Word2Vec

Hung-Hsuan Chen

1

Attribute of a text document

- One-hot encoding the words

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

Word Vector (Passage Vector)

Document Vector

2

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^v$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

3

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.

4

Term frequency tf

- The term frequency $\text{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1+\log_{10} \text{tf}_{t,d}, & \text{if } \text{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:
- score $= \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
  - We want a high weight for rare terms like *arachnocentric*.
- We will use document frequency (df) to capture this

idf weight

- df_t is the document frequency of *t*: the number of documents that contain *t*
  - df_t is an inverse measure of the informativeness of *t*
  - $\text{df}_t \leq N$
- We define the idf (inverse document frequency) of *t* by $\text{idf}_t = \log_{10} (N/\text{df}_t)$
  - We use log (N/df_t) instead of N/df_t to “dampen” the effect of idf.

idf example, suppose *N* = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

$$\mathrm{idf}_t = \log_{10}\,(N/\mathrm{df}_t)$$

There is one idf value for each term *t* in a collection.

9

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$\mathrm{w}_{t,d} = \log(1+\mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection
- Many variants, e.g., BM25

10

These are all bag-of-words model

- The feature sets are the same
  - Each word is a feature
- The feature values are generated in different ways
  - Based on term counts
  - Based on log of term frequency
  - Based on tf-idf
- Compare document similarity by vectors
- Compare word similarity by vectors

11

N-grams

Word-level unigrams

Text	Token Sequence	Token Value
One Two Three Four	1	One
One Two Three Four	2	Two
One Two Three Four	3	Three
One Two Three Four	4	Four

Word-level bigrams

Text	Token Sequence	Token Value
One Two Three Four	1	One Two
One Two Three Four	2	Two Three
One Two Three Four	3	Three Four

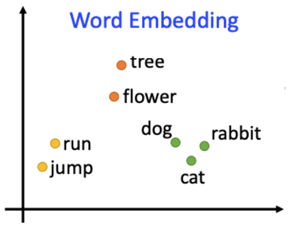
Word-level trigrams

Text	Token Sequence	Token Value
One Two Three Four	1	One Two Three
One Two Three Four	2	Two Three Four

12

Word2Vec

- Learning the word vectors from a large collection of documents
- Resulting word representation
  - Similar words are closer
  - The word vectors capture the semantic of the words
- The output vectors are also known as word embedding



1/11/21 13

13

One-hot encoding vs word embedding

1-of-N Encoding

apple = [1 0 0 0 0]

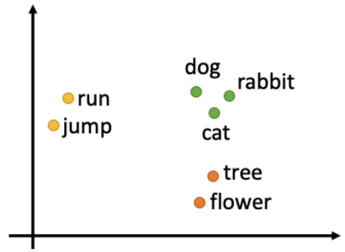
bag = [0 1 0 0 0]

cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

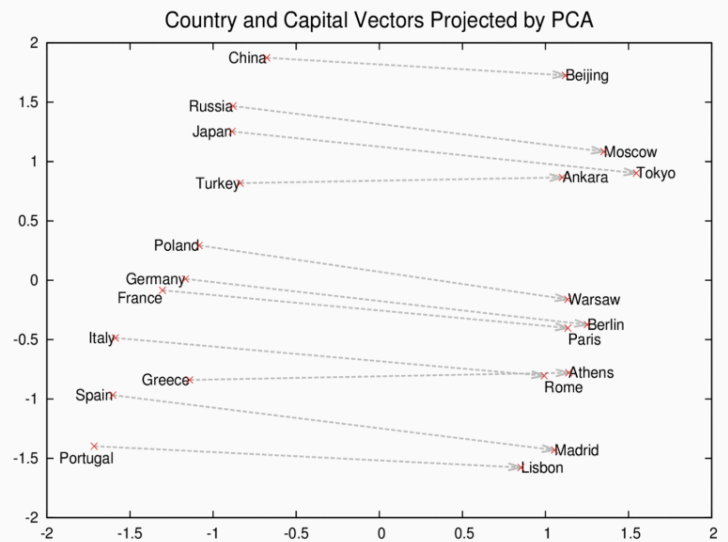
elephant = [0 0 0 0 1]

Word Embedding



1/11/21 14

14



15

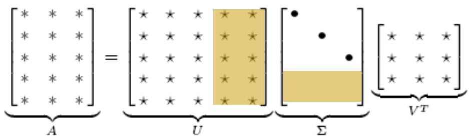
Word2Vec (training data: Chinese Wikipedia)

- 台灣 : 台北 = 法國 : ?
  - ⇒ 巴黎 (0.9350)
- 國民黨 : 馬英九 = 民進黨 : ?
  - ⇒ 陳水扁 (0.9844)
- 海賊王 : 魯夫 = 火影忍者 : ?
  - ⇒ 自來也 (0.7876)
- 爵士樂 : 紐奧良 = 鄉村音樂 : ?
  - ⇒ 納許維爾 (0.9718)
- 中研院 : 李遠哲 = 工研院 : ?
  - ⇒ 林同炎 (0.9381)
- 台灣 : 台灣大學 = 美國 : ?
  - ⇒ 約翰霍普金斯大學 (0.9976)

1/11/21 16

16

Singular Value Decomposition



- U and V are unitary matrices
  - I.e., $U^T U = I$; $V^T V = I$
- Σ is a diagonal matrix
  - The diagonal entries σ_i of Σ are known as the singular values of A, commonly listed in descending order

1/11/21 17

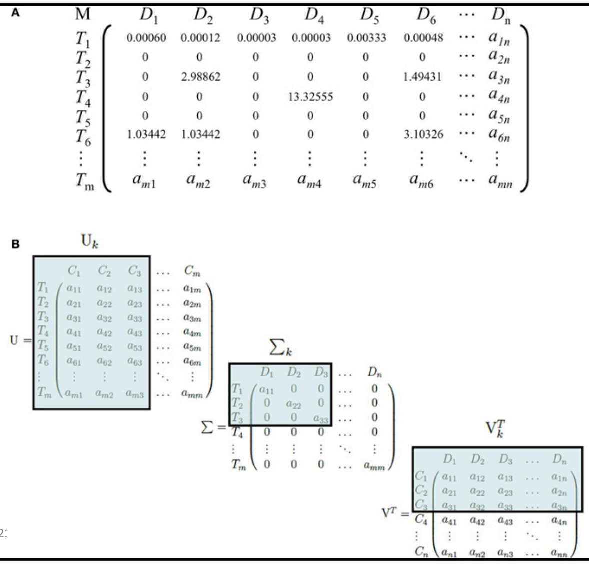
17

Reduced SVD

- If we retain only k **largest** singular values, and set the rest to 0
  - Then Σ is $k \times k$, U is $M \times k$, V^T is $k \times N$, and A_k is $M \times N$
- This is referred to as the reduced SVD
- It is the convenient (space-saving) and usual form for computational applications
- Among matrices with rank k, A_k he best approximation of A (in terms of Frobenius norm)

1/11/21 18

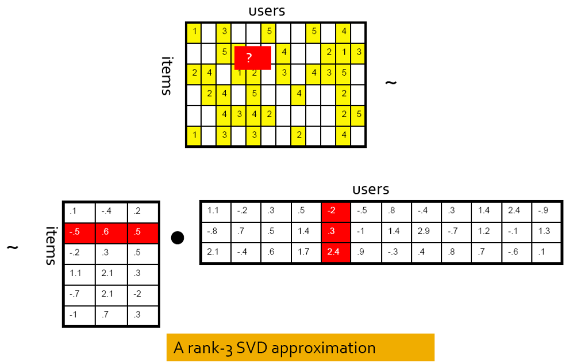
18



1/11/2: 19

19

Relationship with matrix factorization (in recommender systems)



1/11/21 20

20

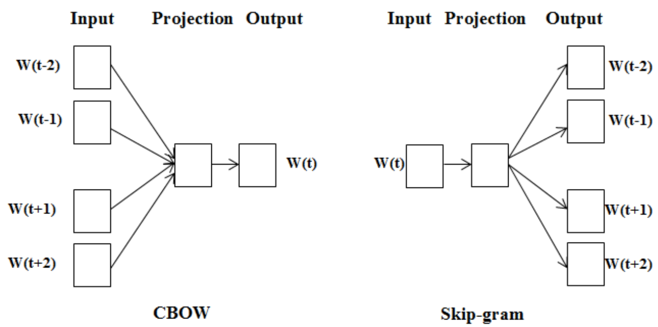
Counting-based vs prediction-based model

- Counting-based
  - Generate the word representation based on the co-occurrence counts
  - Perform matrix factorization
- Prediction-based
  - Predict the words based on the neighboring words
  - The word representation is a by-product of the prediction task

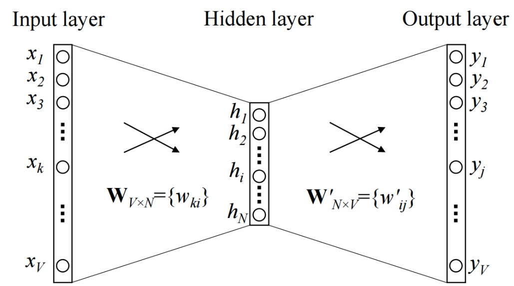
Two popular types of prediction-based model

- Continuous bag-of-words (CBOW)
  - Predict current word based on the context
- Skip-gram
  - Predict the context words based on the current word
- A word's context

CBOW vs Skip-gram



Simple CBOW

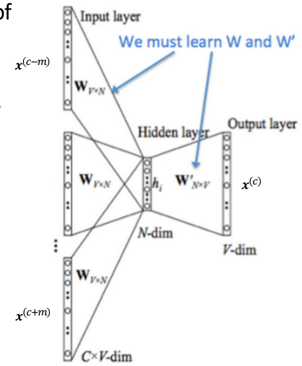


Given $(\boldsymbol{x}, \boldsymbol{y})$ as two neighboring words:

$$\hat{\boldsymbol{y}} = \mathrm{softmax}\left(\boldsymbol{W}'^{T}(\boldsymbol{W}^{T}\boldsymbol{x})\right)$$

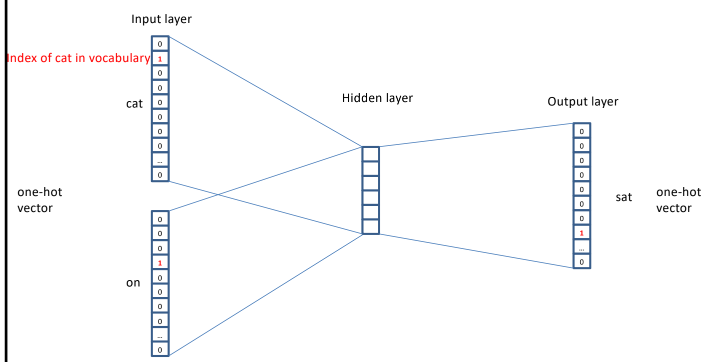
Continuous bag-of-words (CBOW)

- Called bag-of-words, because the order of words are lost
- Forward
  1. generate one-hot word vectors $\left(\boldsymbol{x}^{(c-m)}, \dots, \boldsymbol{x}^{(c-1)}, \boldsymbol{x}^{(c+1)}, \dots, \boldsymbol{x}^{(c+m)}\right)$ for the input context of size m
  2. generate word vectors of the context $\left(\boldsymbol{v}^{(c-m)}, \dots, \boldsymbol{v}^{(c-1)}, \boldsymbol{v}^{(c+1)}, \dots, \boldsymbol{v}^{(c+m)}\right)$, where $\boldsymbol{v}^{(i)} = \boldsymbol{W}^T \boldsymbol{x}^{(i)}$
  3. $\widetilde{\boldsymbol{v}} = \frac{\boldsymbol{v}^{(c-m)} + \dots + \boldsymbol{v}^{(c-1)} + \boldsymbol{v}^{(c+1)} + \dots + \boldsymbol{v}^{(c+m)}}{2m}$
  4. $\boldsymbol{z} = \boldsymbol{W'}^T \widetilde{\boldsymbol{v}}$
  5. $\widehat{\boldsymbol{x}}^{(c)} = \mathrm{softmax}(\boldsymbol{z})$
- Backprop
  – Compute error and perform back-prop to update $\boldsymbol{W}$ and $\boldsymbol{W'}$



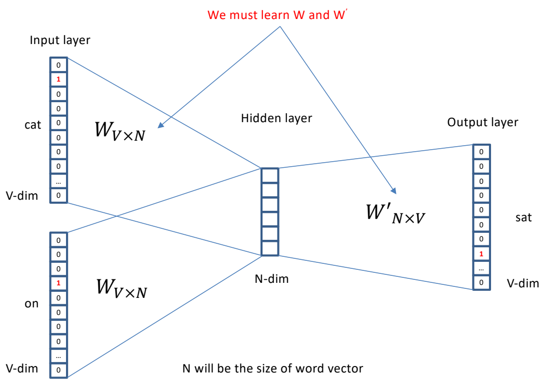
1/11/21 25

25



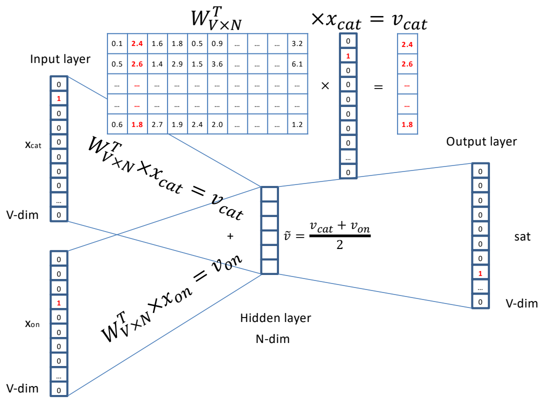
26

26



27

27

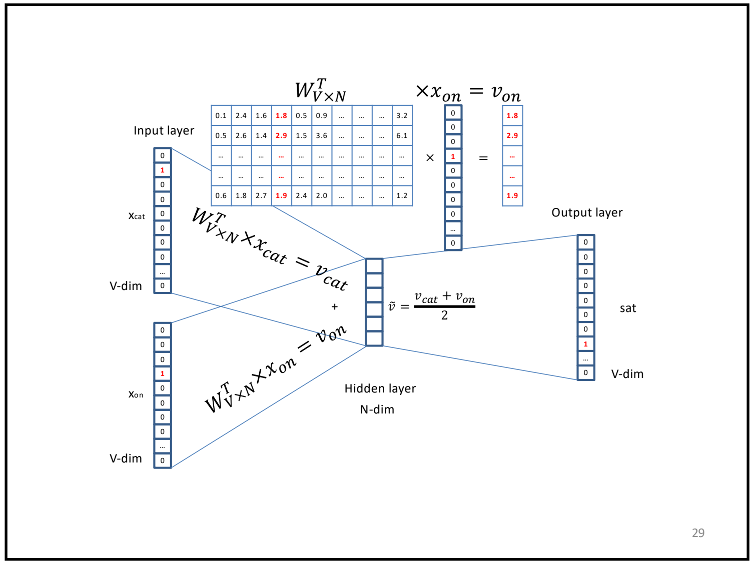


0.1	2.4	1.6	1.8	0.5	0.9	…	…	…	3.2
0.5	2.6	1.4	2.9	1.5	3.6	…	…	…	6.1
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
0.6	1.8	2.7	1.9	2.4	2.0	…	…	…	1.2

$\times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 2.6 \\ \dots \\ \dots \\ 1.8 \end{bmatrix}$

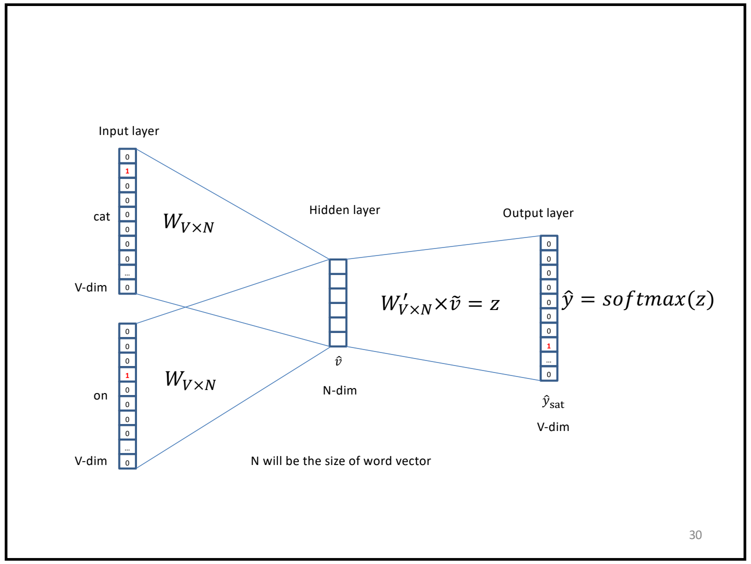
28

28



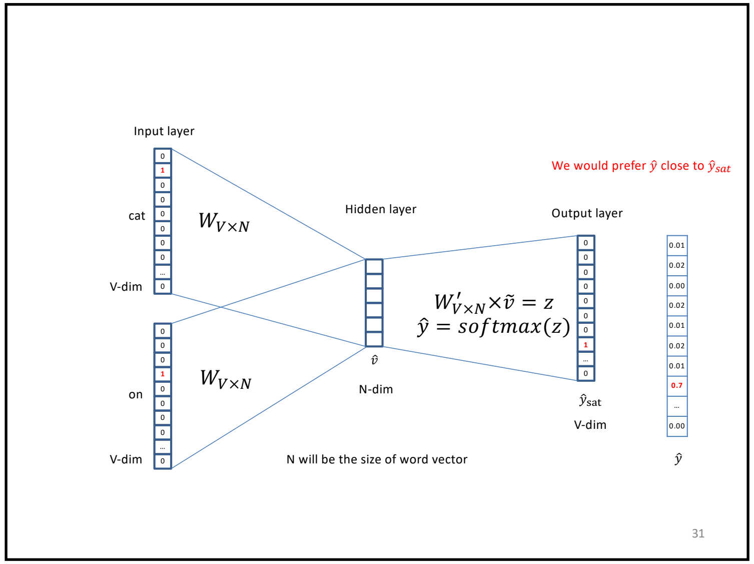
$W^T_{V\times N}$
$\times x_{on} = v_{on}$
0.1 2.4 1.6 1.8 0.5 0.9 ... ... 3.2
0.5 2.6 1.4 2.9 1.5 3.6 ... ... 6.1
0.6 1.8 2.7 1.9 2.4 2.0 ... ... 1.2
1.8 2.9 ... ... 1.9
Input layer
x_{cat}
V-dim
$W^T_{V\times N}\times x_{cat} = v_{cat}$
+
$W^T_{V\times N}\times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
x_{on}
V-dim
Hidden layer
N-dim
Output layer
sat
V-dim
29

29



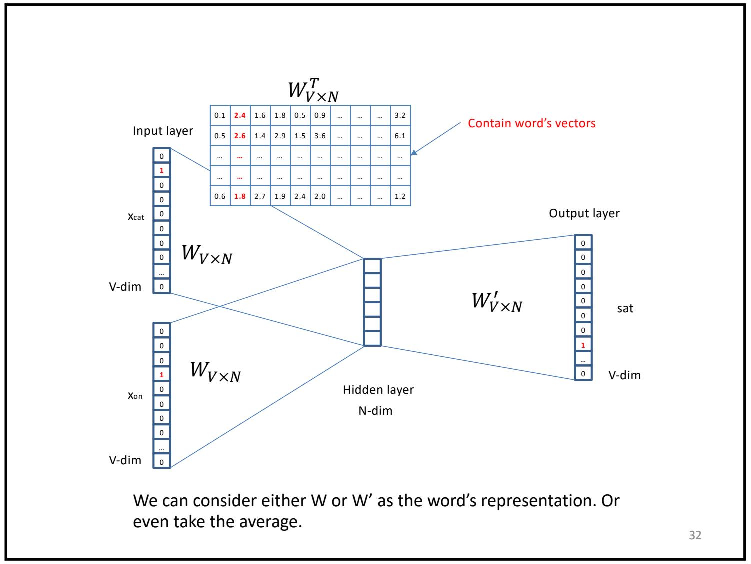
Input layer
cat
V-dim
$W_{V\times N}$
on
V-dim
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N}\times\hat{v} = z$
Output layer
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
N will be the size of word vector
30

30



Input layer
cat
V-dim
$W_{V\times N}$
on
V-dim
$W_{V\times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V\times N}\times\hat{v} = z$
$\hat{y} = softmax(z)$
Output layer
$\hat{y}_{sat}$
V-dim
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
0.01 0.02 0.00 0.02 0.01 0.02 0.01 0.7 ... 0.00
$\hat{y}$
N will be the size of word vector
31

31



$W^T_{V\times N}$
0.1 2.4 1.6 1.8 0.5 0.9 ... ... 3.2
0.5 2.6 1.4 2.9 1.5 3.6 ... ... 6.1
0.6 1.8 2.7 1.9 2.4 2.0 ... ... 1.2
Contain word's vectors
Input layer
x_{cat}
V-dim
$W_{V\times N}$
x_{on}
V-dim
$W_{V\times N}$
Hidden layer
N-dim
$W'_{V\times N}$
Output layer
sat
V-dim
We can consider either W or W' as the word's representation. Or even take the average.
32

32

Skip-gram

- Predict context words by the current word
- Or more precisely, if there are k context words, we predict ***one*** context word based on the current word for k times
- E.g., “this is a wonderful day”, window size=1
  - When current word is “is”, the context words are “this” and “a”
    - Use “is” to predict “this”
    - Use “is” to predict “a”
  - When the current word is “wonderful”, the context words are “a” and “day”
    - Use “wonderful” to predict “a”
    - Use “wonderful” to predict “day”

1/11/21 33

33

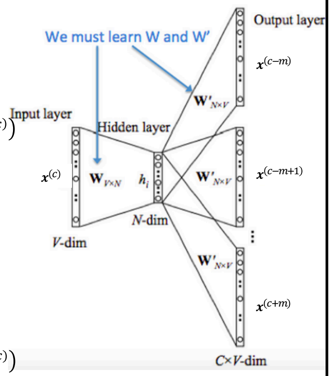
Skip-gram (cont’)

- For each (target, context) pair $x^{(c)}$ and $x^{(d)}$, we want:

$$\boldsymbol{x}^{(d)} \approx \text{softmax}\left(\boldsymbol{W}'^{T}\left(\boldsymbol{W}^{T}\boldsymbol{x}^{(c)}\right)\right)$$

- Objective:

$$J = -\log \mathrm{P}\left(x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)} | x^{(c)}\right)$$
$$= -\log \prod_{j=0, j\neq m}^{2m} P\left(x^{(c-m+j)} | x^{(c)}\right)$$
$$= -\log \prod_{j=0, j\neq m}^{2m} P\left(u^{(c-m+j)} | v^{(c)}\right)$$
$$= -\log \prod_{j=0, j\neq m}^{2m} \frac{\exp\left({u^{(c-m+j)}}^{T} v^{(c)}\right)}{\sum_{k=1}^{|V|} \exp(u_k^T v^{(c)})}$$
$$= -\sum_{j=0, j\neq \mathrm{m}}^{2m} {u^{(c-m+J)}}^{T} v^{(c)} + 2m \log \sum_{k=1}^{|V|} \exp\left(u_K^T v^{(c)}\right)$$



1/11/21 34

34

Negative sampling

- In the objective function, summing over $|V|$ is a huge cost
  - Every update and every objective function evaluation take $O(|V|)$
- Can we approximate the term? ➔ Negative sampling!
  - “Sample” negative examples from $P_n(w)$, whose probability matches the ordering of the frequency of the vocabulary

1/11/21 35

35

Turning prediction function of skip gram into logistic regression

- Positive instances: (w, c) pairs where w is a word and c is a context word
- Negative instances: (w, c') where c' is from negative sampling
- Training: find v_w, v_c, and $v_{c'}$ such that

$$P(w, c) = \frac{1}{1+\exp(-v_c^T v_w)} \approx 1, \text{ and}$$
$$P(w, c') = \frac{1}{1+\exp(-v_{c'}^T v_w)} \approx 0$$

1/11/21 36

36

We may apply similar concept to other domains

- As long as you can identify the "relevant object pairs", it is possible to apply similar concept (X2vec) to your domain

Example: Prod2Vec

- Build a recommendation algorithm for a e-commerce website based on clickstream
- Generate the embedding for the products and make recommendations based on the distance between product embeddings

Example: Node2Vec

- Generate node embeddings for the nodes in a graph

Example: Behavior2Vec

- On a e-commerce website, "viewing" a product *X* and "purchasing" a product *X* are very different behaviors
- Instead of generating embedding for products, we generate two embeddings for each product
  - Embedding of "buying" a product
  - Embedding of "viewing" a product
- Use different embeddings in different scenarios

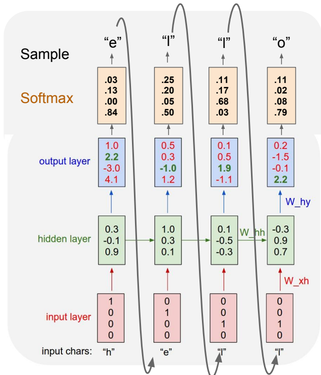
Behavior2Vec: Generating Distributed Representations of Users' Behaviors on Products for Recommender Systems
Hung-Hsuan Chen TKDD ACM Transactions on Knowledge Discovery from Data 12(4), 2018

Previously, we use "one-hot-encoded" words as input for a NN

Example:
Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 10 - 40 May 4, 2017

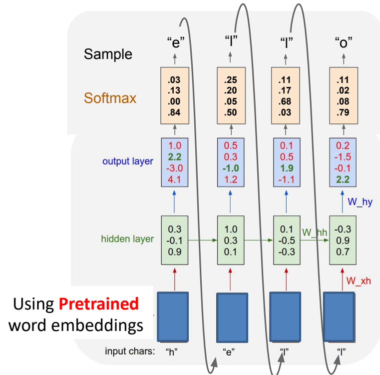
41

But we can use word embedding as the input for a NN

Example:
Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 10 - 40 May 4, 2017

42

Summary

- Word2Vec generates dense vector (instead of one-hot encoded vector) for each word
- Relationship between words may be captured by Word2Vec
- Word embeddings are sometimes pre-trained and regarded as the input features to other tasks
- We may apply similar concept to many fields

1/11/21 43

43