Text Processing

(good survey at https://github.com/kk7nc/Text_Classification)

2023. 8

Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 인터넷에서 다운받아 사용한 그림이나 수식들이 일부 있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

Text Processing

- **Applications**
  - Language translation
  - Sentiment analysis (positive, negative, neutral) (기쁨, 우울, 슬픔, 등)
  - Spam filtering (emails, messages)
  - Information extraction (from SNS, Web, Documents)
  - AI Speaker, Chatbot, Question answering
- **Language**
  - Token: basic text processing unit (character, **word**, n-gram, POS(part-of-speech, 품사), syllable(음절:자음+모음))
  - Corpus (말뭉치): collection of text (or document) dataset
  - Stop words (불용어): most common words in a language (“the”, “is”, etc)
- **Tools**
  - NLTK (Natural Language ToolKit): all sorts of tasks from tokenization, stemming, tagging, parsing, and beyond
  - KoNLPy (NLP library for Korean Language)
  - Gensim.Word2Vec(), tf.Keras.Embedding(), CountVectorizer(), BeautifulSoup()

Text Processing

- **Tokenization**
  - way of separating a piece of text into smaller units called tokens.
  - Tokens can be either words, characters, partial words (n-gram characters)
- **NLTK (Natural Language Toolkit)**

```python
from nltk.tokenize import word_tokenize

text = "After sleeping for four hours, he decided to sleep for another four"
tokens = word_tokenize(text)
print(tokens)
```

```
['After', 'sleeping', 'for', 'four', 'hours', ',', 'he', 'decided', 'to', 'sleep', 'for', 'another', 'four']
```

```python
example_sent = "This is a sample sentence, showing off the stop words filtration."
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
print(word_tokens)
print(filtered_sentence)
```

```
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

Text Processing

- **KoNLPy (Korean Natural Language Processing for Python: 코엔엘파이)**
  - Morpheme tokenizer (형태소 분석기):
    - Okt(Open Korea Text, 옛이름: Twitter)
    - Mecab(메캅), Komoran(코모란), Hannanum(한나눔)
    - Kkma (꼬꼬마)

```python
from konlpy.tag import Okt
okt=Okt()

print(okt.morphs("우리가 이 과제를 잘 할 수 있을까?"))
print(okt.pos("우리가 이 과제를 잘 할 수 있을까?", norm=True, stem=True))
print(okt.nouns("우리가 이 과제를 잘 할 수 있을까?"))
```

```
['우리', '가', '이', '과제', '를', '잘', '할', '수', '있을까', '?']
[('우리', 'Noun'), ('가', 'Josa'), ('이', 'Noun'), ('과제', 'Noun'), ('를', 'Josa'), ('자다', 'Verb'), ('하다', 'Verb'), (
['우리', '이', '과제', '수']
```

```python
from konlpy.tag import Okt
okt=Okt()
word_tags = okt.pos("우리가 이 과제를 잘 할 수 있을까?", norm=True, stem=True)
print(word_tags)
stop_words = [word[0] for word in word_tags if word[1]=="Josa"]
print (stop_words)
```

```
[('우리', 'Noun'), ('가', 'Josa'), ('이', 'Noun'), ('과제', 'Noun'), ('를', 'Josa'), ('자다', 'Verb'), ('하다', 'Verb'), (
['가', '를']
```

Text Processing - steps

- **Data preprocessing:**
  - Tokenization: convert sentences to words
  - Removing punctuations and tags
  - Removing stop words
  - Stemming: words are reduced to a root (ex: formalize -> formal, allowance -> allow)
  - Lemmatization (ex: is, are, am -> be, having -> have)
- **Feature Extraction**
  - One-hot encoding
  - BOW (Bag of Words): each sentence is represented with counts (or presence) of words (discards word order, hence ignoring the context and meaning in the document), sparse
    - CountVectorizer(), TfIdfVectorizer()
  - Word Embedding: represent each word with a meaningful n-dimensional vector, dense
    - Word2vec(): put words of similar meaning (or context) to closer places
    - tf.keras.Embedding(): vector representation is trained (only used as the first layer in deep learning)
- **Choose ML algorithms**
  - Classical ML (simple text classification) or Deep Learning (sentiment analysis or language translation)

Text Processing

- **BoW (Bag of Words)**

  - From http://www.datasciencecourse.org/notes/free_text/
    - Doc1 = "The goal of this lecture is to explain the basics of free text processing"
    - Doc2 = "The bag of words model is one such approach"
    - Doc3 = "Text processing via bag of words"

$$
X = \begin{array}{c}
\begin{array}{cccccccccccc} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} & \\ \end{array} \\
\left[\begin{array}{cccccccccccc} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{array}\right]
\begin{array}{l} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{array}
\end{array}
$$

$$
X = \begin{array}{c}
\begin{array}{ccccc} \text{the} & \text{is} & \text{of} & \text{goal} & \end{array} \\
\begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 & \\ 0.4 & 0.4 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & \end{bmatrix}
\end{array}
$$

$$
\mathrm{idf}_j = \log\left(\frac{\#\ \text{documents}}{\#\ \text{documents with word } j}\right)
$$

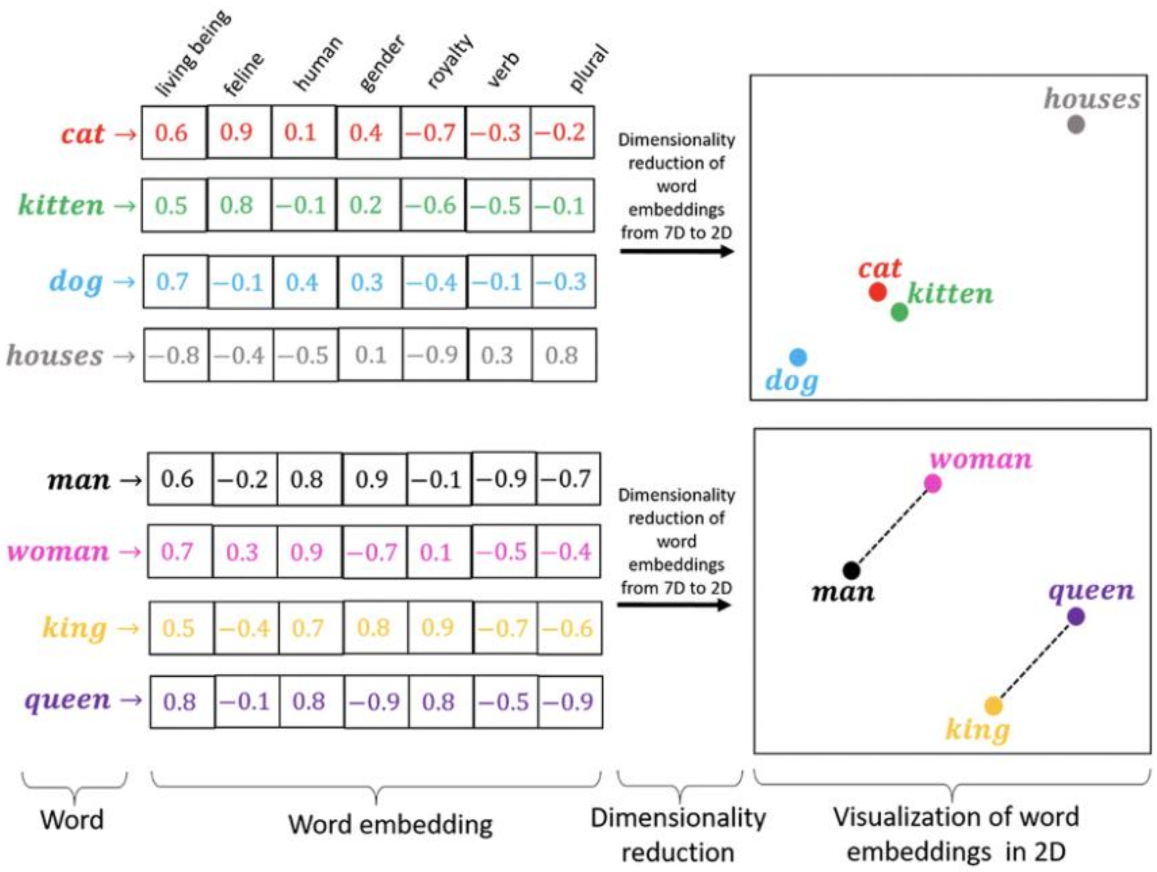
$$
\mathrm{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0
$$

$$
\mathrm{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405
$$

$$
\mathrm{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098
$$

Text Processing

- **Word Embedding**
  - Every word has a unique word embedding (or "vector"), and similar words end up with similar embedding values.
  - Word2vec from google: based on proximity
  - Glove: pre-trained (from Stanford)
  - Embedding() layer in deep learning: trained for the specific purpose




```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, LSTM, GRU

model = Sequential()
model.add(Embedding(input_dim=10000, output_dim=8,
                    input_length=maxlen))
model.add(LSTM(128))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

One-Hot encoding

- **One-Hot encoding**
  - Each word is written or encoded as one hot vector, with each one hot vector being **unique**.
  - One word is represented as a **vector,** therefore a sentence is represented as an array of vectors or a **matrix.**
  - A list of sentences will end up with a **three dimensional tensor** which can be fed to the Neural network.

The cat sat on the mat

The: [0 1 0 0 0 0 0]

cat: [0 0 1 0 0 0 0]

sat: [0 0 0 1 0 0 0]

on: [0 0 0 0 1 0 0]

the: [0 0 0 0 0 1 0]

mat: [0 0 0 0 0 0 1]

Bag of Words (BoW)

- A representation of text that describes the occurrence of words within a document
- Just **keep track of word counts** and **disregard the grammatical details** (word order), that's why called "bag" of words
- Occurrence of words:
  - Counts
  - Frequencies
- Limitations
  - Sparsity: little information with large representation space
  - Vocabulary: should be carefully designed due to impact on sparsity
  - Meaning: context and meaning of words (semantics) are disregarded
- Need text cleaning techniques
  - Ignoring cases, punctuations, frequent words ("a", "of", etc)
  - Fixing misspelled words
  - Reducing words to their stem (e.g. "play" from "playing"): stemization

Bag of Words (BoW)

- **N-gram**
  - N-token sequence of words
  - For example, 2-gram (more commonly called a bigram) is a two-word sequence of words (e.g. “please turn”, “turn your”, or “your homework”)
  - Often a simple bigram approach is better than a 1-gram bag-of-words model for tasks like documentation classification.
- **CountVectorizer()**
  - Document Term matrix
- **TfIdfVectorizer()**
  - Term frequency (tf) : frequency of the words
  - Inverse Document frequency (Idf) : how rare the word is across documents
- **Keras**
  - Keras.preprocessing.text.Tokenizer.fit_on_texts()
  - Keras.preprocessing.sequence.pad_sequences()

Document Term Matrix

- From http://www.datasciencecourse.org/notes/free_text/
  - Doc1 = "The goal of this lecture is to explain the basics of free text processing"
  - Doc2 = "The bag of words model is one such approach"
  - Doc3 = "Text processing via bag of words"

$$X = \begin{array}{cl} \begin{array}{ccccccccccc} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} \end{array} & \\ \left[\begin{array}{ccccccccccc} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{array}\right] & \begin{array}{l} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{array} \end{array}$$

TFIDF

- **Term frequency**
  - Counts of each word in a document
  - tf *i,j* = frequency of word *j* in document *i*
- **Inverse document frequency**
  - Term frequencies tend to be "overloaded" with very common words ("the", "is", "of", etc)
  - Idea if inverse document frequency weight words negatively in proportion to how often they occur in the entire set of documents

$$\mathrm{idf}_j = \log\left(\frac{\#\ \text{documents}}{\#\ \text{documents with word}\ j}\right)$$

TFIDF

$$X = \begin{array}{c} \begin{array}{ccccccccccc} \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} \end{array} \\ \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \ldots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{array}$$

$$X = \begin{array}{c} \begin{array}{cccc} \text{the} & \text{is} & \text{of} & \text{goal} \end{array} \\ \begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 & \\ 0.4 & 0.4 & 0 & 0 & \ldots \\ 0 & 0 & 0 & 0 & \end{bmatrix} \end{array}$$

$$\text{idf}_{\text{of}} = \log\left(\frac{3}{3}\right) = 0$$

$$\text{idf}_{\text{is}} = \log\left(\frac{3}{2}\right) = 0.405$$

$$\text{idf}_{\text{goal}} = \log\left(\frac{3}{1}\right) = 1.098$$

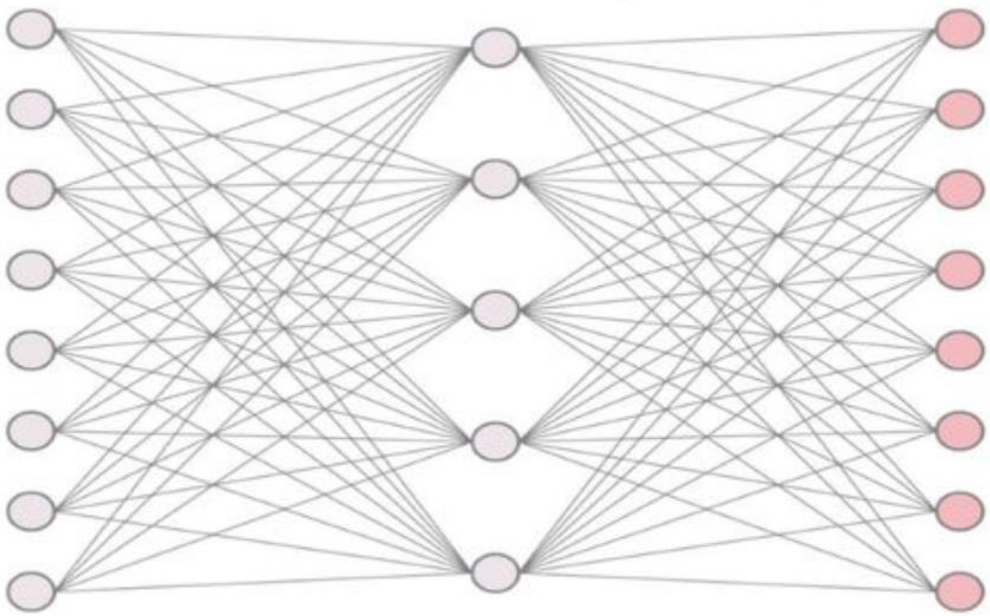
[note] In TfidfVectorizer(), if smooth_idf=true (default), the constant “1” is added to the numerator and denominator of the idf as if an extra document was seen containing every term in the collection exactly once, which prevents zero divisions: idf(t) = log [(1 + n) / (1 + df(t))] + 1

Word Embedding

- Techniques to compute Word embedding
  - Using Supervised learning
    - Take an NLP problem and try to solve it. In that pursuit as a side effect, you get word embedding.
    - Keras Embedding() layer
  - Using Self-supervised learning
    - Word2Vec (more popular)
    - Glove

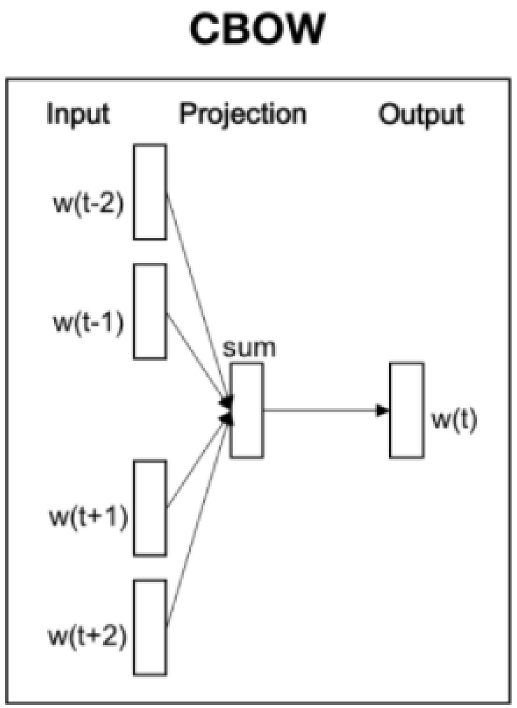
Word2Vec

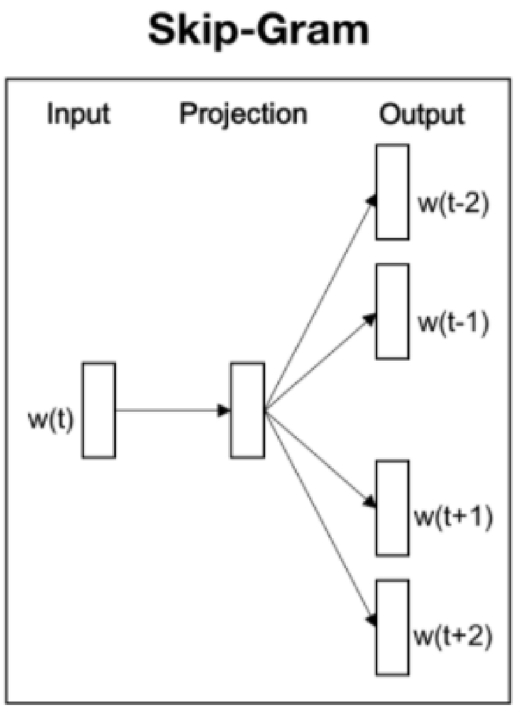
- Predict words using context
  - Concept: **'비슷한 위치 (근처)에서 등장하는 단어들은 비슷한 의미를 가진다'**
- Word2Vec Neural Network model
  - NN with single hidden layer
  - Often used for auto-encoder to compress input vector in hidden layer



Word2Vec

- **Two versions: CBOW(continuous bag of words) and Skip-Gram**
  - CBOW: 문맥 단어를 보고 기준 단어가 무엇인지 예측하는 모델
  - Skip-Gram: 기준 단어를 보고 어떤 문맥 단어가 등장할지 예측하는 모델 (in most cases, skip-gram is better)





https://wooono.tistory.com/244

Word2Vec - CBOW

- (ex) “I like playing football with my friends” with window = 2

center word context words

	y	X
	center word	context words
I like playing football with my friends	[1,0,0,0,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0]
I like playing football with my friends	[0,1,0,0,0,0,0]	[1,0,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,1,0,0,0]
I like playing football with my friends	[0,0,1,0,0,0,0]	[1,0,0,0,0,0,0] [0,1,0,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,1,0,0]
I like playing football with my friends	[0,0,0,1,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,1,0]
I like playing football with my friends	[0,0,0,0,1,0,0]	[0,0,1,0,0,0,0] [0,0,0,1,0,0,0] [0,0,0,0,0,1,0] [0,0,0,0,0,0,1]
I like playing football with my friends	[0,0,0,0,0,1,0]	[1,0,0,1,0,0,0] [0,0,0,0,1,0,0] [0,0,0,0,0,0,1]
I like playing football with my friends	[0,0,0,0,0,0,1]	[0,0,0,0,1,0,0] [0,0,0,0,0,1,0]

<u>Training samples</u>

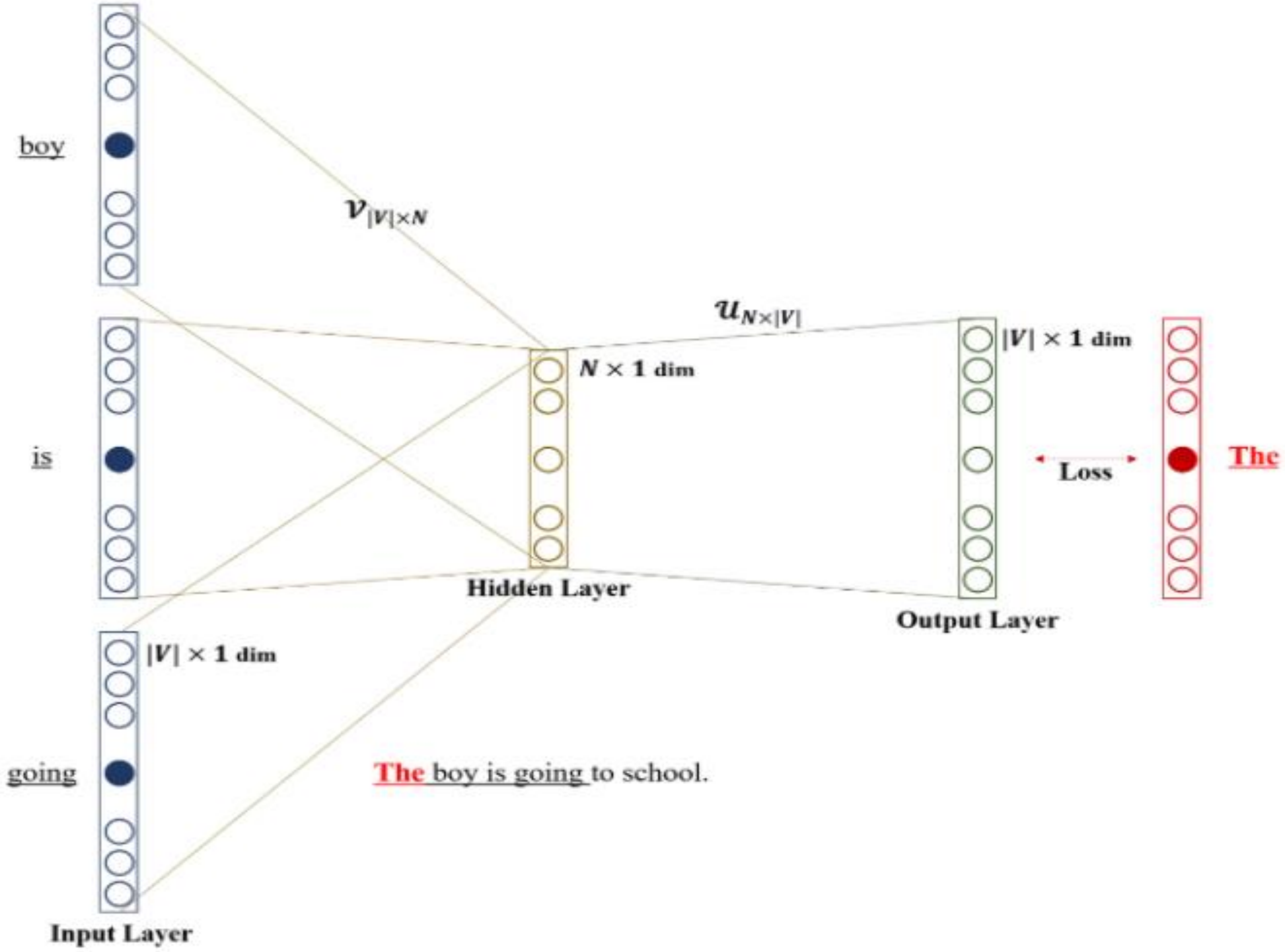
(I, like)
(I, playing)

(like, I)
(like, playing)
(like, football)

(playing, I)
(playing, like)
(playing, football)
(playing, with)
.
.
.
.
.
.
.
.
.
.
.
.
.

Word2Vec - CBOW

- **CBOW architecture**



Word2Vec - CBOW

- **CBOW Training**

1. (Input Layer) context word 각각의 one-hot vector를 입력

$$x^{(c-m)}, \dots, x^{(c-1)}, x^{(c+1)}, \dots, x^{(c+m)} \in \mathbb{R}^{|V|}$$

2. (Input Layer -> Hidden Layer) v = V^T * input -> (N-dim vector)

$$v_{c-m} = \mathcal{V}x^{(c-m)}, \dots, v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^n$$

3. (Hidden Layer) averaging v's of Step.2 -> v_hat

$$\hat{v} = \frac{v_{c-m} + \cdots + v_{c+m}}{2m} \in \mathbb{R}^n$$

4. (Hidden Layer -> Output Layer) z = U * v_hat

$$z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$$

5. (Output Layer) y_hat = softmax(z)

$$\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$$

6. (Loss) error = y_hat - y(center word)

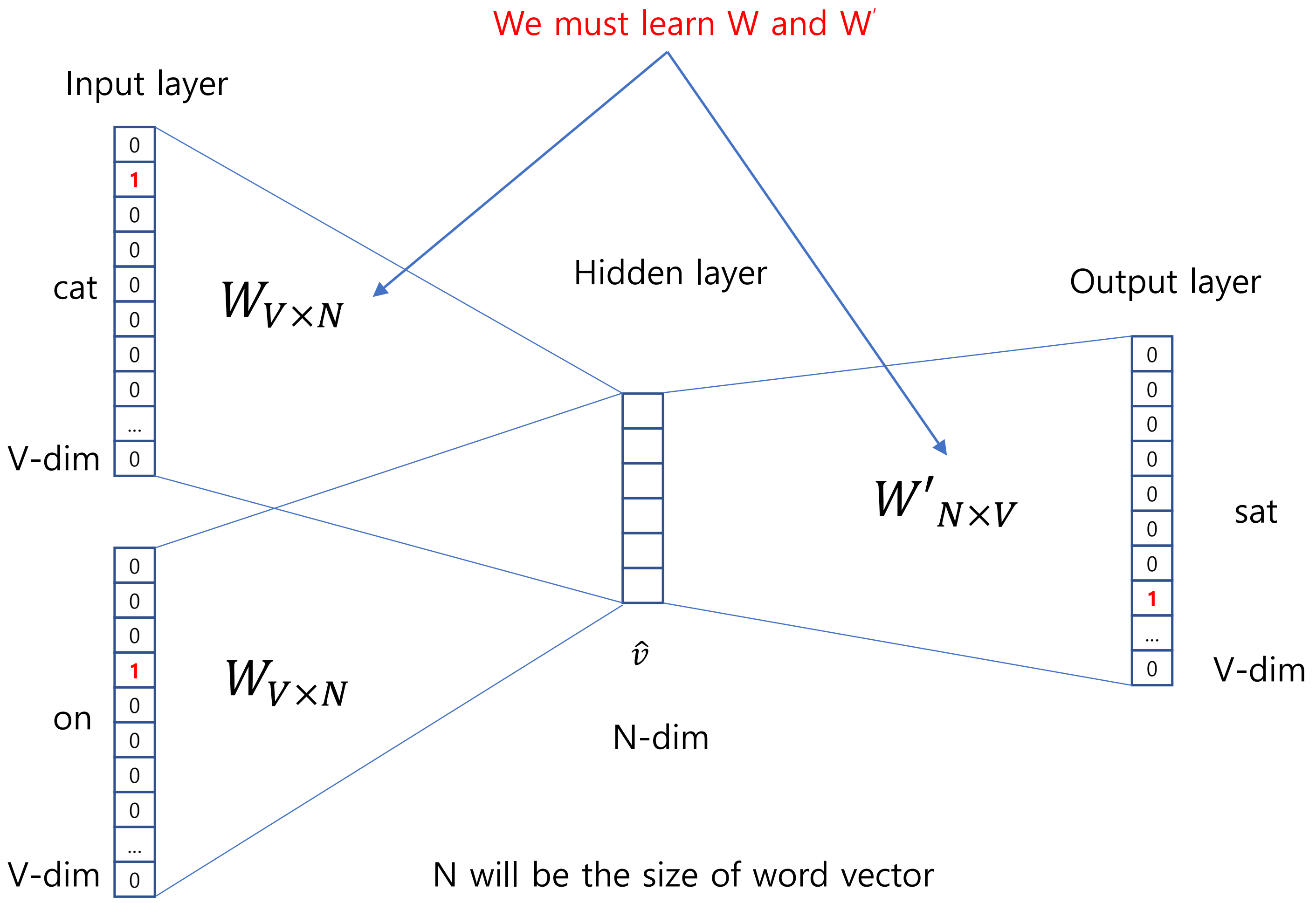
$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log \hat{y}_j = -y_i \log \hat{y}_i$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\hat{y}, y)$$

https://wooono.tistory.com/244

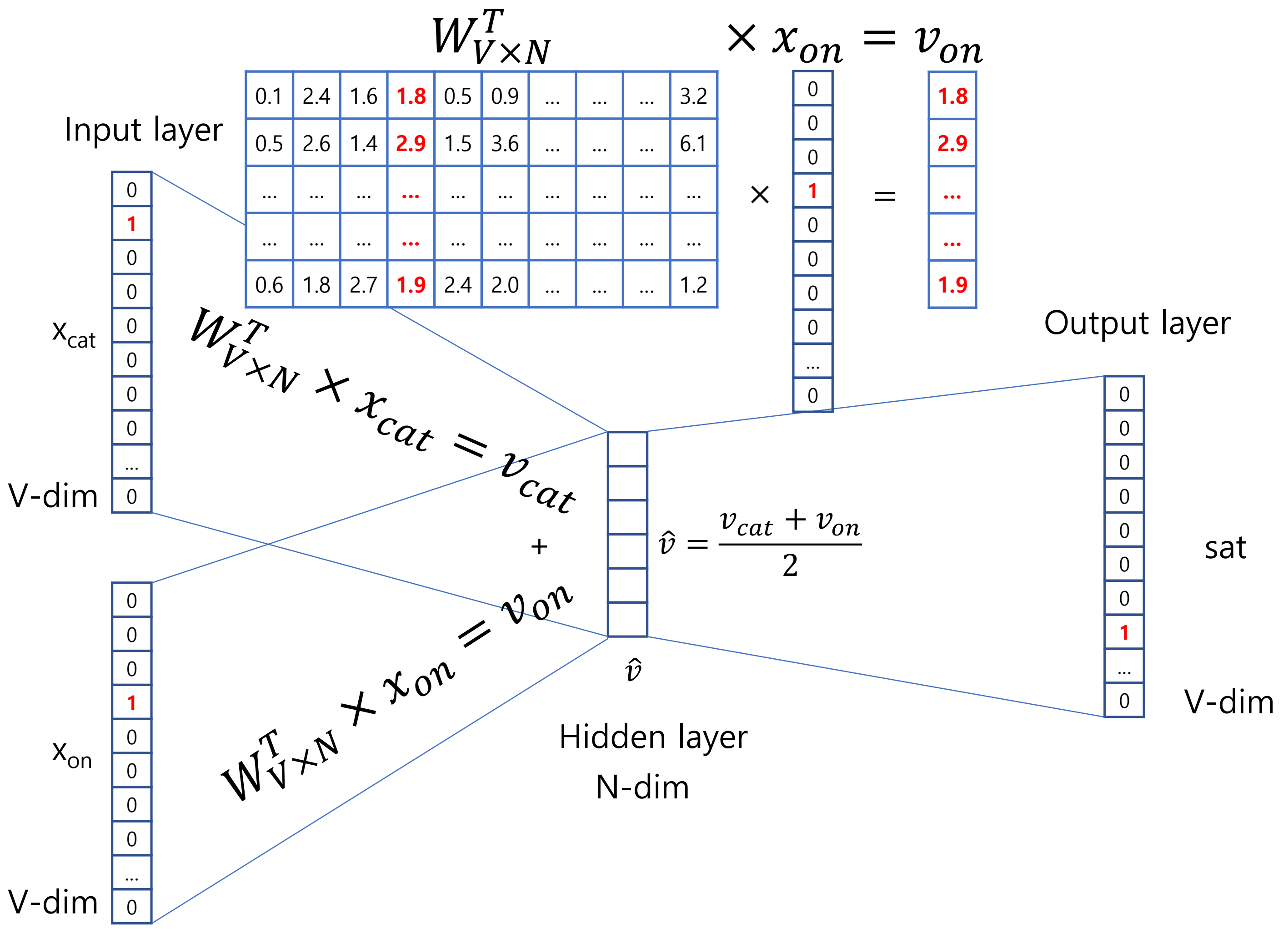
Word2Vec - CBOW

- (ex) "The cat sat on floor" with window_size = 1



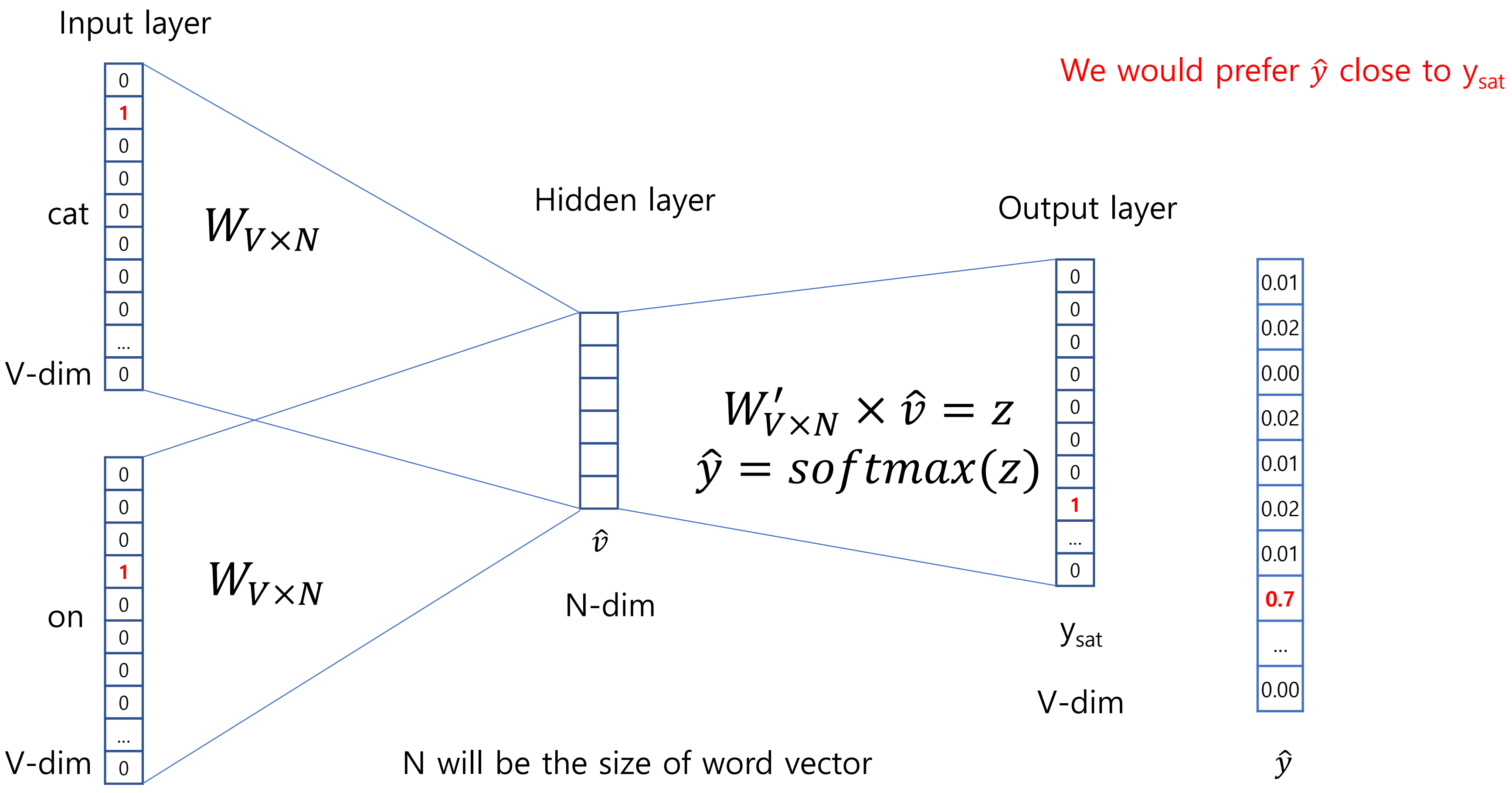
Word2Vec - CBOW

- (ex) “The cat sat on floor” with window_size = 1



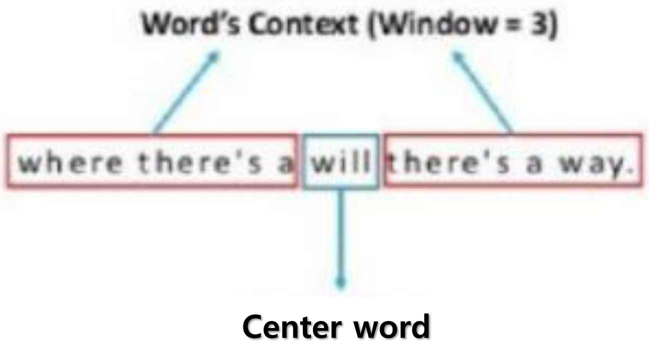
Word2Vec - CBOW

- (ex) "The cat sat on floor" with window_size = 1

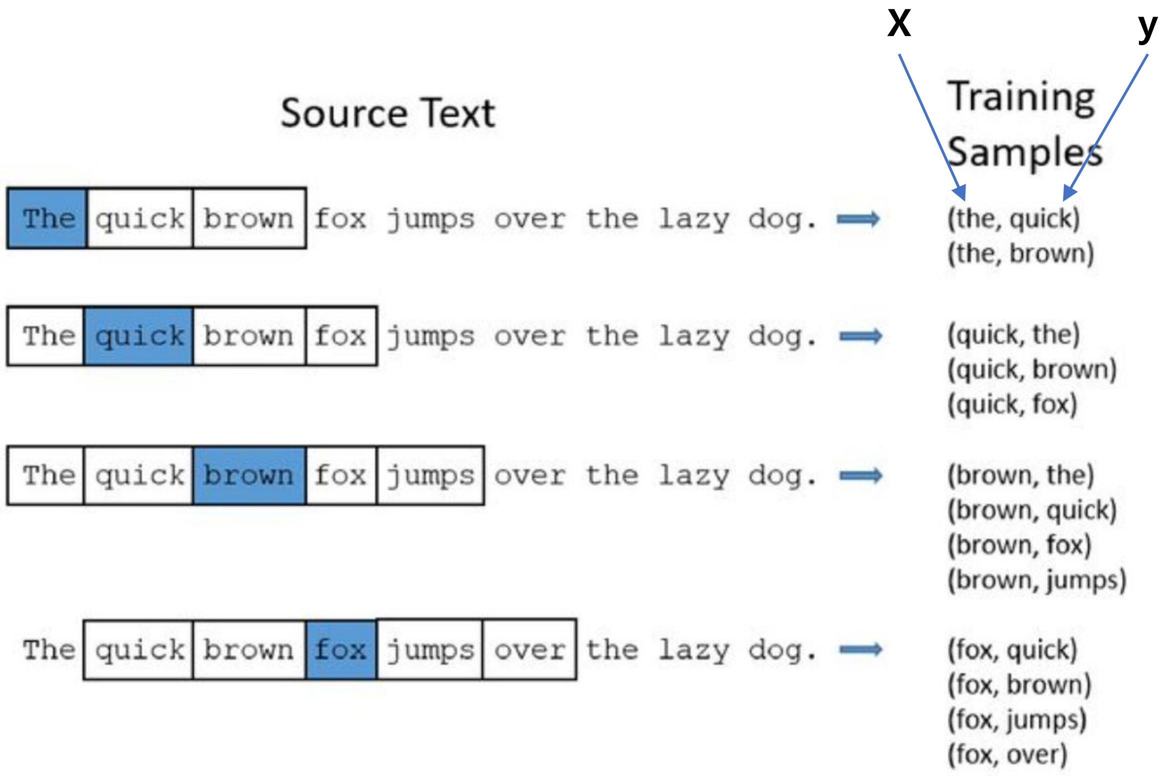


Word2Vec – Skip Gram

- Predict neighbors of a center word using Skip-gram model

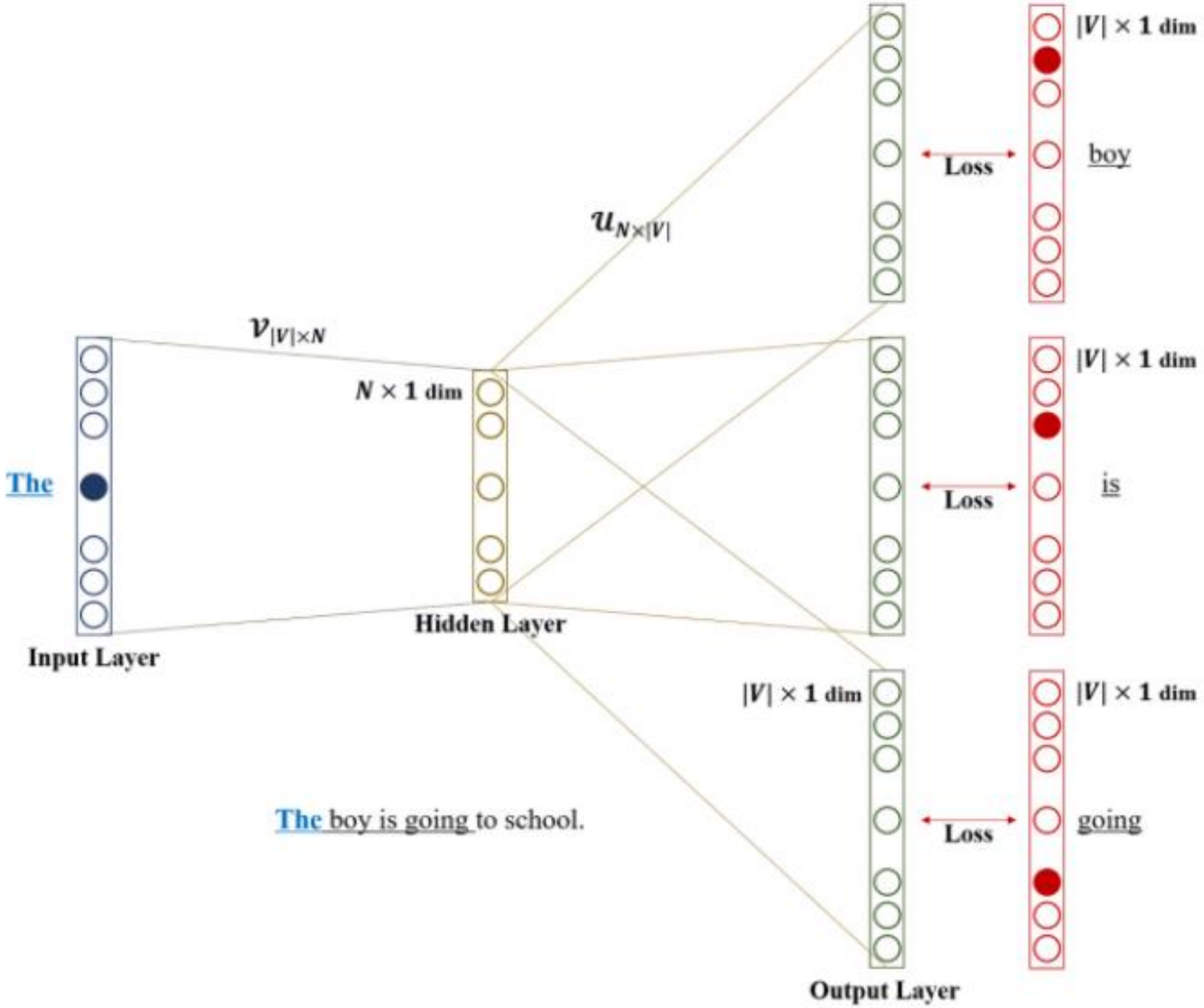


- Collection of Training samples
  - with a window of size 2



Word2Vec – Skip Gram

- **Skip-Gram architecture**



Word2Vec - Skip-Gram

- **Skip-Gram Training**

1. (Input Layer) center word 각각의 one-hot vector를 입력

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

2. (Input Layer -> Hidden Layer) v_c = V^T * input -> (N-dim vector)

$$v_c = \mathcal{V}x \in \mathbb{R}^n$$

3. (Hidden Layer -> Output Layer) z = U * v_c

$$z = \mathcal{U}v_c \in \mathbb{R}^{|V|}$$

4. (Output Layer) y_hat = softmax(z)

$$\hat{y} = softmax(z) \in \mathbb{R}^{|V|}$$

5. error e_c= y_hat – y(context word)

   위의 Step 을 모든 context word (주변단어) 에 대해 실행

6. (Loss) prediction error = $\sum_{2m}$ (ec)

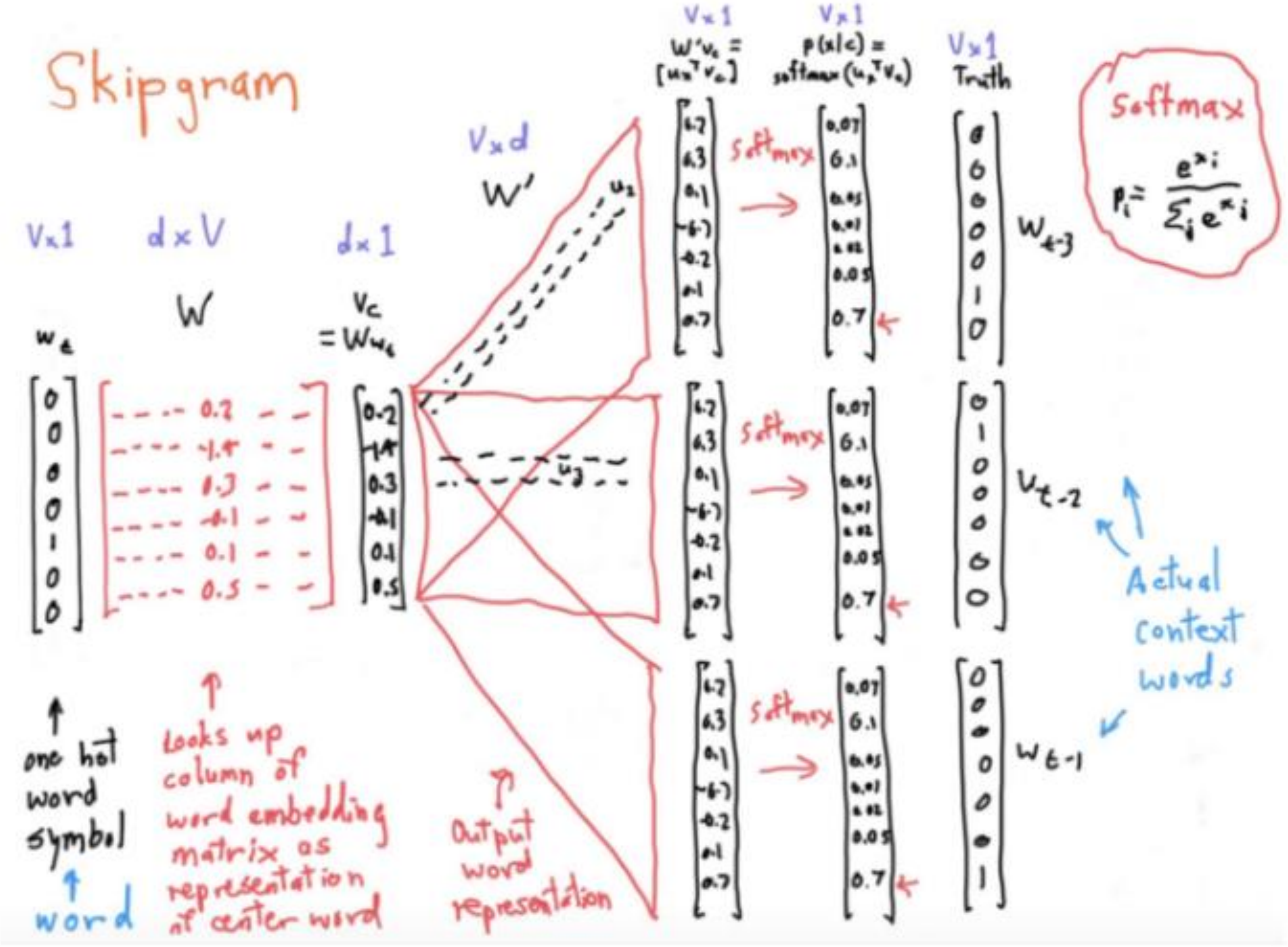
$$\mathcal{L}(\hat{y}, y) = -\sum_{j=0, j\neq m}^{2m} \sum_{k=1}^{|V|} y_k^{(c-j)} \log \hat{y}_k^{(c-j)} = -\sum_{j=0, j\neq m}^{2m} y^{(c-j)} \log \hat{y}^{(c-j)}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\hat{y}, y)$$

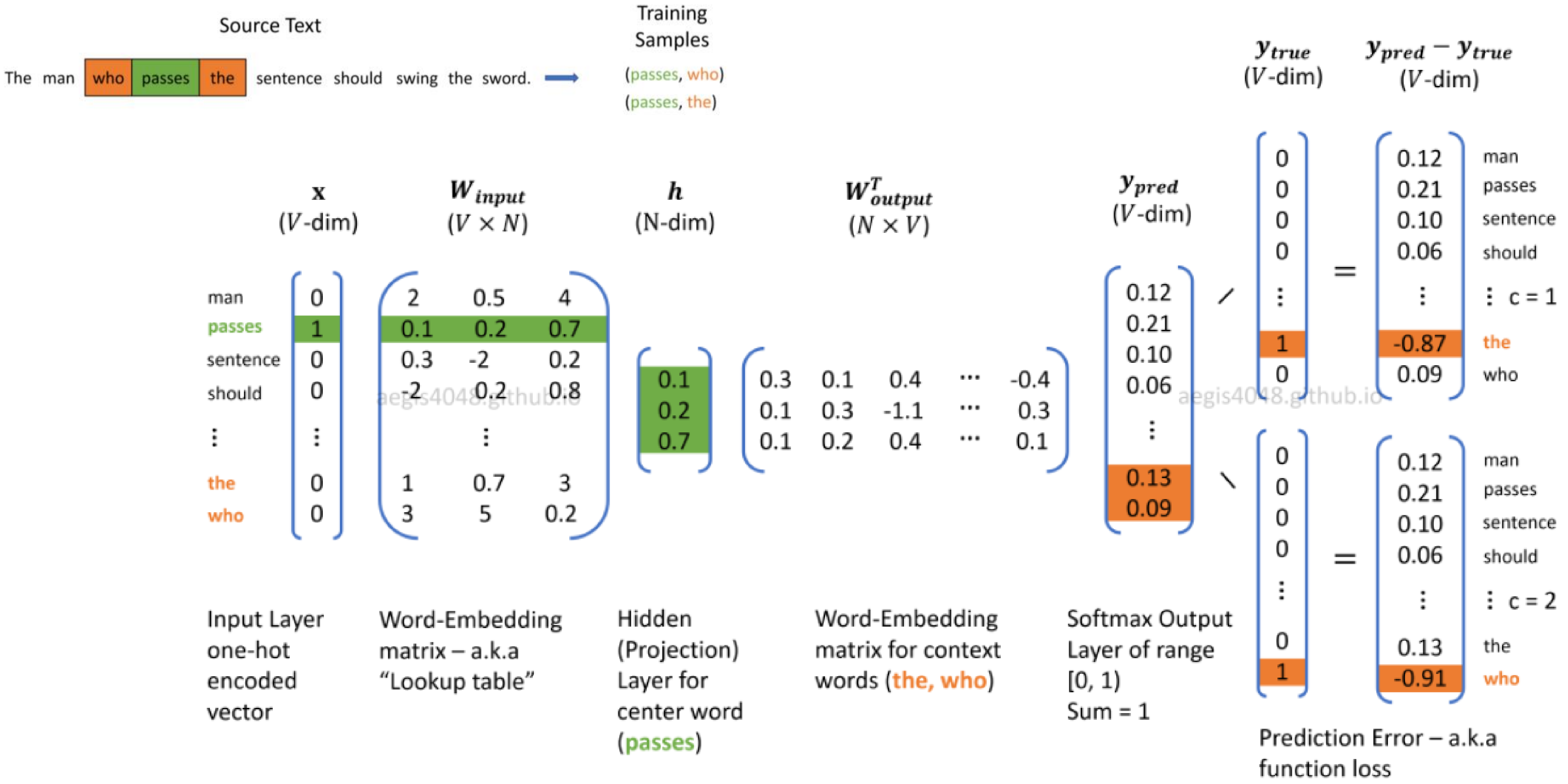
http://cs224d.stanford.edu/lecture_notes/notes1.pdf
https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

Word2Vec – Skip Gram

- **Summary**



Skip Gram – model example



Skip Gram – Cost function

$$\underset{\theta}{\operatorname{argmax}}\ p(w_1, w_2, \ldots, w_C | w_{center}; \theta)$$

to simplify derivation

$$\underset{\theta}{\operatorname{argmax}}\ log\, p(w_1, w_2, \ldots, w_C | w_{center}; \theta)$$

h : hidden layer word vector

softmax, naïve assumption

$$\underset{\theta}{\operatorname{argmax}}\ log \prod_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

$$p(w_{context} | w_{center}; \theta) = \frac{exp(W_{output_{(context)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

max -> min

$$\boxed{J(\theta; w^{(t)}) = -log \prod_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}}$$

c : index of the context words around the center word (w_t)
C : window size

$$= -\sum_{c=1}^{C} log \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)} \quad = -\sum_{c=1}^{C} (W_{output_{(c)}} \cdot h) + C \cdot log \sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)$$

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

Skip Gram – Cost function

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta \cdot \frac{\partial J}{\partial W_{input}} \qquad \frac{\partial J}{\partial W_{input}} = x \cdot (W_{output}^{T} \sum_{c=1}^{C} e_c)$$

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta \cdot \frac{\partial J}{\partial W_{output}} \qquad \frac{\partial J}{\partial W_{output}} = h \cdot \sum_{c=1}^{C} e_c$$

$$\boxed{\begin{aligned} W_{input}^{(new)} &= W_{input}^{(old)} - \eta \cdot x \cdot (W_{output}^{T} \sum_{c=1}^{C} e_c) \\ W_{output}^{(new)} &= W_{output}^{(old)} - \eta \cdot h \cdot \sum_{c=1}^{C} e_c \end{aligned}}$$

$$\sum_{c=1}^{C} e_c = \overset{c=1}{\begin{bmatrix} 0.12 \\ 0.21 \\ 0.10 \\ 0.06 \\ \vdots \\ -0.87 \\ 0.09 \end{bmatrix}} + \overset{c=2}{\begin{bmatrix} 0.12 \\ 0.21 \\ 0.10 \\ 0.06 \\ \vdots \\ 0.13 \\ -0.91 \end{bmatrix}} = \begin{bmatrix} 0.24 \\ 0.42 \\ 0.20 \\ 0.12 \\ \vdots \\ -0.74 \\ -0.82 \end{bmatrix} \begin{array}{l} \text{man} \\ \text{passes} \\ \text{sentence} \\ \text{should} \\ \vdots \\ \text{the} \\ \text{who} \end{array}$$

prediction error

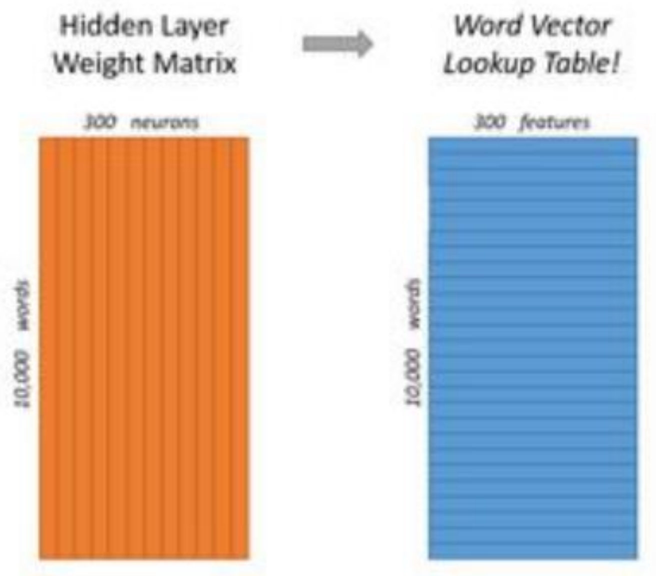
https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

Word Embedding

- **Build a vocabulary of words from training documents**
  - (e.g.) a vocabulary of 10,000 unique words
- **Represent an input word as a one-hot vector**
  - This vector will have 10,000 components (one for every word in our vocabulary)
  - This vector will have "1" in the position corresponding to the word, say "ants", and 0s in all of the other positions.
- **No activation function for hidden layer neurons, but output neurons use softmax.**
- **When *training* the network with word pairs, the input is a one-hot vector and output is also a one-hot vector representing the output word.**
- **When evaluating the network on an input word, the output vector will actually be a probability distribution (i.e., a bunch of floating point values, not a one-hot vector).**

Word Embedding

- **Hidden layer is represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron)**
  - 300 features
- **Rows of this weight matrix are the word vectors!**



Word Embedding

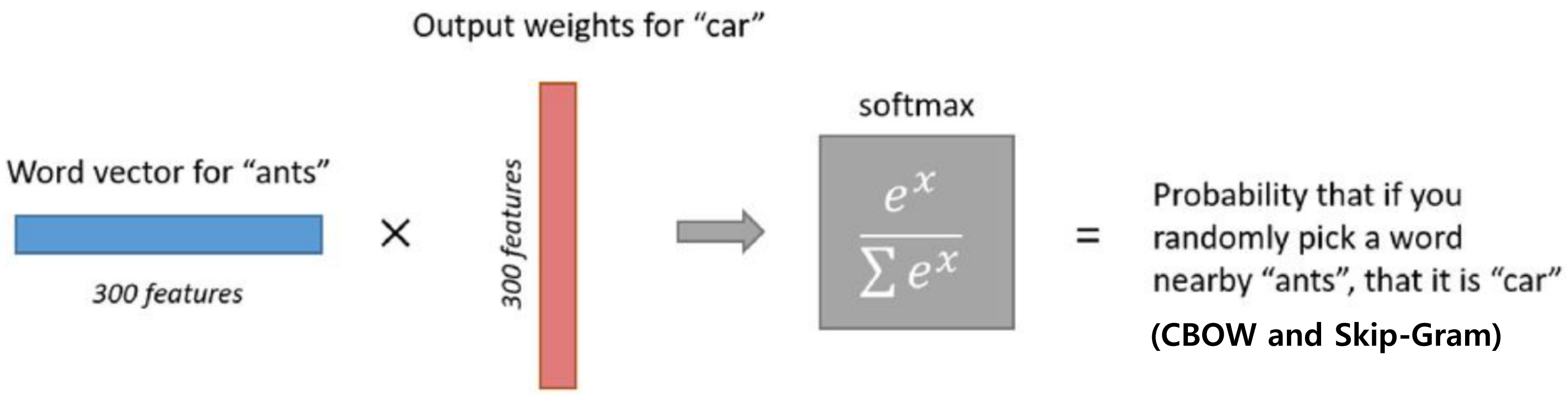
- **If we multiply a 1x10,000 one-hot vector by a 10,000x300 matrix, it will effectively just *select* the matrix row corresponding to the "1"**

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

- **Hidden layer is really just operating as a lookup table !!**
- **The output of the hidden layer is just the "word vector" for the input word.**

Word Embedding

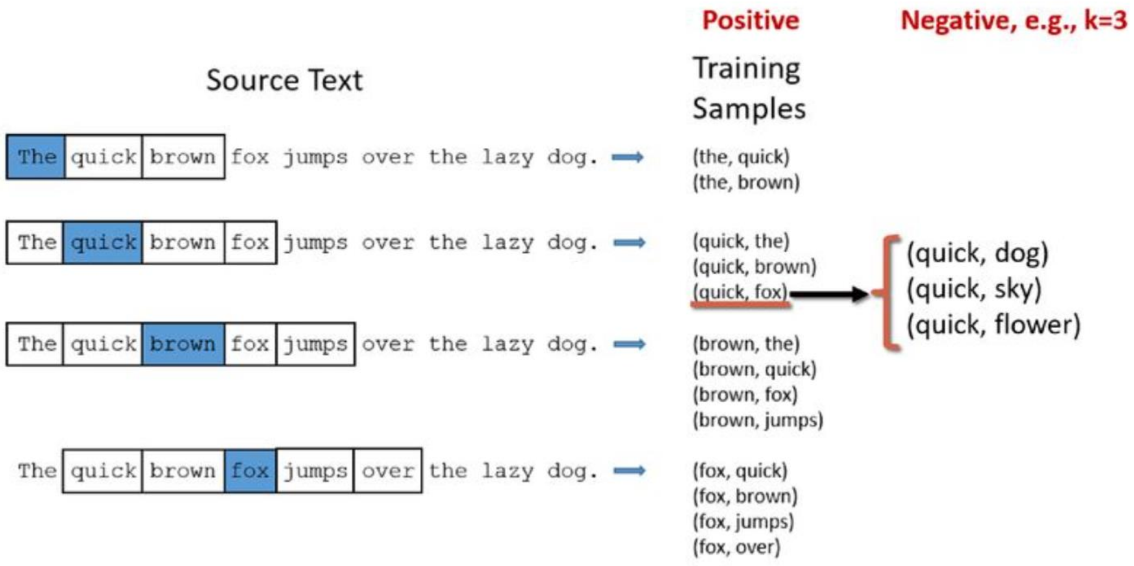
- The output layer is a softmax regression classifier
- Each output neuron has a weight vector which it multiplies against the word vector from the hidden layer, then it applies the function exp(x) to the result. Finally, in order to get the outputs to sum up to 1, we divide this result by the sum of the results from *all* 10,000 output nodes.



(CBOW and Skip-Gram)

Negative Sampling

- **Softmax function requires too much computation for its denominator (say, summation of 10,000 terms)**
- **When training, negative sampling considers only a small number of negative words (let's say 5) including the positive word for this summation (normally choose more than 20 based on some probability distribution)**
  - "negative" word is one for which network outputs "0" and "positive" word is one for "1"
  - 즉, 파라미터 조정할 때 전체 단어 집합이 아닌 일부 단어만 조정 (주변 단어(positive)들은 모두 조정하지만 그렇지 않은 단어들(negative)은 일부만 조정)



Negative Sampling

- In the Skip-gram case, we build a new objective function that tries to maximize both P(D=1|w,c,θ) and P(D=0|w,c,θ). We model P(D=1|w,c,θ) and P(D=0|w,c,θ) as follows:

$$P(D=1|w,c,\theta) = \sigma({u_c}^T v_w) = \frac{1}{1+e^{-{u_c}^T v_w}}$$

Probability that the word pair (w,c) is in corpus

$$P(D=0|w,c,\theta) = 1 - P(D=1|w,c,\theta) = 1 - \frac{1}{1+e^{-{u_c}^T v_w}} = \frac{1}{1+e^{{u_c}^T v_w}} = \sigma(-{u_c}^T v_w)$$

Probability that the word pair (w,c) is not in corpus

- We take a simple maximum likelihood approach of these two probabilities. (Here we take θ to be the parameters of the model, and in our case it is V and U.)

$$\begin{aligned}
\theta &= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1 - P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1 - P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1 - \frac{1}{1+\exp(-u_w^T v_c)}) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})
\end{aligned}$$

Θ: matrix U, V

http://cs224d.stanford.edu/lecture_notes/notes1.pdf
https://wikidocs.net/69141
https://dalpo0814.tistory.com/6

Negative Sampling

- Unnatural sentences that should get a low probability of ever occurring. We can generate D˜ on the fly by randomly sampling this negative from the word bank. Our new objective function would then be:

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+\exp(u_w^T v_c)}\right) = \log\sigma(u_{c-m+j}^T \cdot v_c) + \sum_{k=1}^{K} \log\sigma(-\tilde{u}_k^T \cdot v_c)$$

D: positive sample space

D~: negative sample space

- Original Skip-gram

$$\begin{aligned} \text{minimize } J &= -\log P(w_{c-m},\ldots,w_{c-1},w_{c+1},\ldots,w_{c+m}|w_c) \\ &= -\log \prod_{j=0,j\neq m}^{2m} P(w_{c-m+j}|w_c) \\ &= -\log \prod_{j=0,j\neq m}^{2m} P(u_{c-m+j}|v_c) \\ &= -\log \prod_{j=0,j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|}\exp(u_k^T v_c)} \\ &= -\sum_{j=0,j\neq m}^{2m} u_{c-m+j}^T v_c + 2m\log\sum_{k=1}^{|V|}\exp(u_k^T v_c) \end{aligned}$$

$$-\sum_{j=0,j\neq m}^{2m} U_{c-m+j}\cdot v_c + 2mlog\sum_{j=1}^{|V|} e^{U_j\cdot v_c}$$

- Skip-gram with Negative Sampling (SGNS)

$$\boxed{\sum_{j=0,j\neq m}^{2m} log(\sigma(U_{c-m+j}\cdot v_c)) + \sum_{j=1}^{K} log(\sigma(-\tilde{U}_j\cdot v_c))}$$

Only K samples

Word Analogies

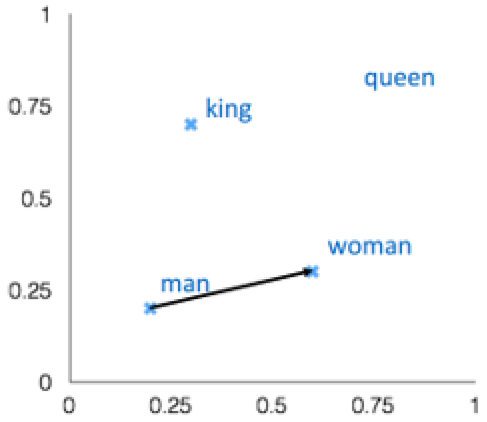
- **Interesting result**

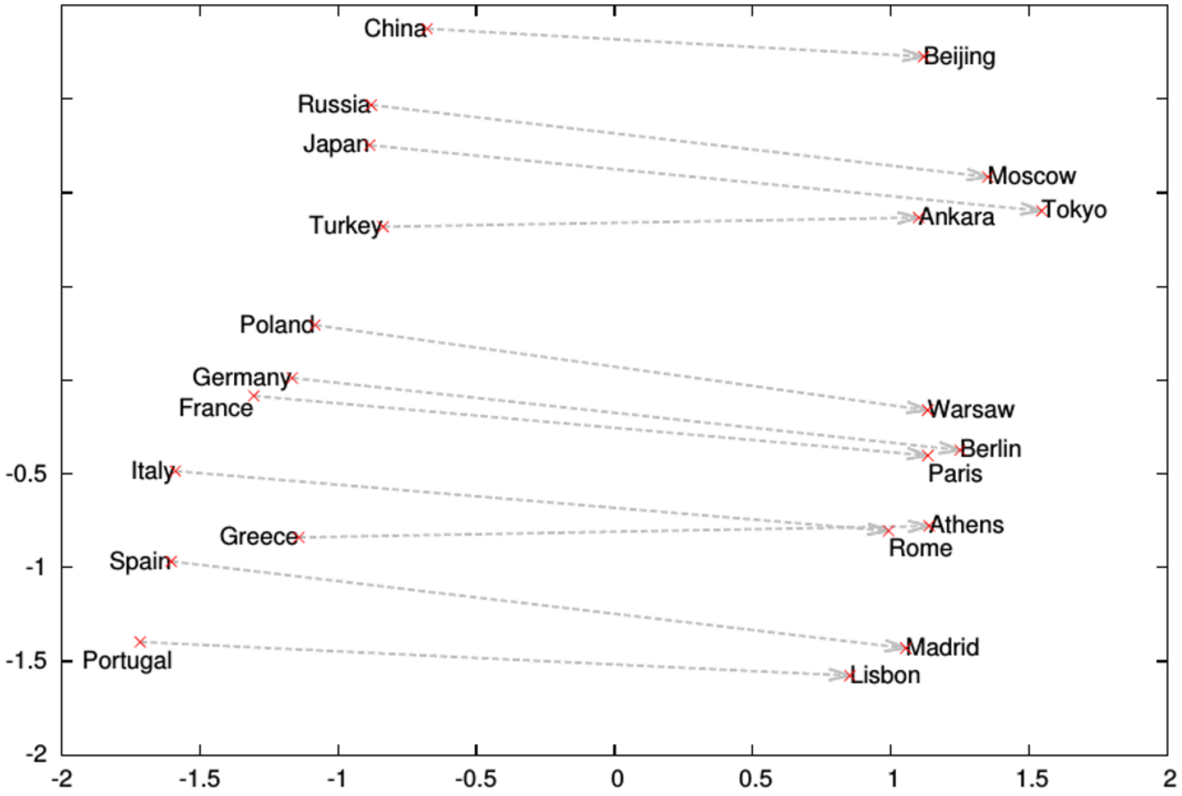
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:? → $$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

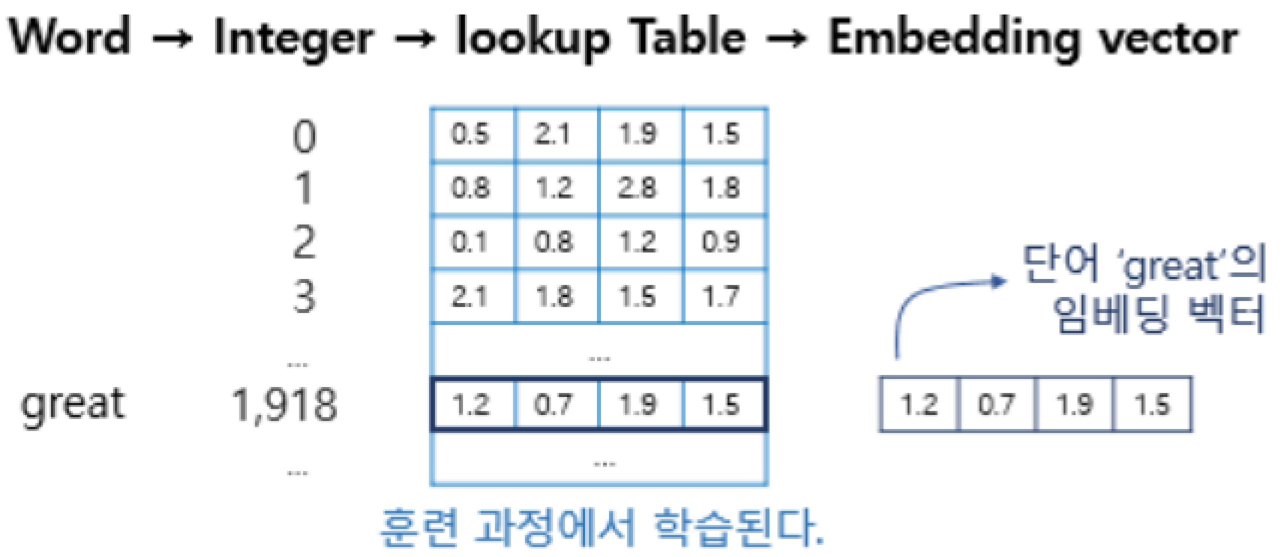
+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]





Keras Embedding() Layer

- A word embedding can be learned as part of a deep learning model. This can be a slower approach, but tailors the model to a specific training dataset.
- Embedding layer is just a Look-up Table.
  - All the words should be encoded as integers to be used as input.



Keras Embedding() Layer

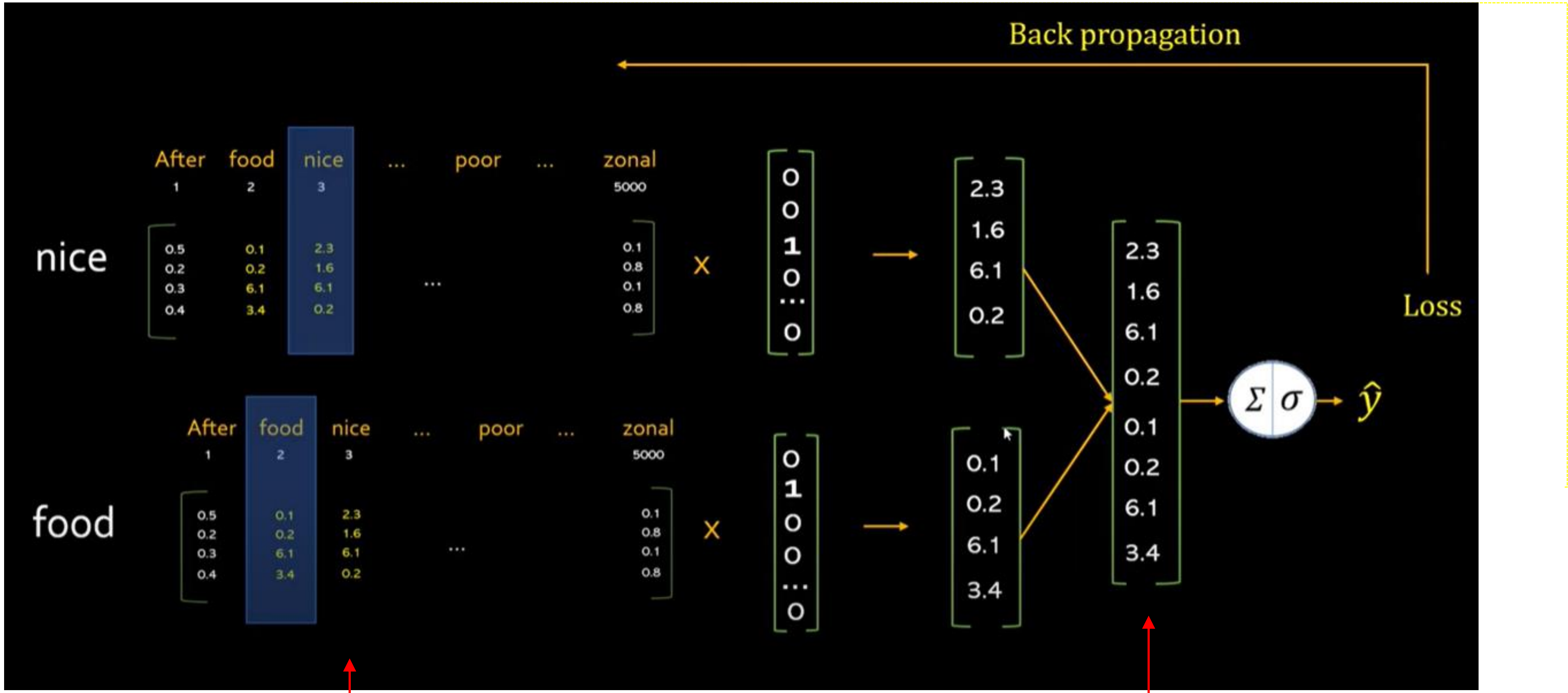
- Flexible layer
  - Can be used alone, or can be used as part of deep learning model
  - Can also be used to load a pre-trained word embedding model
- Embedding layer is defined as the first layer of a network.
- To be connected directly to Dense layer, the 2D output must be flattened to 1D.
- 3 arguments
  - input_dim: size of the vocabulary
  - output_dim: size of the vector space
  - input_length: length of input sequences (e.g. if all of your input documents are comprised of 1000 words, this would be 1000.)

input_length * word

2D vector
(one embedding per each word)

(*) The result of Embedding is just a permutation of inner trainable weights.
Where the permutation is denoted by "indices" in your input array. (see the next slide)

Inside of Keras Embedding() Layer

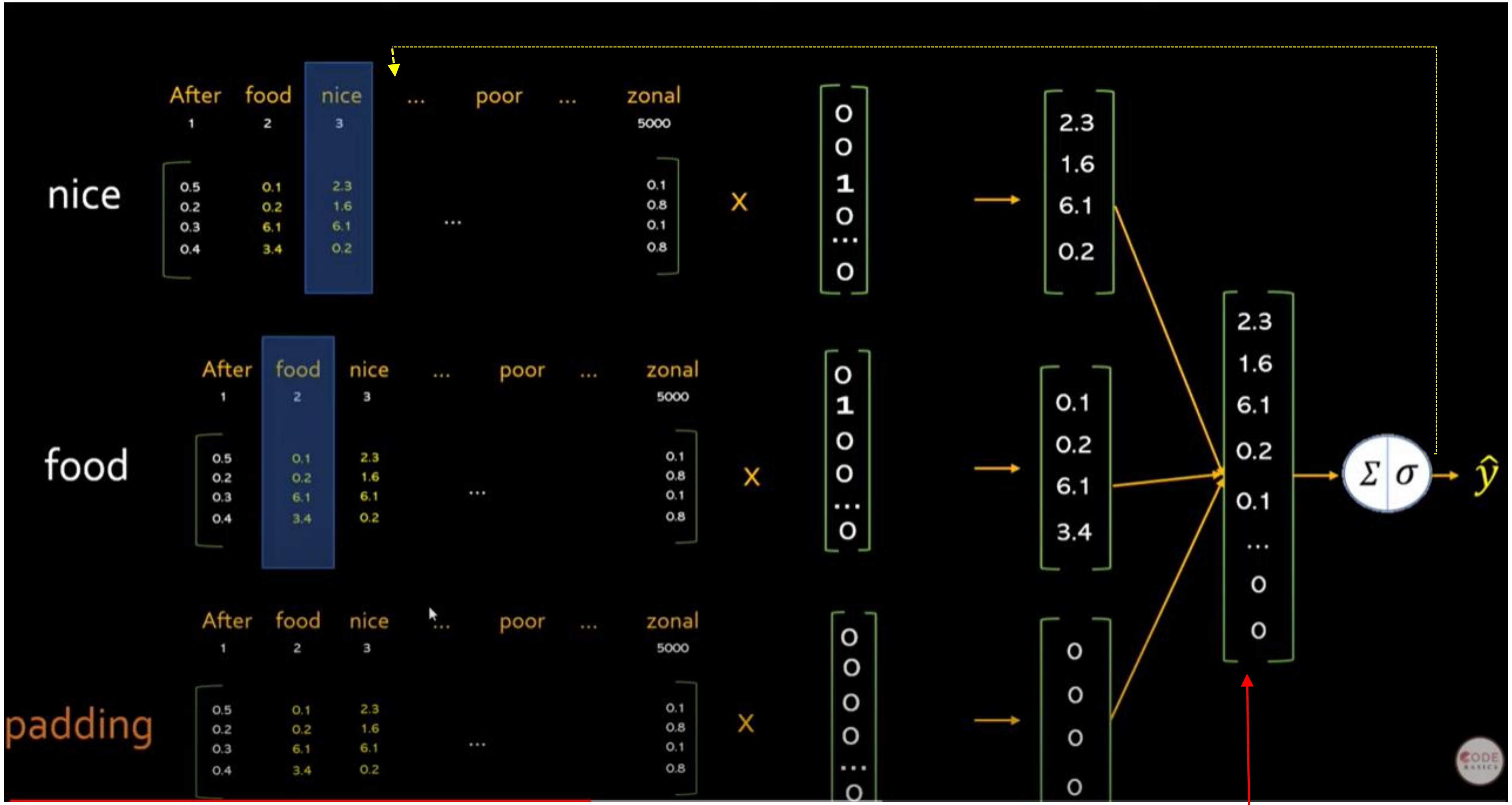


Values of the matrix are initially randomly selected, and will keep changing during the training stage

number of neurons is fixed

https://www.youtube.com/watch?v=Fuw0wv3X-0o

Inside of Keras Embedding() Layer

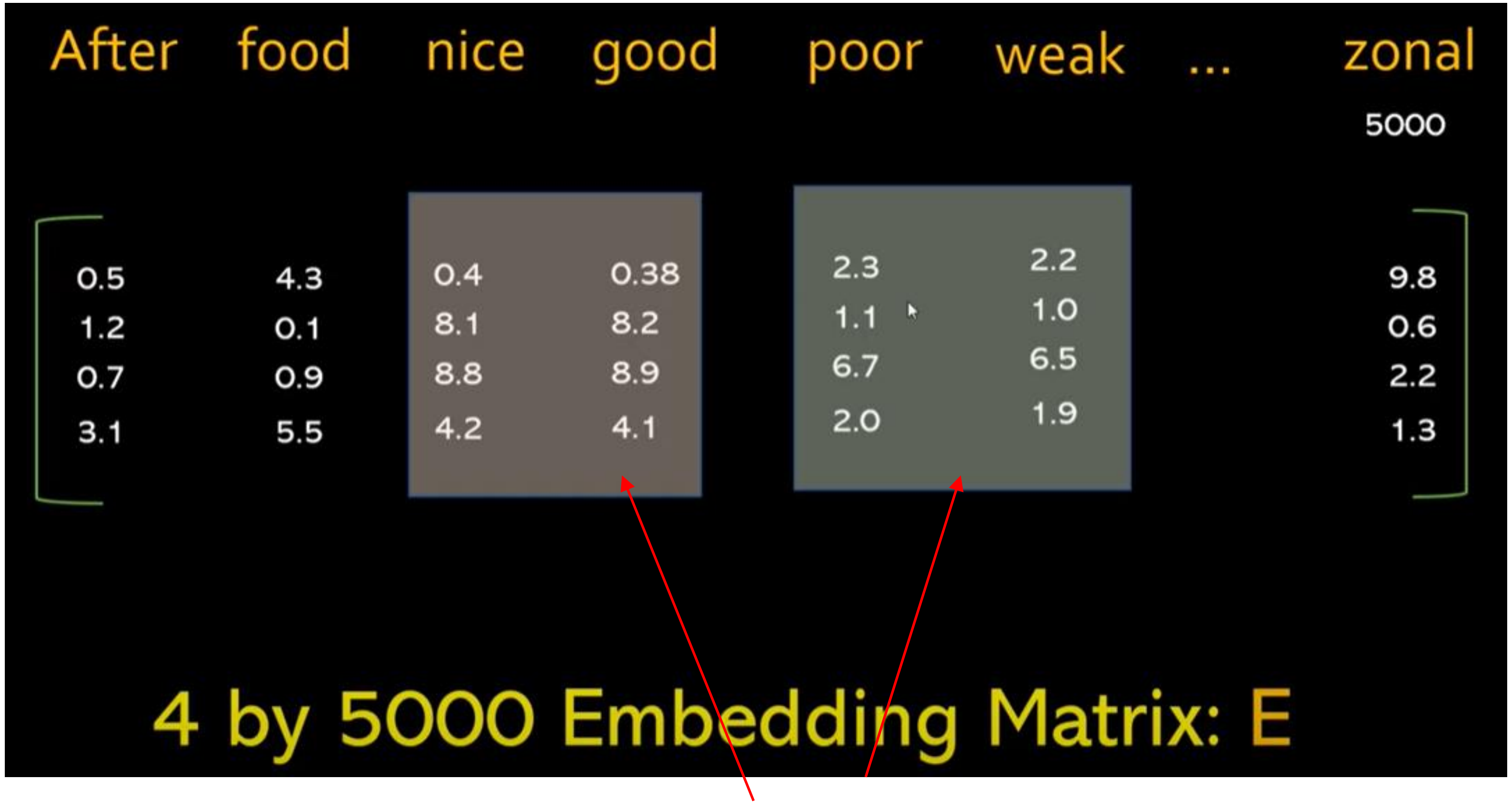


Need padding to fix the network

https://www.youtube.com/watch?v=Fuw0wv3X-0o

Inside of Keras Embedding() Layer

- **The matrix after training**



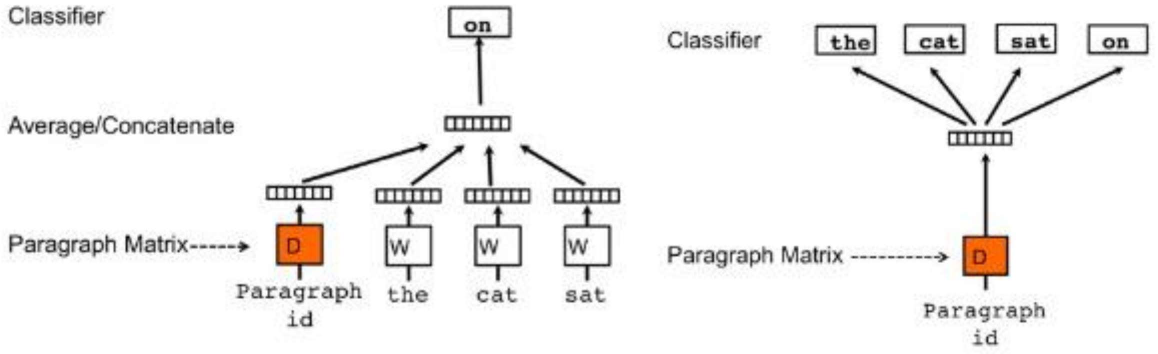
similar words are embedded similarly

Doc2Vec

- **Embedding the paragraphs (documents) to vectors**
- **Extension of Word2Vec**
- **Basic Idea**
  - act as if a document has another floating word-like vector, which contributes to all training predictions, and is updated like other word-vectors, but we will call it a doc-vector.
- **Two implementations**
  - Paragraph Vector – Distributed Memory (PV-DM)
  - Paragraph Vector – Distributed bag of Words (PV-DBOW)

Doc2Vec

- **Example: “The cat sat on the mat.” in paragrah_1**
  - (Training data)
  - window size k = 3
  - [paragraph_1, the cat, sat] – on
  - [paragraph_1, cat, sat, on] – the
  - [paragraph_1, sat, on, the] – mat
- **PY-D2V and PV-DBOW**



Paragraph vector with distributed memory (PV-DM)

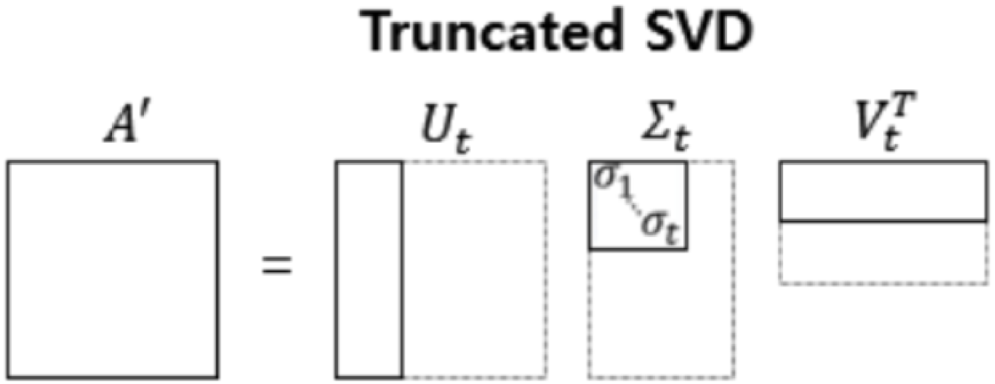
Distributed bag of words version (PVDBOW)

Topic Modeling

- **Topic Modeling:** a technique used to discover underlying topics or themes within a collection of documents.
  - It assumes that documents are mixtures of different topics, and topics are distributions over words.
  - The goal is to identify these topics and their distributions in a way that helps uncover the main themes present in the text data.
- **Latent Semantic Analysis (LSA):** LSA is a technique that uses SVD to perform topic modeling. It operates on a term-document matrix (TDM). By applying SVD to this matrix, LSA identifies latent (hidden) topics by capturing patterns of co-occurrence in the data.
- **Latent Dirichlet Allocation (LDA):** a generative probabilistic model used in natural language processing and machine learning.

Topic Modeling

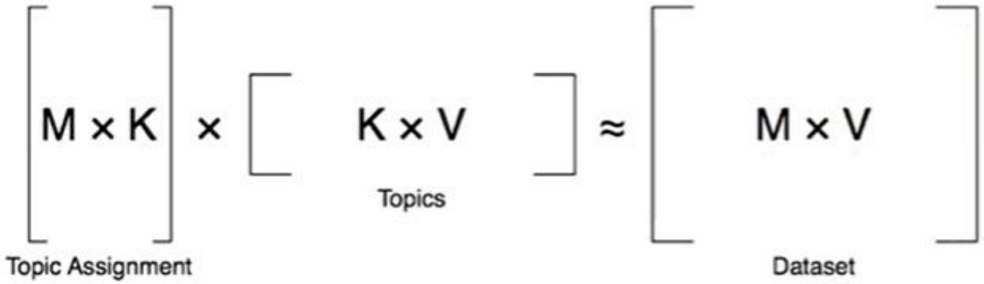
- **LSA (Latent Semantic Analysis)**
  - LDA (토픽 모델링) 에 아이디어 제공한 알고리즘
  - DTM(document-term matrix) 나 TfIdf matrix 에 Truncated SVD 이용
  - DTM을 차원 축소 하여 축소 차원에서 근접 단어들을 토픽으로 묶는다.



- **LDA (Latent Dirichlet Allocation)**
  - 각 단어가 특정 토픽에 존재할 확률과 문서에 특정 토픽이 존재할 확률을 결합확률로 추정하여 토픽을 추출하여 할당

LSA (Latent Semantic Analysis)

- **Matrix decomposition**

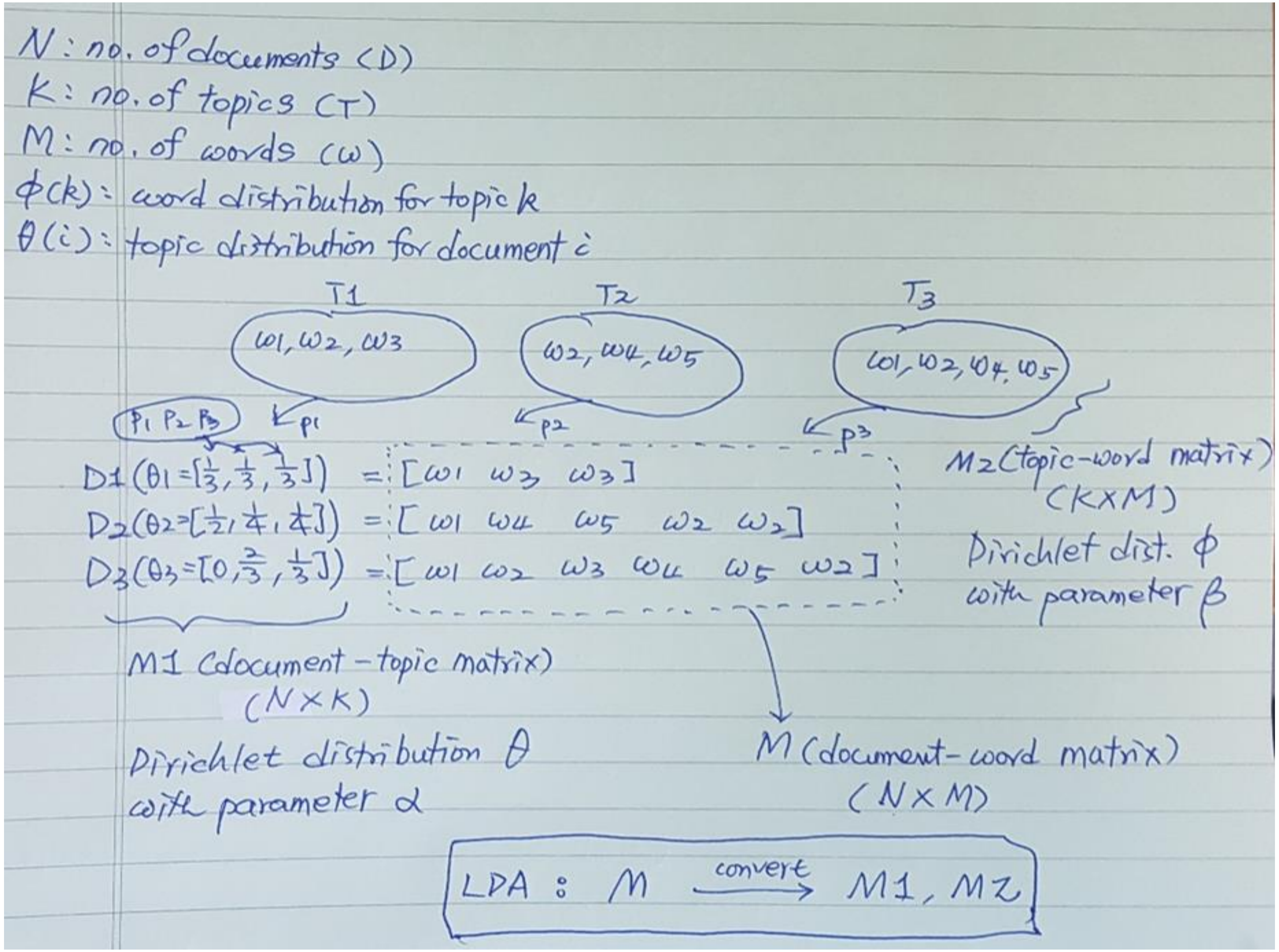


K: Number of topics
M: Number of documents
V: number of words

- **M = U * Σ * VT**
  - Each row of U can be seen as a vector that represents the document in terms of the underlying topics (This means that each document is associated with a distribution of topics, where some topics might be more prominent than others for that particular document.)
  - Each column of VT can be interpreted as a vector that represents a term's contribution to the various topics. (This means that each term is associated with a distribution of topics, indicating which topics it is most relevant to.)
  - The singular values in Σ don't have a direct semantic interpretation in terms of topics or documents. (The primary role of Σ in LSA is in the reduction of dimensions.)
  - Only the relative magnitudes of values within the matrices are important (not the absolute values). -> Larger values represent stronger associations.

LDA (Latent Dirichlet Allocation)

- Assumption:
  - The documents (D) are generated by the procedure shown in the memo.
- LDA
  - Backtracking the procedure
  - In LDA, you estimate the parameters α and β from the observed data (i.e. the documents) using techniques like Gibbs sampling.



LDA (Latent Dirichlet Allocation)

- **알파 (Alpha):**
  - 문서-토픽 분포에 대한 디리클레 분포의 파라미터
  - 각 문서가 어떤 토픽을 가질지에 대한 확률을 제어
  - 알파가 높을수록 문서는 다양한 토픽을 가질 가능성이 높아지고, 알파가 낮을수록 문서는 특정 토픽에 집중될 가능성이 높아진다
- **베타 (Beta):**
  - 토픽-단어 분포에 대한 디리클레 분포의 파라미터
  - 각 토픽이 특정 단어를 선택할 확률을 제어
  - 베타가 높을수록 각 토픽은 다양한 단어를 포함할 가능성이 높아지고, 베타가 낮을수록 토픽은 특정 단어에 집중될 가능성이 높아진다

Hyperparameters (sklearn 에서는 기본으로 symmetric 으로 설정됨 (각 문서와 토픽 분포와 각 토픽의 단어 분포가 모두 균일하다고 설정)

- **Theta(θ):**
  - 각 문서가 각 토픽을 가질 확률 분포
  - 각 문서에 대해 토픽의 분포를 표현
  - 예를 들어, θ(d)는 문서 d가 각 토픽을 가질 확률을 나타낸다
- **Phi(φ):**
  - 각 토픽이 각 단어를 가질 확률 분포
  - 각 토픽에 속하는 단어의 분포를 표현
  - 예를 들어, φ(k)는 토픽 k가 각 단어를 가질 확률을 나타낸다

LDA (Latent Dirichlet Allocation)

- Dirichlet Distribution (Probabilistic k-simplex)

$$f(x_1, \cdots, x_k; \alpha_1, \cdots, \alpha_k) = \frac{1}{\mathrm{B}(\alpha)} \prod_{i=1}^{k} x_i^{\alpha_i - 1}$$

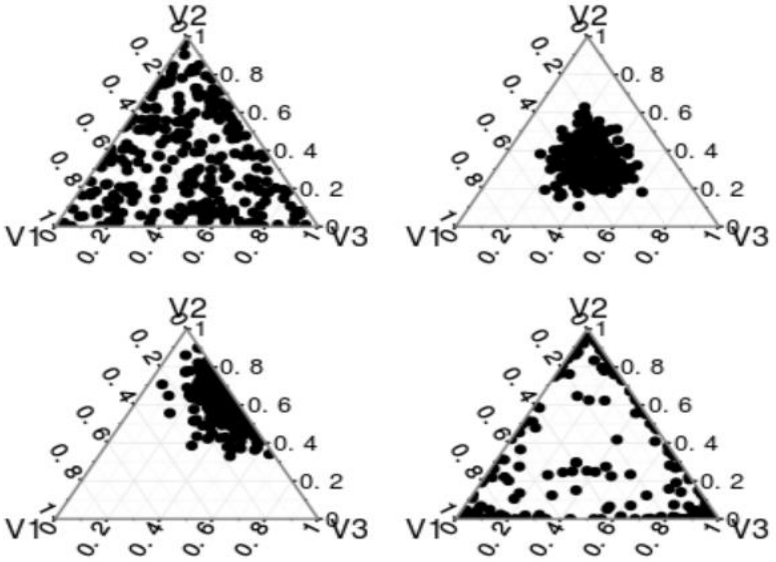
$$\mathrm{B}(\alpha) = \frac{\prod_{i=1}^{k} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{k} \alpha_i\right)} \quad (\Gamma \text{is gamma function})$$

continuous random variables

$x_k, \sum_k x_k = 1, \forall x_k \geq 0$

- Example (k=3)

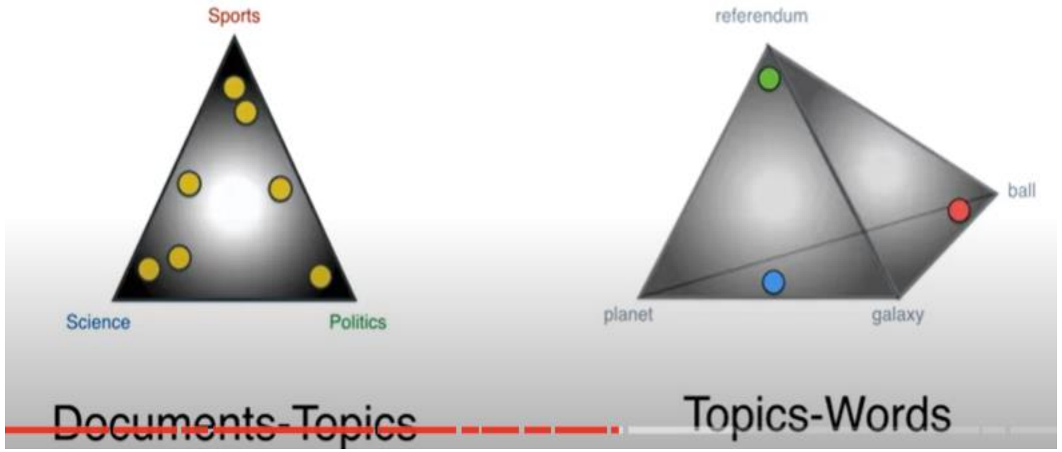
(a) $\alpha_1 = \alpha_2 = \alpha_3 = 1$
(b) $\alpha_1 = \alpha_2 = \alpha_3 = 10$
(c) $\alpha_1 = 1, \alpha_2 = 10, \alpha_3 = 5$
(d) $\alpha_1 = \alpha_2 = \alpha_3 = 0.2$



ref: https://stats.stackexchange.com/questions/244917/what-exactly-is-the-alpha-in-the-dirichlet-distribution

LDA

- **Goal:**
  - Assign an appropriate Topic to each Word
- **Properties:**
  - Each document is as monochromatic as possible
  - Each word is as monochromatic as possible
- **Probabilistic Topic model**
  - Each document is a probability distribution over topics
  - Each topic is a probability distribution over words
- **Two Dirichlet distributions**
  - (ex) 3 Topics (Red, Green, Blue)
  - 7 documents with 4 words (referendum, ball, galaxy, planet)



LDA (Latent Dirichlet Allocation)

- **Example**

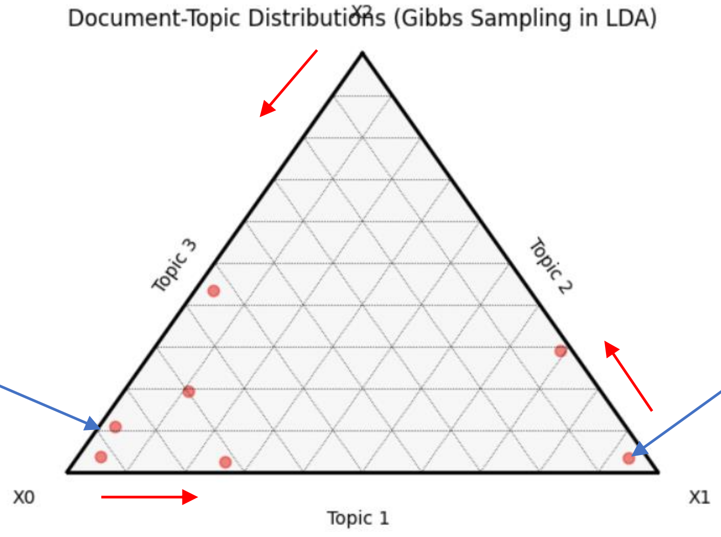
document_topic_distribution =

```
[[0.03, 0.11, 0.87],
 [0.93, 0.03, 0.03],
 [0.26, 0.02, 0.72],
 [0.03, 0.43, 0.54],
 [0.11, 0.19, 0.7 ],
 [0.69, 0.29, 0.02],
 [0.04, 0.04, 0.92]]
```

topic_word_distribution =

```
[[0.31, 0.4 , 0.01, 0.27],
 [0.02, 0.68, 0.27, 0.02],
 [0.21, 0.01, 0.36, 0.43]]
```

[0.03, 0.11, 0.87]

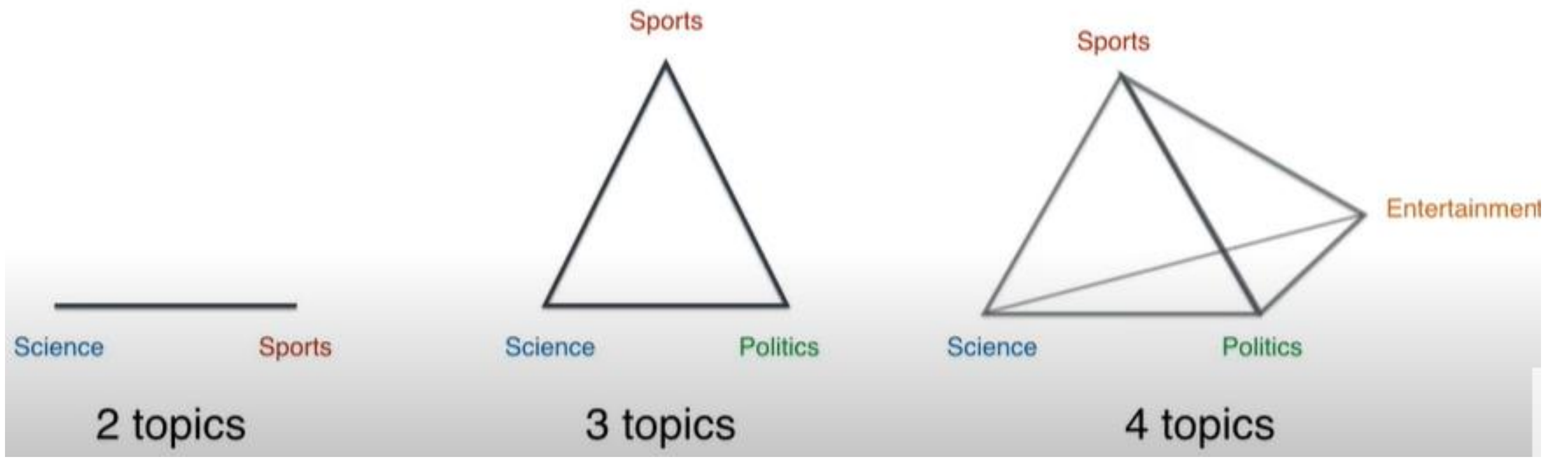


[0.93, 0.03, 0.03]

LDA (Latent Dirichlet Allocation)

- **Dirichlet Distribution**

  (from https://www.youtube.com/watch?v=BaM1uiCpj_E)



LDA

- **LDA**
  - 두 개의 Dirichlet 분포가 있으며, 이들은 각각 Document-Topic 분포(θ)와 Topic-Word 분포(Φ)를 나타낸다.
  - 이러한 두 분포는 서로 연결되어 있어 직접 샘플링하기가 어렵다. 따라서 Gibbs 샘플링 방법을 사용하여 이 분포들을 추정한다. (즉, Multivariate 문제 -> Univariate 문제로 바꾸어 해결)
- **Gibbs sampling**
  - 초기화: 각 단어에 대해 임의의 주제 할당
  - 반복적 샘플링
    - 각 단어에 대해 현재 할당된 주제를 제거하고
    - **조건부 확률을** 계산하여 새로운 주제를 샘플링
    - 이 과정을 반복하면서 Document-Topic 분포(θ) 와 Topic-Word 분포(Φ) 를 점차적으로 업데이트

조건부 확률 수식

$$P(z_{di} = k \mid \mathbf{z}_{-di}, \mathbf{w}) \propto \frac{n_{dk}^{-i}+\alpha_k}{n_d^{-i}+\sum_k \alpha_k} \cdot \frac{n_{kv}^{-i}+\beta_v}{n_k^{-i}+\sum_v \beta_v}$$

(현재 상태의 모델에서 특정 단어가 특정 주제로 할당될 확률)

- z_{di}: 문서 d의 i번째 단어에 할당된 주제.
- k: 특정 주제 (주제의 인덱스).
- w_{di}: 문서 d의 i번째 단어.
- n_{dk}^{-i}: 문서 d 내의 주제 k에 할당된 단어의 수 (현재 단어 w_{di}를 제외한).
- n_d^{-i}: 문서 d 내의 모든 단어 수 (현재 단어 w_{di}를 제외한).
- n_{kv}^{-i}: 주제 k에 할당된 단어 w의 수 (현재 단어 w_{di}를 제외한).
- n_k^{-i}: 주제 k에 할당된 모든 단어 수 (현재 단어 w_{di}를 제외한).
- α_k: 문서-주제 분포에 대한 Dirichlet 분포의 하이퍼파라미터.
- β_v: 주제-단어 분포에 대한 Dirichlet 분포의 하이퍼파라미터.

LDA: two implementations

	sklearn	gensim
Focus	a simpler implementation (primary focus is on dimensionality reduction techniques like PCA)	more specialized for natural language processing tasks, including topic modeling (core topic modeling algorithm)
Corpora and Dictionaries	requires a term-document matrix (TF-IDF matrix) as input. It doesn't include utilities for creating corpora or dictionaries.	Provides tools for creating and working with corpora and dictionaries. It uses its own internal representation called corpus and dictionary objects.
Input Data Format	term-document matrix (TF-IDF matrix)	Expects a corpus object, which is a collection of documents represented as lists of (word ID, word frequency) tuples. It also requires a dictionary mapping word IDs to words.
Model Parameters	fewer options for configuring LDA, which makes it easier for beginners	more flexible (more control over model parameters and hyperparameters)
Additional Functionality	focuses on dimensionality reduction and simpler	Offers various additional functionalities for topic modeling, such as computing topic coherence, visualization using pyLDAvis, and more.
Training	Batch approach Use standard interface: fit_transform()	On-line approach (model is trained iteratively using smaller batches of documents) When you initialize the LdaModel, it starts with random topic assignments for words. Then, you use the LdaModel's train() method to iteratively update the topic and document distributions until convergence.