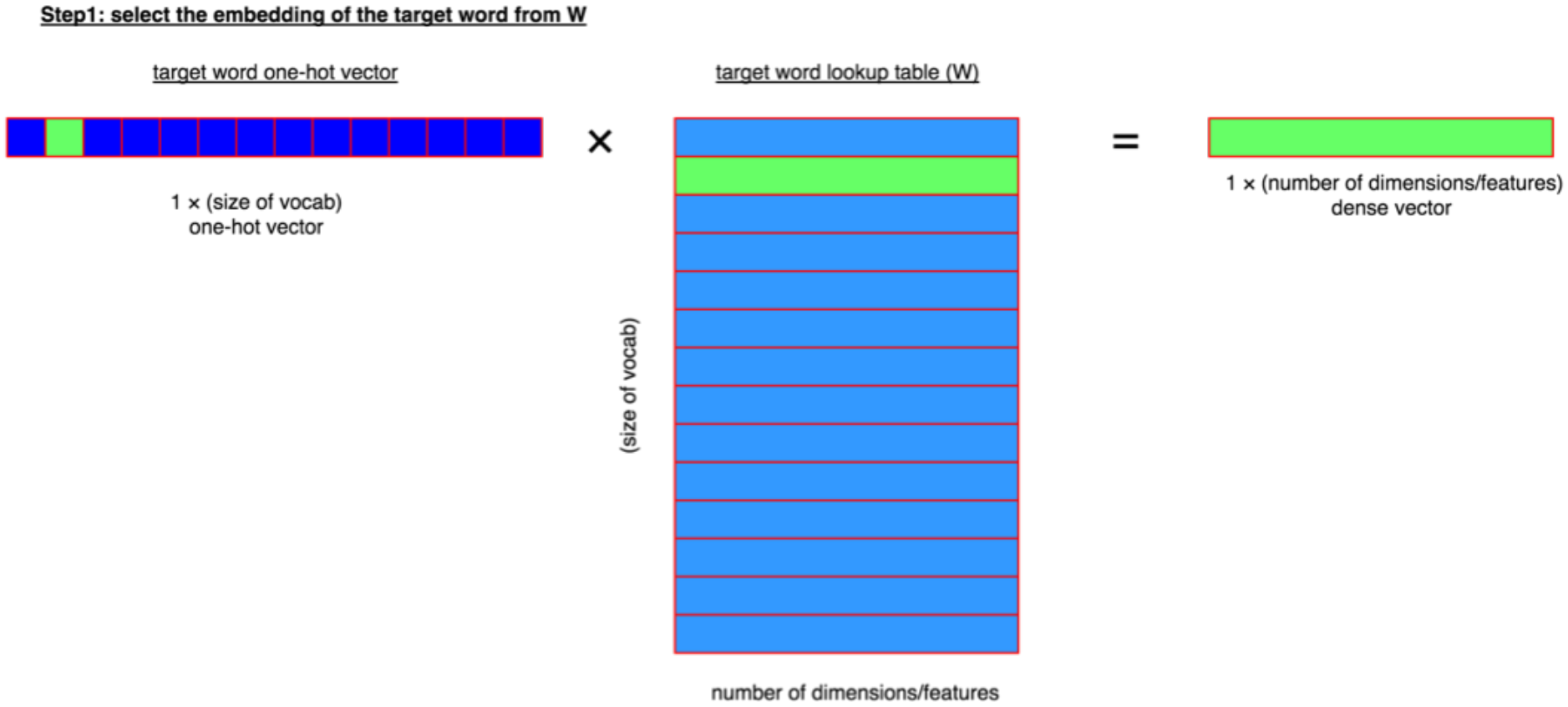
Reference: Boonkwan, Prachya. "Word2Vec: When Language Meets Number Crunching", https://goo.gl/hhA3hO, Feb 2017

Word2Vec training methods : Negative Sampling (2)

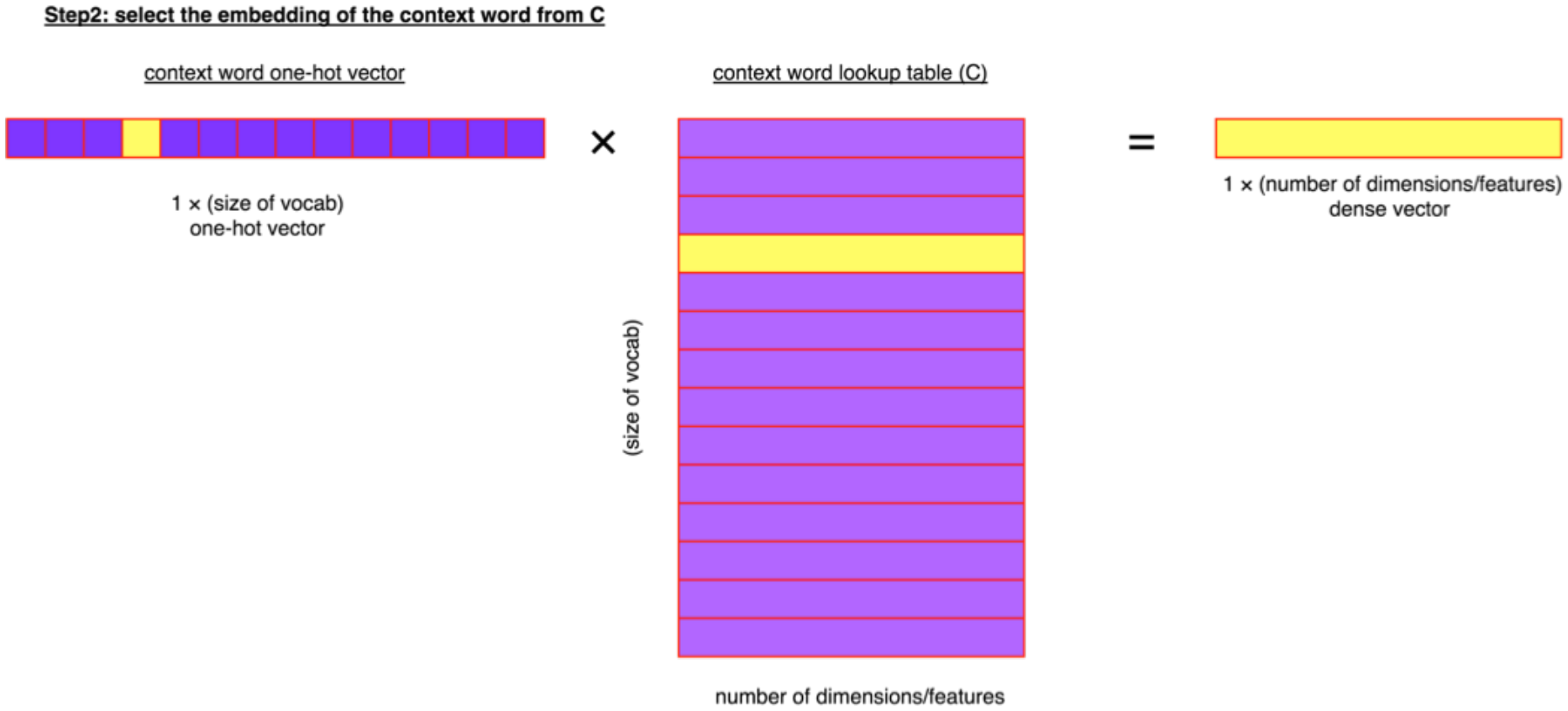
- Why ¾?
  - Chosen based on empirical experiments
- Intuition:
  - ที่: $0.9^{3/4} = 0.92$
  - ชาวนา: $0.09^{3/4} = 0.16$
  - ตถาตา: $0.01^{3/4} = 0.032$
  - A rare word such as 'ตถาตา' is now 3x more likely to be sampled
  - While the probability of a frequent word 'ที่' only went up marginally

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

Dense vector representations : Skip-gram (Negative Sampling)

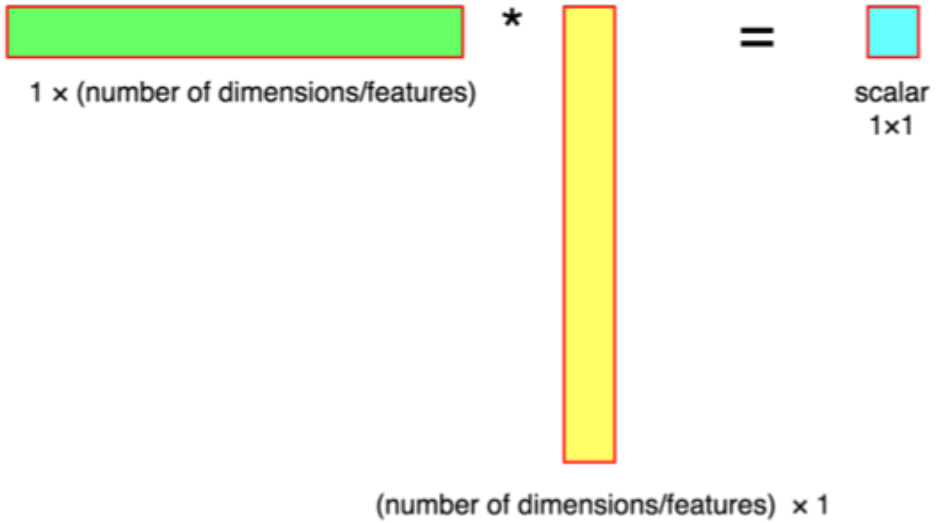


Dense vector representations : Skip-gram (Negative Sampling) (cont.)

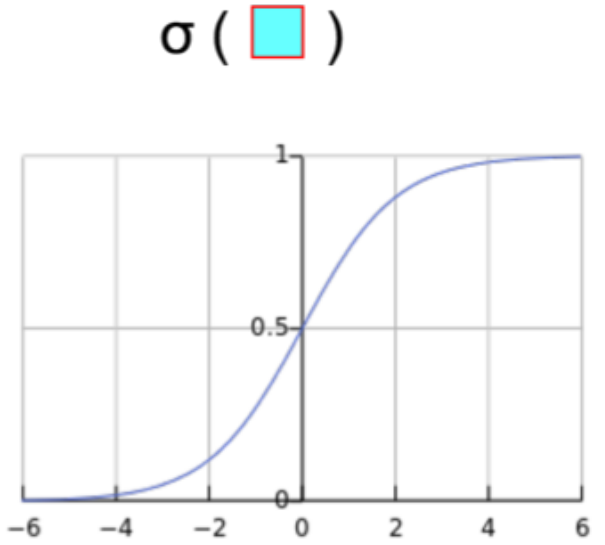


Dense vector representations : Skip-gram (Negative Sampling) (cont.)

Step3: compute the dot product:w*c



Step4: normailize dot products into probability



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

Dense vector representations : Skip-gram (Negative Sampling) (cont.)

- 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

Negative samples

Pre-trained Word2Vec

- GloVe
  - https://nlp.stanford.edu/projects/glove/
- fastText [Available in Thai language]
  - https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

Pre-trained Word2Vec: GloVe

- 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)

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: fastText

- 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 294 languages (including Thai) trained on Wikipedia.

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).

Adaptation (1)

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

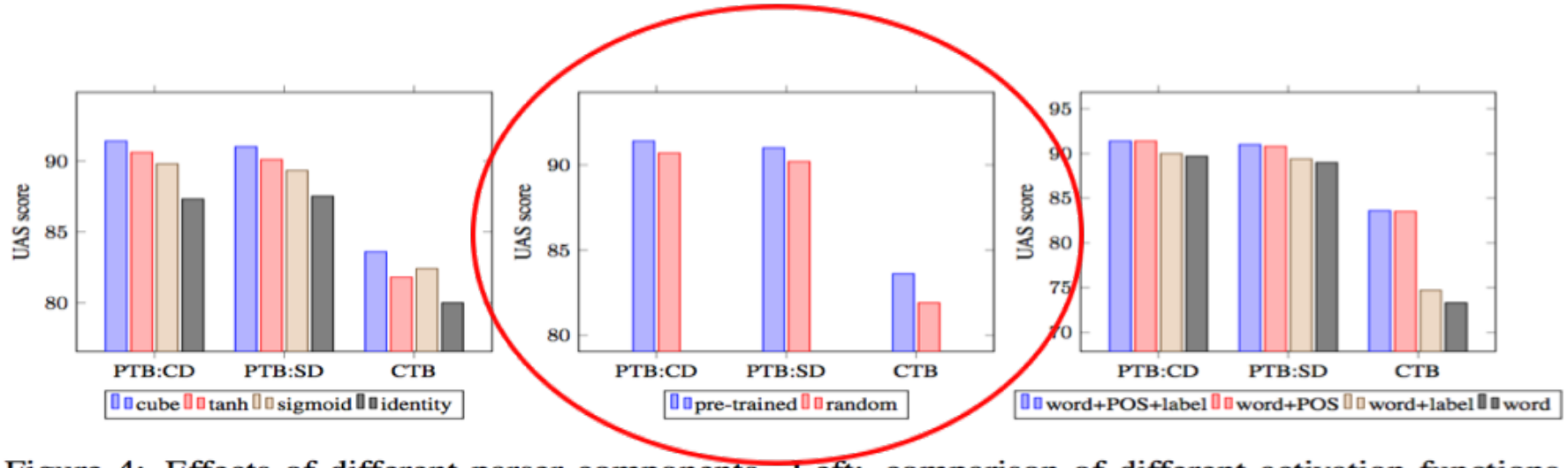
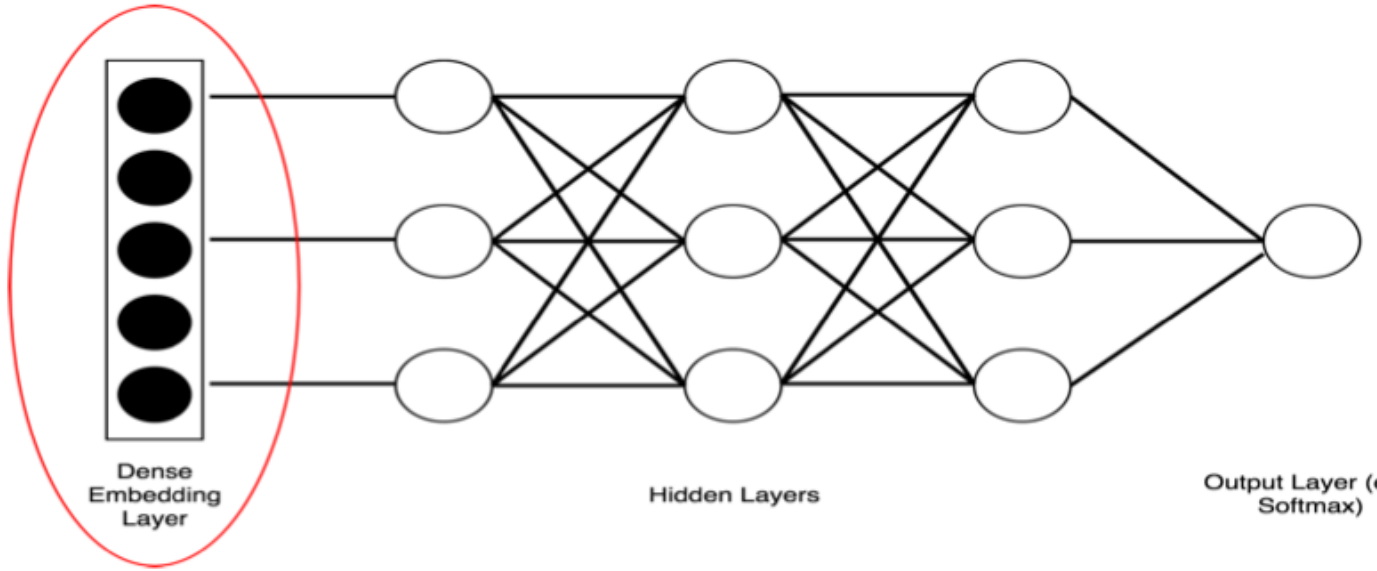


Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

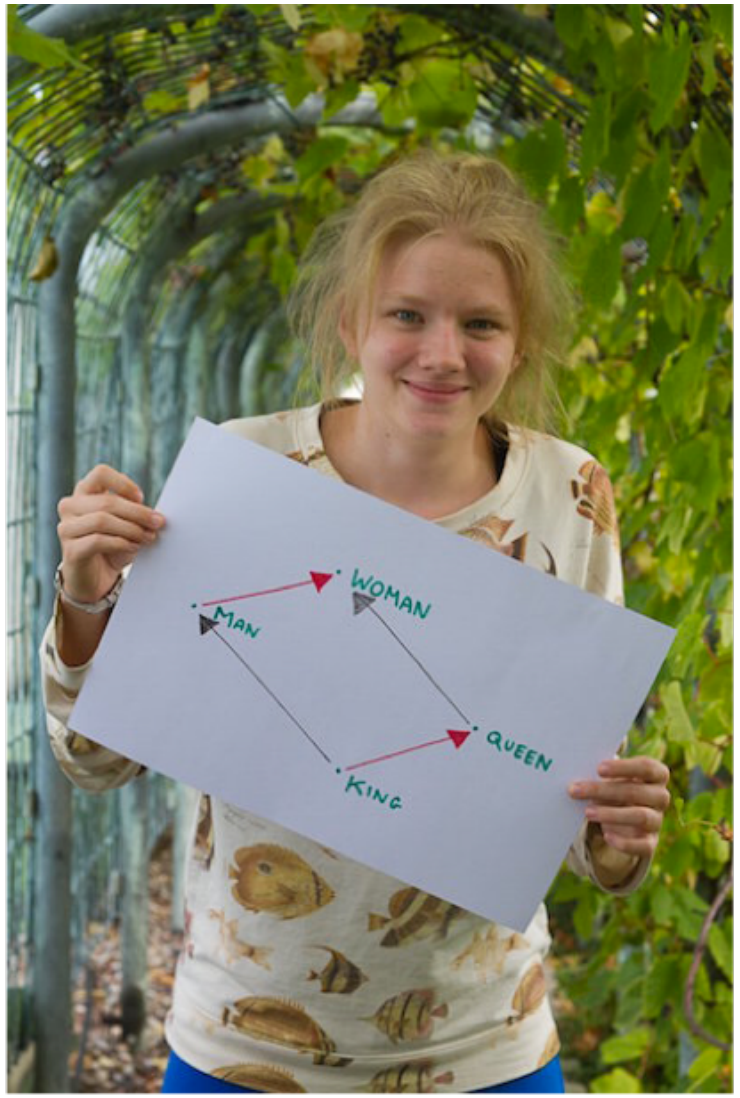
Image reference: Chen, Danqi, and Christopher Manning. "A fast and accurate dependency parser using neural networks." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014. APA

Adaptation (2)

- Classification Model with pre-trained word embedding
- You already know how to create a classification model
  - In homework 1, word segmentation is a classification task
- 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

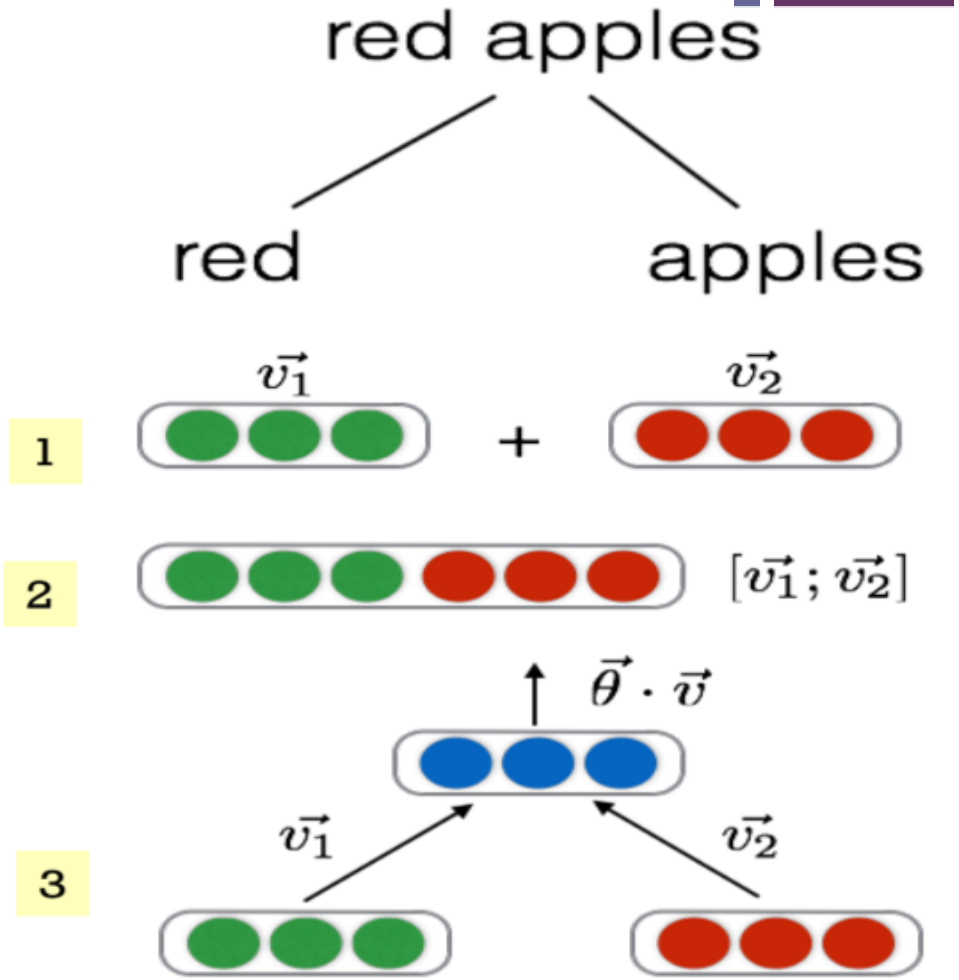


Image ref: Prof. Regina Barzilay , NLP@MIT

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
  - **Sectional Preference**: 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
- Adaptation
- Learned Semantic-Syntactic Relationships
- Compositionality
- Word2Vec evaluation: extrinsic evaluation and intrinsic evaluation