

Text Processing

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

2025. 8

Yongjin Jeong, KwangWoon University

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

Contents

1. Introduction

- 텍스트 처리 응용 분야
- 핵심 용어와 주요 도구

2. 텍스트 처리 워크플로우 (The Big Picture)

- 전체 파이프라인: 데이터에서 모델까지
- 텍스트 전처리 (Preprocessing)
- 서브워드 토큰화 (BPE & WordPiece)

3. 텍스트의 수치화 (Feature Extraction)

- 단어 가방(Bag-of-Words)
- TF-IDF: 중요한 단어에 가중치 부여
- 단어 임베딩(Word Embedding)
- Word2Vec: 문맥으로 단어를 학습
- Keras Embedding Layer: 딥러닝 임베딩

4. 현대 자연어 처리 (Modern NLP)

- 문맥을 이해하는 임베딩의 필요성
- 트랜스포머와 BERT: NLP의 새로운 판도

5. 모델링 및 평가

- 모델 학습 및 성능 평가

6. 심화 주제: 토픽 모델링 (Advanced Topic)

- 문서의 숨겨진 주제를 찾는 법: LSA와 LDA

Introduction to Text Processing

Text Processing

- **What is Text Processing?**

- 텍스트 처리(또는 자연어 처리, NLP: Natural Language Translation)는 컴퓨터가 인간의 언어를 이해하고, 해석하며, 생성할 수 있도록 하는 기술

- **Applications**

- 스팸 필터링 (Spam Filtering): 메일이 스팸인지 아닌지 분류
- 감성 분석 (Sentiment Analysis): 영화 리뷰나 소셜 미디어 텍스트의 긍정/부정을 판단
- 기계 번역 (Machine Translation): 한 언어를 다른 언어로 자동 번역한다.
- 정보 추출 및 Q&A: 챗봇, AI 스피커처럼 문서에서 질문에 대한 답을 찾아낸다.

핵심 용어 및 도구

- **핵심 용어**

- 말뭉치 (Corpus): 분석할 텍스트 데이터의 집합 (예: 영화 리뷰 데이터셋)
- 토큰 (Token): 텍스트를 나누는 기본 단위. 단어, 문자, 형태소 등이 될 수 있다.
- 불용어 (Stop words): 분석에 큰 의미가 없는 단어들 (예: 'a', 'the', 'is', 조사 '은/는/이/가').

- **주요 파이썬 라이브러리**

- NLTK(natural Language Toolkit): 영어 자연어 처리를 위한 전통적인 종합 도구 (from tokenization, stemming, tagging, parsing, and beyond)
- KoNLPy(NLP library for Korean Language): 한국어 형태소 분석 등 한국어 처리에 필수적인 도구.
- Scikit-learn: TDM, TF-IDF, 머신러닝 모델 구현에 사용.
- Gensim: Word2Vec, 토픽 모델링 등 다양한 모델링 기능을 제공.
- TensorFlow/Keras: 딥러닝 모델과 Embedding Layer 구현에 사용.
- Transformers: 최신 NLP 모델(BERT, GPT 등)을 쉽게 사용할 수 있는 허깅페이스 라이브러리.

Text Processing – 전체 파이프라인

- 1단계: **텍스트 전처리 (Text Preprocessing)**:
 - 분석에 용이하도록 텍스트를 정제하고 나누는 과정
 - (예: 문장 -> 단어 토큰화, 불용어 제거)
- 2단계: **특징 추출 (Feature Extraction)**
 - 정제된 텍스트(단어)를 컴퓨터가 이해할 수 있는 숫자 벡터로 변환
 - (예: BoW, TF-IDF, Word Embedding)
- 3단계: **모델링 (Modeling)**
 - 숫자 벡터를 입력받아 예측을 수행하는 머신러닝/딥러닝 모델을 학습
- 4단계: **평가 (Evaluation)**
 - 만들어진 모델이 얼마나 정확하게 예측하는지 성능을 측정.

Tokenization and Preprocessing

Text Preprocessing (텍스트 전처리)

- **Step 1: 텍스트를 깨끗하게 다듬기:**
 - 분석의 품질을 높이기 위해 원시 텍스트(Raw Text)에 포함된 노이즈를 제거하는 과정
- **주요 전처리 단계**
 - **토큰화 (Tokenization):** 문장을 단어 또는 형태소 단위로 나눈다.
 - **정제 (Cleaning):** 구두점, HTML 태그 등 불필요한 문자를 제거한다.
 - **불용어 제거 (Stopword Removal):** 분석에 불필요한 단어를 목록에서 제외한다.
 - **정규화 (Normalization):** 단어를 표준 형태로 변환한다.
 - **어간 추출 (Stemming):** 단어의 어간을 추출 (예: studying -> studi).
 - **표제어 추출 (Lemmatization):** 단어의 원형을 추출 (예: is, are -> be).

Tokenization

- **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)**

```
1 from nltk.tokenize import word_tokenize
2
3 text = "After sleeping for four hours, he decided to sleep for another four"
4 tokens = word_tokenize(text)
5 print(tokens)
```

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

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

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

Tokenization

- 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)

```
1 from konlpy.tag import Okt
2 okt=Okt()
3
4 print(okt.morphs("우리가 이 과제를 잘 할 수 있을까?"))
5 print(okt.pos("우리가 이 과제를 잘 할 수 있을까?", norm=True, stem=True))
6 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']
```

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

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

Subword Tokenization

- **Tokenizer 의 진화 (단어와 형태소의 절충점) – Subword Tokenization**
 - 최신 언어 모델(BERT, GPT 등)은 단어나 형태소 단위가 아닌, Subword로 텍스트를 분리한다.
- **어휘 외 단어(Out-of-Vocabulary, OOV) 문제 해결:**
 - 단어 토큰화는 사전에 없는 단어(신조어, 오타 등)를 만나면 의미를 파악하지 못하고 <UNK> (Unknown) 토큰으로 처리한다.
 - Subword는 처음 보는 단어도 의미 있는 조각(Subword)들로 분해하여 의미를 유추할 수 있다.
 - (예) jetpackification → jet, pack, ification
- **어휘 사전 크기 관리:**
 - 모든 단어를 사전에 등록하면 어휘 사전이 비대해진다.
 - 자주 쓰이는 단어는 통째로, 드물게 쓰이는 단어는 Subword로 분리하여 저장함으로써 효율적으로 어휘 사전을 관리.
- **주요 알고리즘:**
 - BPE (Byte-Pair Encoding): GPT 계열에서 주로 사용.
 - WordPiece: BERT에서 주로 사용.
 - SentencePiece: 구현한 위 알고리즘들을 구글의 라이브러리.

Subword Tokenization (예)

```
1 # transformers 라이브러리 설치
2 !pip install transformers
3
4 from transformers import BertTokenizer
5
6 # 'bert-base-uncased' 모델이 사용한 WordPiece 토크나이즈를 로드한다.
7 tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
8 # 일반 단어(tokenization)와 oov 단어(huggingfaceization)가 포함된 예시 문장
9 text = "Modern NLP uses subword tokenization. Let's try huggingfaceization."
10
11 # 토큰화 실행
12 tokens = tokenizer.tokenize(text)
13 print(tokens)
```

['modern', 'nl', '##p', 'uses', 'sub', '##word', 'token', '##ization', '.', 'let', "'", 's', 'try', 'hugging', '##face', '##ization', '.']

- tokenization → ['token', '##ization'] : 사전에 있는 단어도 더 기본적인 단위로 분리
- huggingfaceization → ['hugging', '##face', '##ization'] : 사전에 없는 신조어임에도 불구하고, 의미 있는 단위(hugging, face)로 분해하여 <UNK> 토큰이 되는 것을 방지.
- ##의 의미: WordPiece 토크나이즈는 단어의 시작 부분이 아닌, 앞 토큰에 이어지는 subword라는 것을 표시하기 위해 ## 접두사를 붙인다. 이를 통해 나중에 원래 단어로 쉽게 복원할 수 있다.

Feature Extraction

Feature Extraction: Bag-of-Words (단어가방)

- **Step 2A: 텍스트를 숫자로 표현하는 가장 간단한 방법:**
- **Bag-of-Words (BoW)란?**
 - 문장의 구조나 단어의 순서를 무시하고, 단순히 단어들의 등장 횟수(빈도)만으로 텍스트를 표현하는 방법.
 - 단어들을 순서 없이 가방에 넣어 담는다는 의미.
- **BoW 표현 과정**
 - 전체 말뭉치에서 고유한 단어 사전을 만든다.
 - 각 문서를 사전의 단어들이 몇 번 등장했는지 카운트하여 벡터로 표현한다. (Document-Term Matrix: DTM)
- **명확한 한계**
 - 단어 순서 무시: "나는 너를 좋아해"와 "너는 나를 좋아해"를 비슷하게 처리할 수 있다.
 - 희소성 (Sparsity): 단어 사전에 수만 개 단어가 있다면, 대부분의 값이 0인 거대한 벡터가 생성된다.
 - 의미 정보 부족: 단어의 문맥적 의미를 전혀 반영하지 못한다.

TF-IDF: 중요한 단어에 가중치 부여

- 모든 단어가 똑같이 중요할까? TF-IDF
- **핵심 아이디어:** 단순히 단어의 빈도만 세는 것(BoW)을 넘어, 문서 내에서 얼마나 중요한지를 가중치로 표현한다.
- **TF (Term Frequency, 단어 빈도)**
 - 한 문서 안에서 특정 단어가 얼마나 자주 등장하는가. (TF가 높을수록 문서 내에서 중요한 단어)
- **IDF (Inverse Document Frequency, 역문서 빈도)**
 - 특정 단어가 전체 문서들 중에서 얼마나 희귀하게 등장하는가. (모든 문서에 등장하는 'the' 같은 단어는 IDF가 낮고, 특정 문서에만 나오는 단어는 IDF가 높다)

$$idf_j = \log\left(\frac{\#documents}{\#documents\ with\ word\ j}\right)$$

- **TF-IDF 계산: $TF-IDF = TF \times IDF$**
 - 한 문서에 자주 등장하면서(TF 높음), 다른 문서에는 잘 안 나오는(IDF 높음) 단어일수록 높은 값을 가진다.

Libraries for BoW and TF-IDF

- **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 (DTM)
- **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()`

BoW (예)

- 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"

- **Document-Term Matrix (DTM)**

$$X = \begin{matrix} & \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & & \text{approach} \\ \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \dots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} & \begin{matrix} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{matrix} \end{matrix}$$

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

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

DTM and TFIDF

$$X = \begin{matrix} & \text{the} & \text{is} & \text{of} & \text{goal} & \text{lecture} & \text{bag} & \text{words} & \text{via} & \text{text} & \text{approach} \\ \begin{matrix} \text{Document 1} \\ \text{Document 2} \\ \text{Document 3} \end{matrix} & \begin{bmatrix} 2 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \dots & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix} \quad \rightarrow \quad X = \begin{bmatrix} 0.8 & 0.4 & 0 & 1.1 \\ 0.4 & 0.4 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \end{bmatrix}$$

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

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

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

[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: $\text{idf}(t) = \log \left[\frac{(1 + n)}{(1 + \text{df}(t))} \right] + 1$

Word E,mbedding

Feature Extraction: Word Embedding

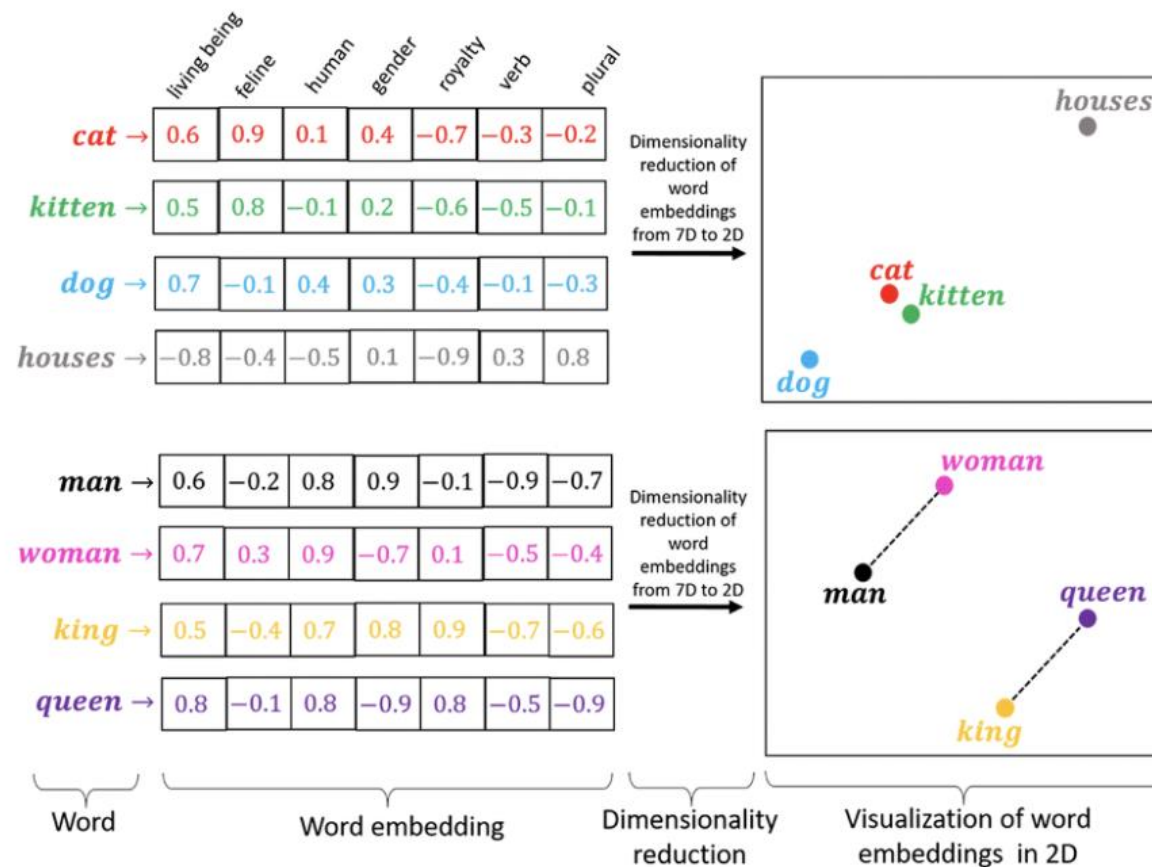
- Step 2B: 단어의 '의미'를 벡터 공간에 담는다.
- Word Embedding이란?
 - 단어를 고차원의 밀집 벡터(Dense Vector)로 표현하는 기술.
 - BoW의 희소 벡터(Sparse Vector)와 달리, 적은 차원(예: 100~300차원)에 의미를 압축한다.
- 핵심 철학: 분포 가설 (Distributional Hypothesis)
 - "비슷한 문맥에서 등장하는 단어는 비슷한 의미를 가진다."
 - 이 가설을 바탕으로, 벡터 공간에서 의미가 비슷한 단어들을 가까이 위치시킨다.
- 장점:
 - 단어 간의 의미적, 문법적 관계를 벡터 연산으로 파악할 수 있다.
 - BoW보다 훨씬 적은 차원으로 풍부한 정보를 표현한다.

Word Embedding (예)

- **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

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import SimpleRNN, Dense, LSTM, GRU
3
4 model = Sequential()
5 model.add(Embedding(input_dim=10000, output_dim=8,
6 | | | | | input_length=maxlen))
7 model.add(LSTM(128))
8 model.add(Flatten())
9 model.add(Dense(1, activation='sigmoid'))
```



Word Analogies

- Interesting result

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

a:b :: c:?



$$d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|}$$

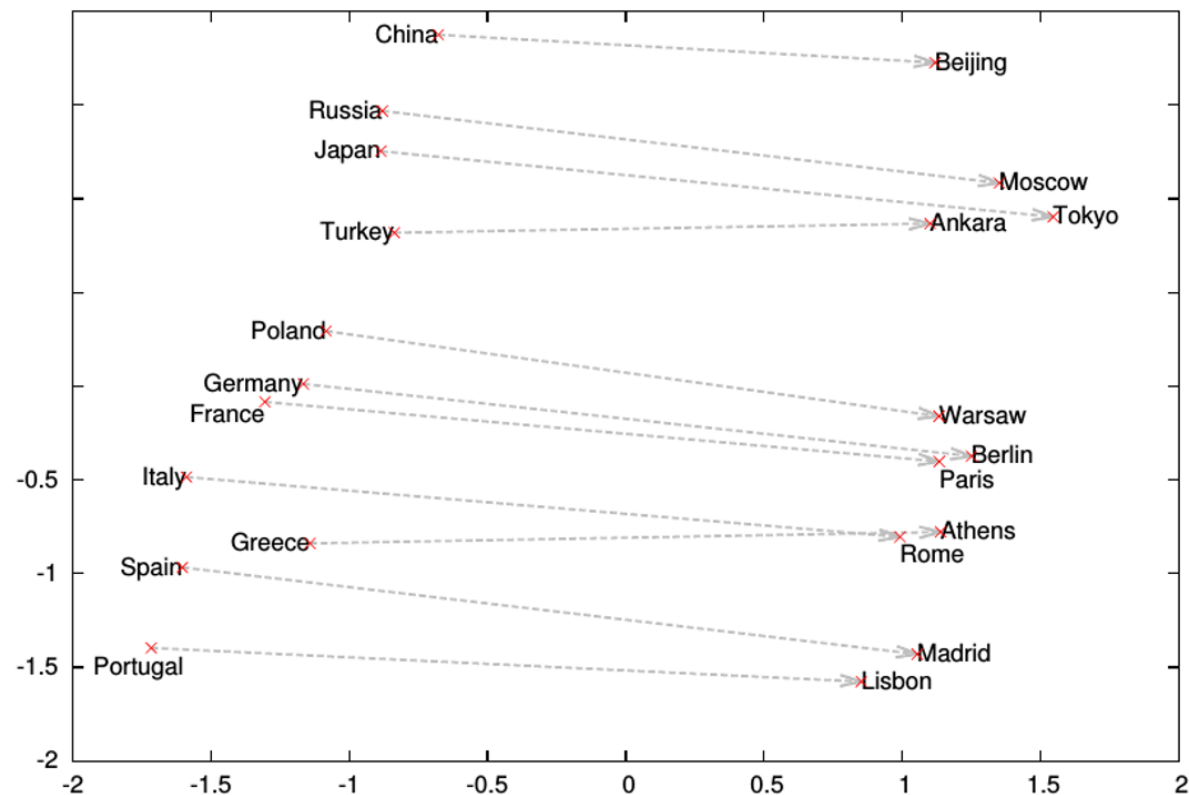
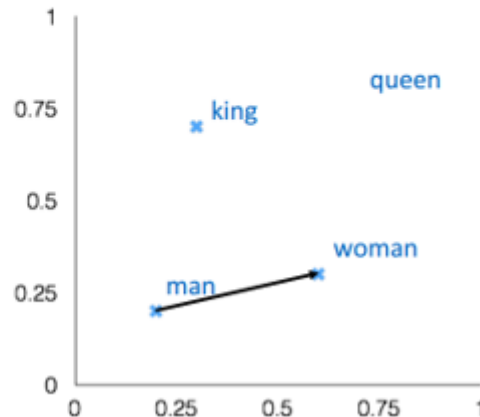
man:woman :: king:?

+ king [0.30 0.70]

- man [0.20 0.20]

+ woman [0.60 0.30]

queen [0.70 0.80]

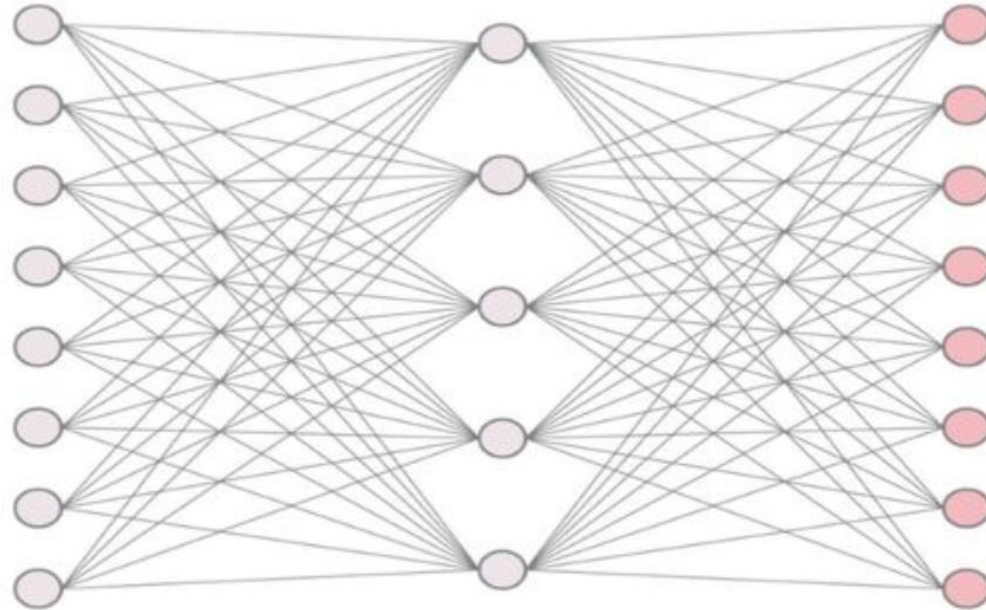


Word Embedding – Word2Vec

- **Word2Vec**: 문맥을 기반으로 단어의 벡터 표현을 학습하는 신경망 모델.
- **CBOW (Continuous Bag-of-Words)**
 - 주변 단어들(문맥)을 가지고 중심 단어를 예측하는 모델.
- **Skip-Gram**
 - 중심 단어를 가지고 주변 단어들을 예측하는 모델.
 - 일반적으로 CBOW보다 성능이 좋다고 알려져 있다.
- **핵심 원리 (간략화)**
 - 얇은 신경망을 학습하는 과정에서, Input Layer와 Hidden Layer 사이의 가중치 행렬 (Weight Matrix)이 업데이트된다.
 - 학습이 완료된 후, 이 가중치 행렬이 바로 우리가 원하는 Word Embedding이 된다.
 - Negative Sampling: Softmax 계산 비용을 줄여 학습 속도를 높이는 핵심 기술.

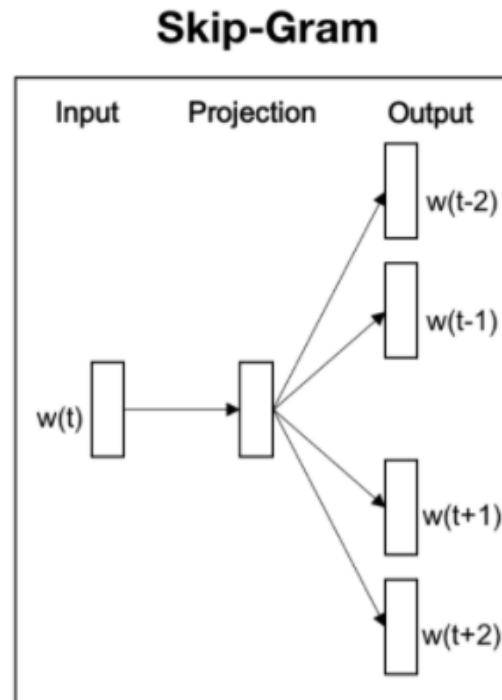
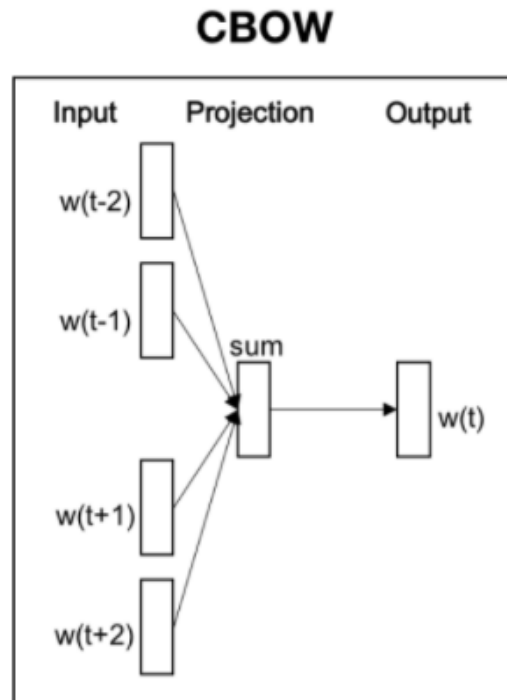
Word2Vec

- 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)



Word2Vec - CBOW

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

center word context words

I like playing football with my friends

I like playing football with my friends

I like playing football with my friends

I like playing football with my friends

I like playing football with my friends

I like playing football with my friends

I like playing football with my friends

y	X
center word	context words
[1,0,0,0,0,0,0]	[0,1,0,0,0,0,0] [0,0,1,0,0,0,0]
[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]
[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]
[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]
[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]
[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]
[0,0,0,0,0,0,1]	[0,0,0,0,1,0,0] [0,0,0,0,0,1,0]

Training samples

(I, like)
(I, playing)

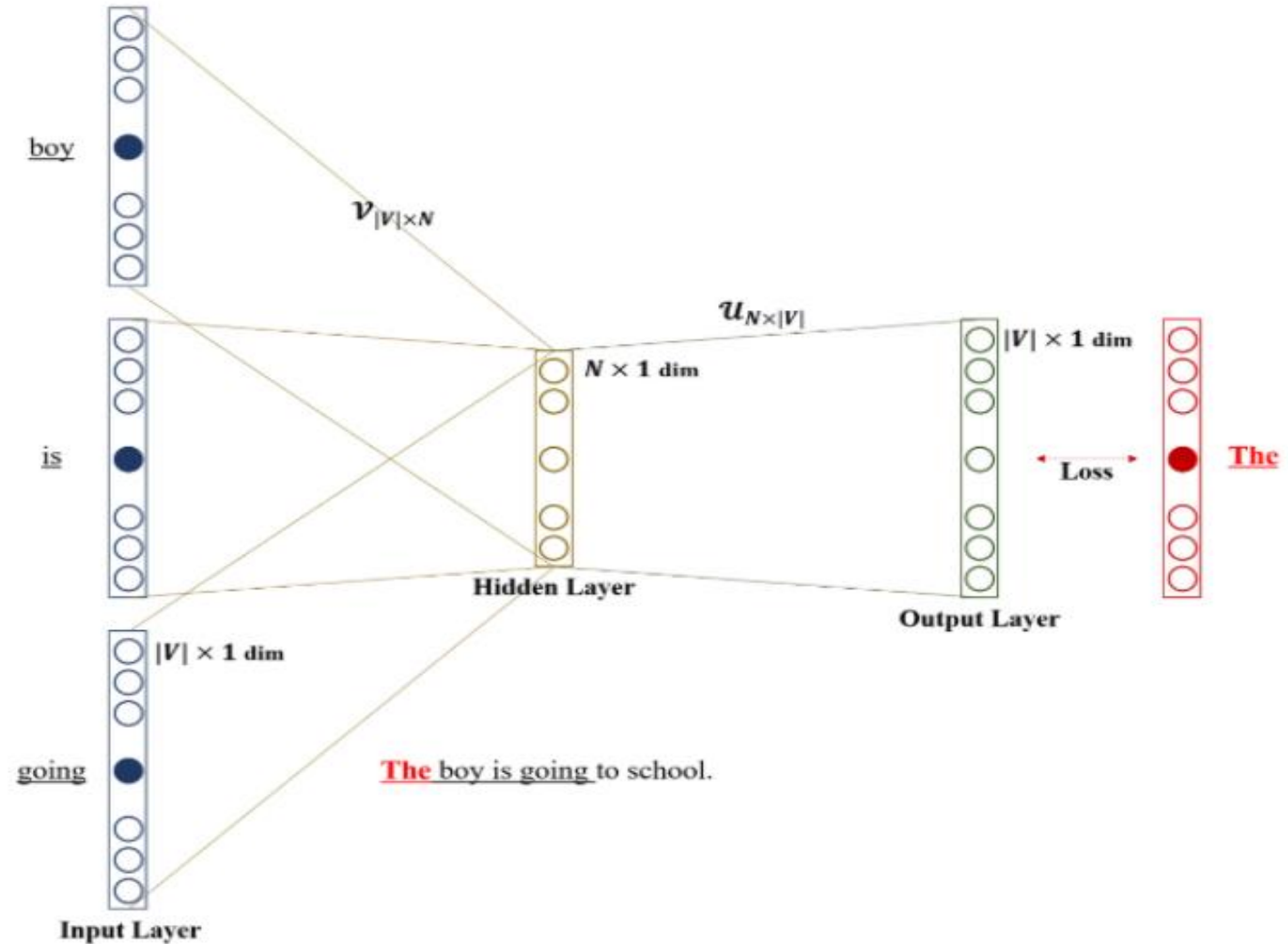
(like, I)
(like, playing)
(like, football)

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

..
..
..
..
..
..
..
..
..
..

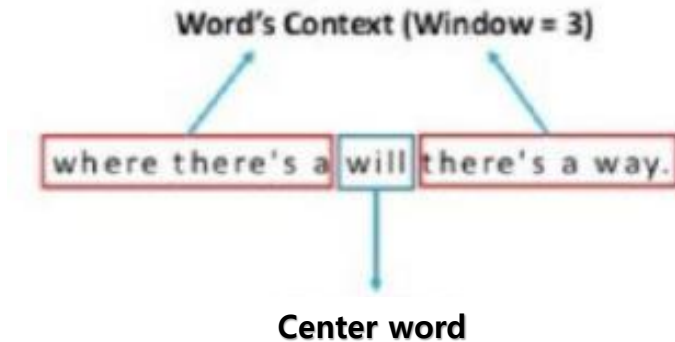
Word2Vec - CBOW

- CBOW architecture

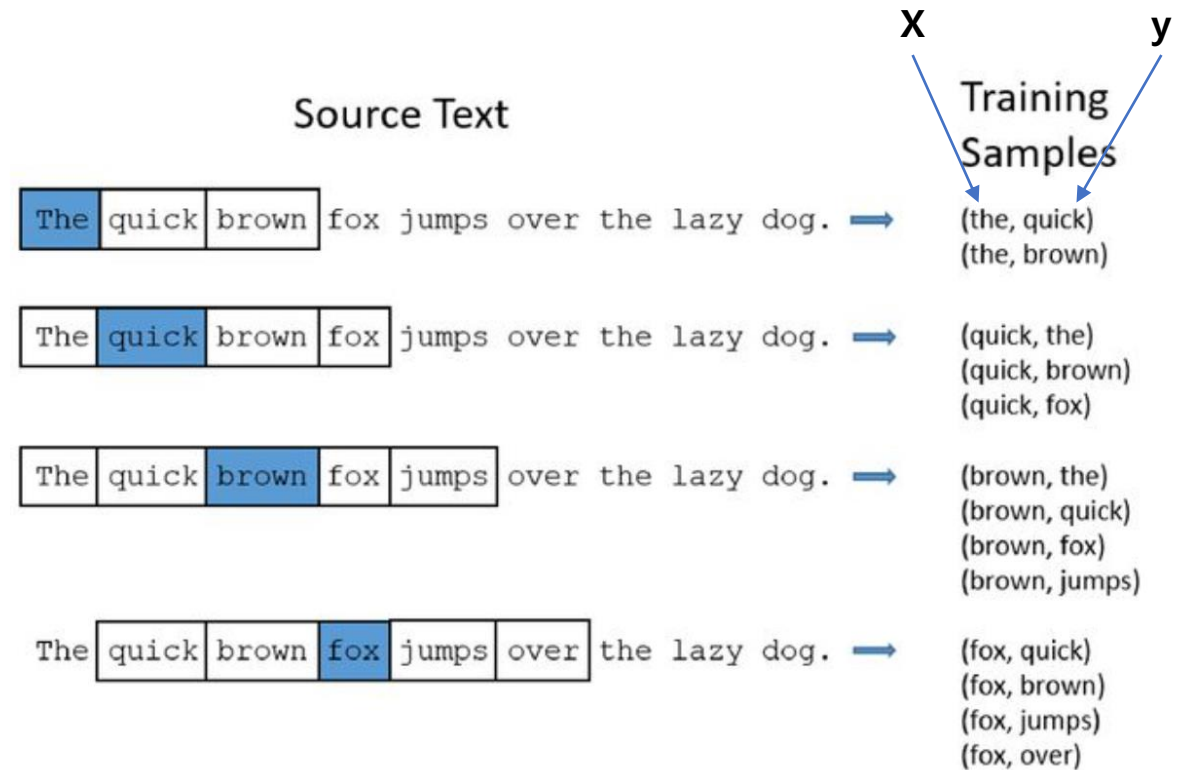


Word2Vec – Skip Gram

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

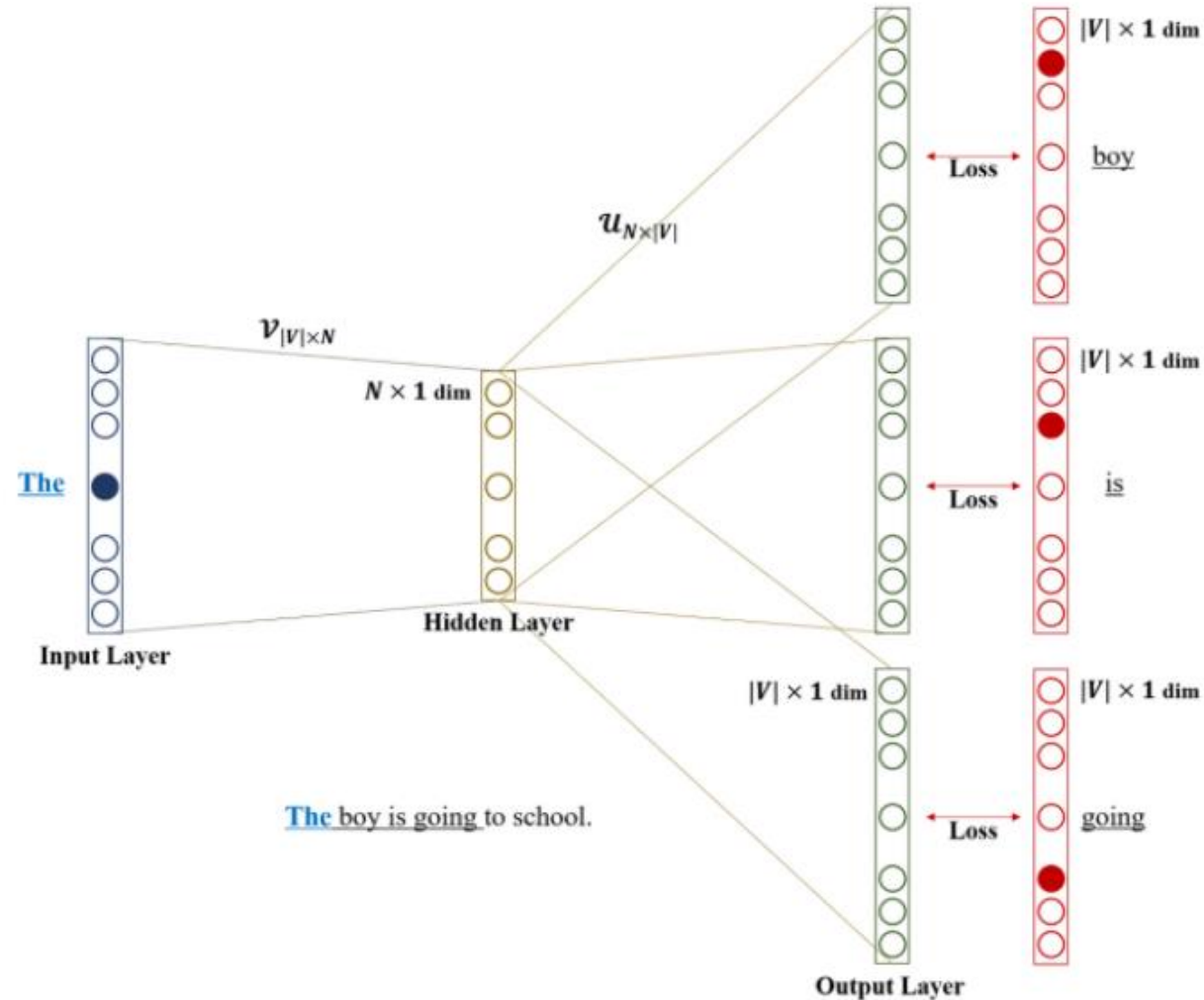


- **Collection of Training samples**
 - with a window of size 2



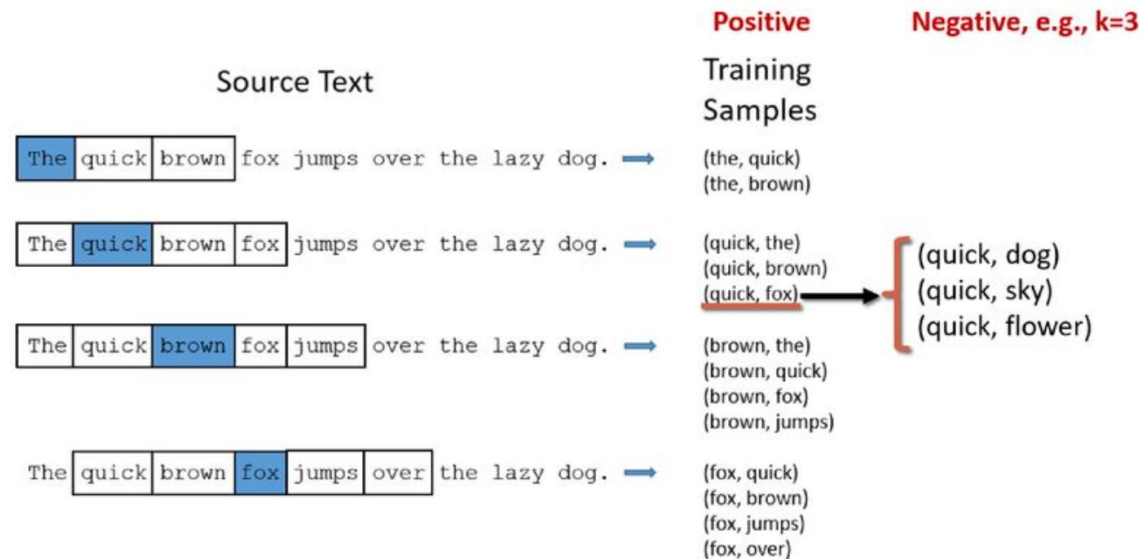
Word2Vec – Skip Gram

- Skip-Gram architecture



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)



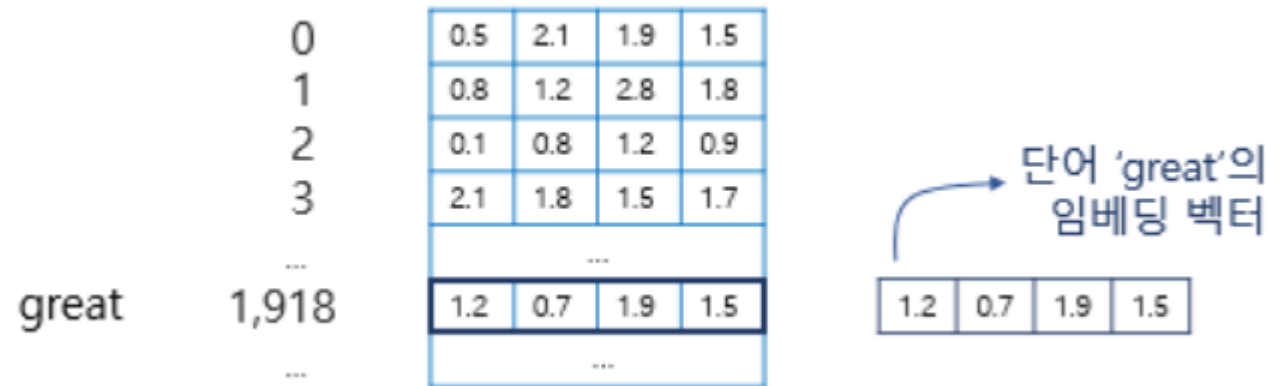
Feature Extraction: Keras Embedding Layer

- **Embedding Layer:** 딥러닝 모델을 이용해 임베딩 구현
- 딥러닝 모델 내에서 단어 임베딩을 처리하는 전용 레이어 (맨 첫 레이어에 위치)
- **작동 방식:** 거대한 조회 테이블(Look-up Table)처럼 작동
 - 단어를 정수 인덱스로 변환한다 (예: great -> 1918).
 - Embedding 레이어는 해당 인덱스에 맞는 임베딩 벡터를 테이블에서 찾아 반환한다.
 - Word → Integer → Look-up Table → Embedding vector
- **특징**
 - 사전 학습된 임베딩 로드: Word2Vec 등으로 미리 학습된 벡터를 불러와 사용할 수 있다.
 - 직접 학습 (Train from scratch): 모델의 다른 부분과 함께 임베딩 벡터를 특정 과제에 맞게 처음부터 학습할 수도 있다.

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.

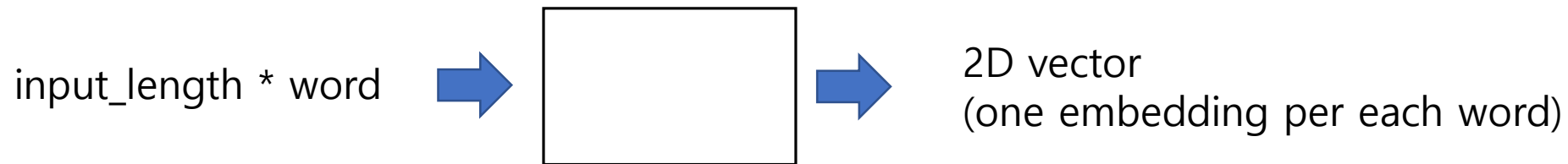
Word → Integer → lookup Table → Embedding vector



훈련 과정에서 학습된다.

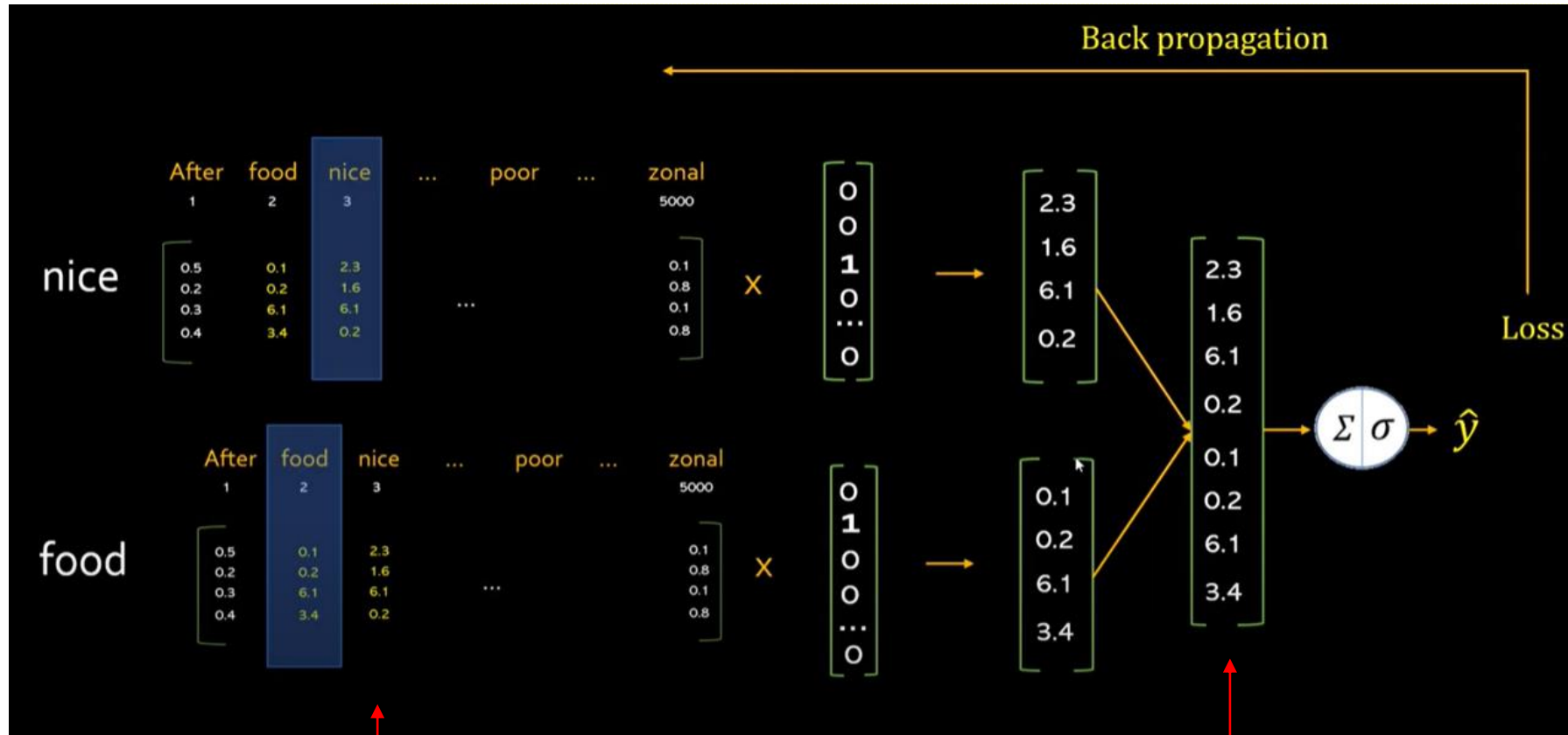
Keras Embedding() Layer

- **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.)



(*) 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

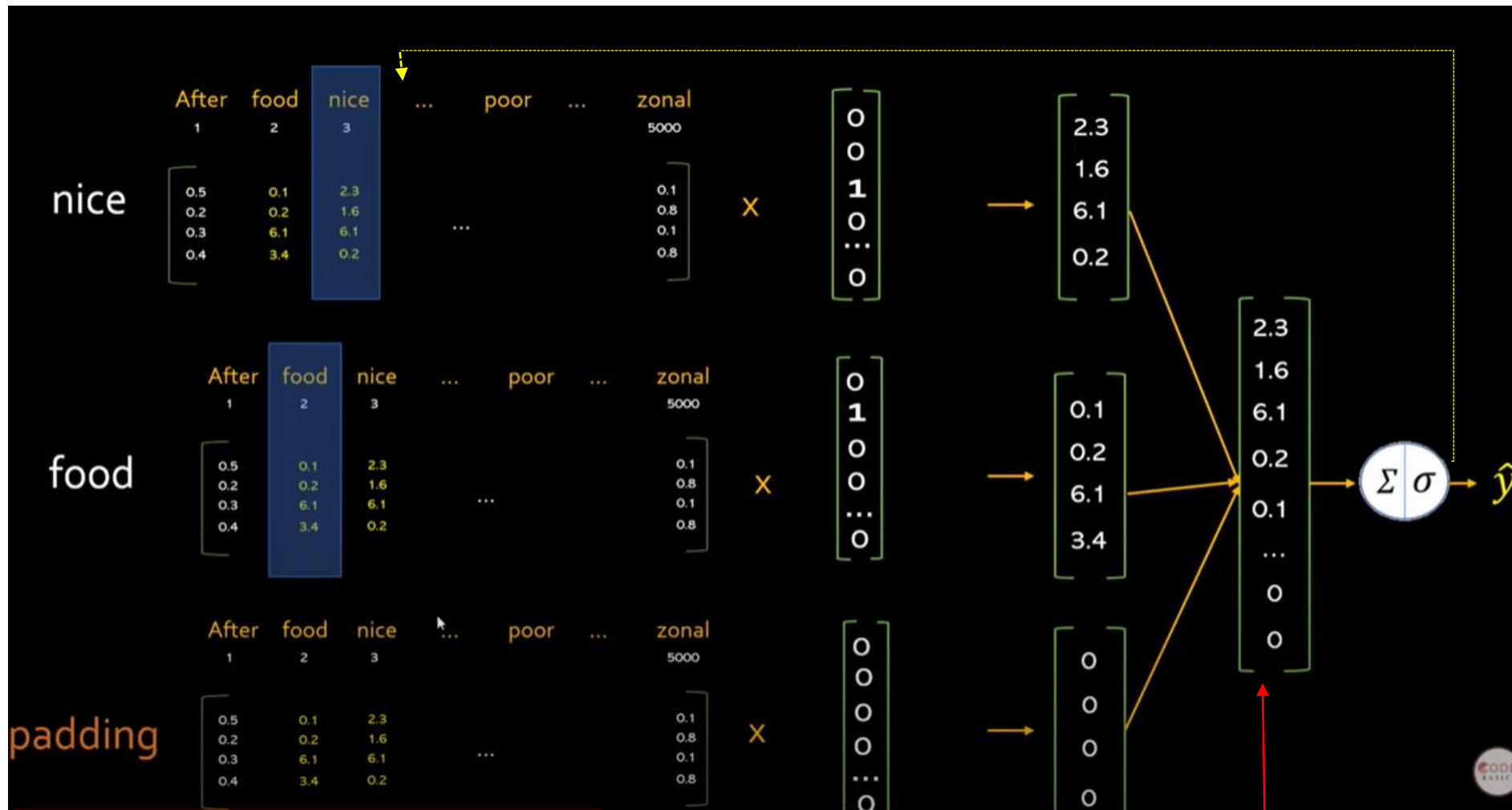


Values of the matrix are initially randomly selected, and will keep changing during the training stage

number of neurons is fixed

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

Inside of Keras Embedding() Layer



Need padding to fix the network

Inside of Keras Embedding() Layer

- The matrix after training



similar words are embedded similarly

Contextual Embedding

- Word2Vec의 한계와 새로운 질문 - 문맥을 이해하는 임베딩의 필요성
- Word2Vec의 결정적 한계: 동음이의어(Homonym, Polysemy) 문제
 - Word2Vec은 단어 하나에 단 하나의 벡터만 할당한다. 따라서 아래 두 문장에서 'bank'는 완전히 동일한 벡터로 표현된다. (Static Word Embedding)
 - "I need to go to the bank to deposit money." (은행)
 - "He sat on the river bank." (강둑)
 - 즉, 문맥에 따라 달라지는 단어의 의미를 포착하지 못한다.
- 문장 전체의 문맥을 고려하여 단어의 의미를 동적으로 결정할 수는 없을까?
 - -> **문맥 기반 임베딩 (Contextual Embedding)** 등장 (같은 단어라도 문맥에 따라 전혀 다른 값을 가지게 됨)

Transformer and BERT

- **현대 NLP의 심장: 트랜스포머와 BERT**
- 트랜스포머 (Transformer) 아키텍처
 - 2017년 구글이 발표한 모델로, Self-Attention이라는 메커니즘을 사용.
 - Self-Attention: 문장 내 단어들의 관계를 파악하여, 어떤 단어가 다른 단어에 더 중요한지를 동적으로 계산. (예: "The animal didn't cross the street because it was too tired."에서 'it'이 'animal'을 가리킨다는 것을 높은 가중치로 학습)
- BERT (Bidirectional Encoder Representations from Transformers)
 - 트랜스포머의 인코더 구조를 활용한 사전 학습 언어 모델(Pre-trained Language Model)
 - 양방향(Bidirectional): 단어의 의미를 파악하기 위해 문장의 왼쪽과 오른쪽 문맥을 동시에 고려.
 - Word2Vec과 달리, BERT는 문장이 주어지면 각 단어에 문맥을 반영한 동적인 임베딩 벡터를 생성.

Machine Learning Model

- **Step 3: 모델 학습 (Model Training)**

- 텍스트에서 추출한 숫자 벡터(TF-IDF, Embedding Vector)를 모델의 입력으로 사용.
- 전통적 머신러닝: TF-IDF 벡터 + 로지스틱 회귀, SVM 등.
- 딥러닝: Embedding Vector + RNN, LSTM, 또는 BERT 같은 트랜스포머 기반 모델

- **Step 4: 성능 평가 (Performance Evaluation)**

- 학습된 모델이 얼마나 정확하게 예측하는지 객관적인 지표로 확인.
- 주요 분류 평가 지표
 - 정확도 (Accuracy): 전체 예측 중 정답의 비율. 가장 직관적이지만 데이터가 불균형할 때 왜곡될 수 있다.
 - 정밀도 (Precision): 모델이 'Positive'로 예측한 것 중 실제 'Positive'의 비율. (스팸 메일 분류에서 중요)
 - 재현율 (Recall): 실제 'Positive'인 것 중 모델이 'Positive'로 예측한 비율. (암 진단 등에서 중요)
 - F1-Score: 정밀도와 재현율의 조화 평균.

Topic Modeling

Advanced Topic: Topic Modeling

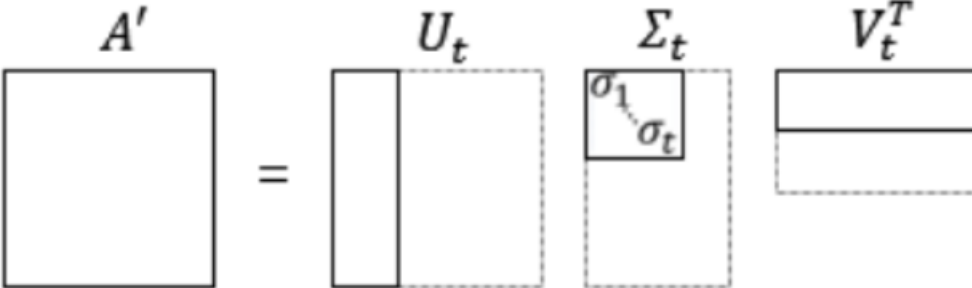
- 토픽 모델링이란?
 - 대규모 문서 집합에서 숨겨진(Latent) 주제나 테마를 자동으로 발견하는 비지도 학습 기법.
 - 예: 뉴스 기사 문치에서 '스포츠', '정치', '경제' 등의 주제를 찾아내는 것.
- 주요 알고리즘
 - LSA (Latent Semantic Analysis, 잠재 의미 분석)
 - DTM이나 TF-IDF 행렬에 특이값 분해(SVD)를 적용하여 차원을 축소하고, 이 축소된 차원을 **잠재적 토픽**으로 간주한다.
 - LDA (Latent Dirichlet Allocation, 잠재 디리클레 할당)
 - 문서가 여러 토픽의 혼합으로, 토픽은 여러 단어의 혼합으로 구성된다고 가정하는 확률적 생성 모델.
 - "각 문서가 어떤 토픽 분포를 가지며, 각 토픽이 어떤 단어 분포를 가지는지"를 추정한다

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.

Truncated SVD

$$A' = U_t \Sigma_t V_t^T$$


- **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

$$\begin{array}{c} \left[\begin{array}{c} M \times K \end{array} \right] \\ \text{Topic Assignment} \end{array} \times \begin{array}{c} \left[\begin{array}{c} K \times V \end{array} \right] \\ \text{Topics} \end{array} \approx \begin{array}{c} \left[\begin{array}{c} M \times V \end{array} \right] \\ \text{Dataset} \end{array}$$

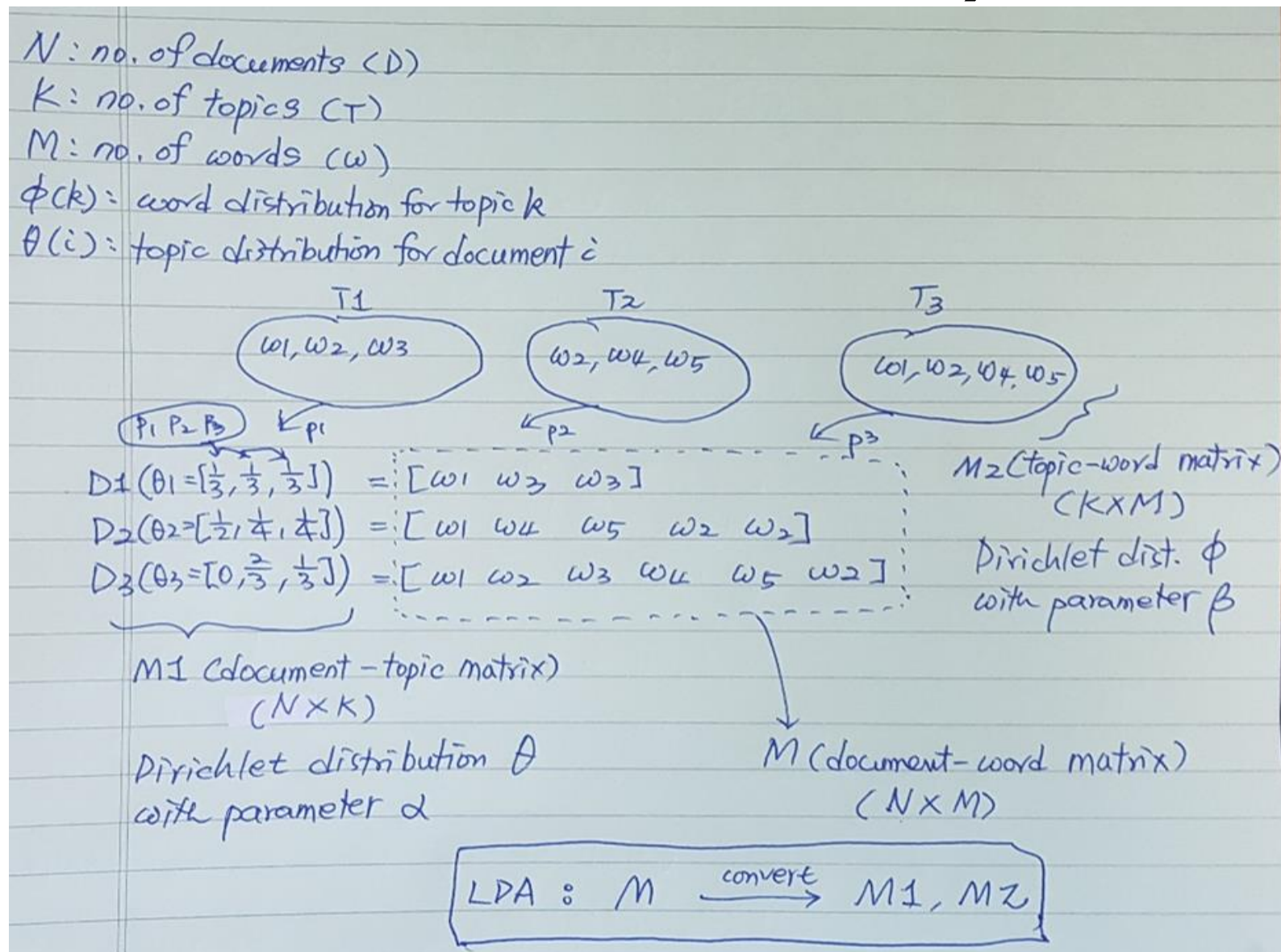
K: Number of topics
M: Number of documents
V: number of words

- $M = U * \Sigma * VT$

- 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 (latent) **topic assignments** (topic-word distributions and document-topic distributions) using techniques like Gibbs sampling.
 - α and β : Hyper-parameters



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

- **Theta(θ):**

- 각 문서가 각 토픽을 가질 확률 분포
- 각 문서에 대해 토픽의 분포를 표현
- 예를 들어, $\theta(d)$ 는 문서 d 가 각 토픽을 가질 확률을 나타낸다

- **Phi(ϕ):**

- 각 토픽이 각 단어를 가질 확률 분포
- 각 토픽에 속하는 단어의 분포를 표현
- 예를 들어, $\phi(k)$ 는 토픽 k 가 각 단어를 가질 확률을 나타낸다

Hyperparameters
(sklearn, gensim 에서
는 기본으로 symmetric
으로 설정됨 (각 문서의
토픽 분포와 각 토픽의
단어 분포가 모두 균일
하다고 설정)

Sklearn:

- doc_topic_prior, topic_word_prior

Gensim:

- alpha, eta

LDA (Latent Dirichlet Allocation)

- Dirichlet Distribution (Probabilistic k-simplex)

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

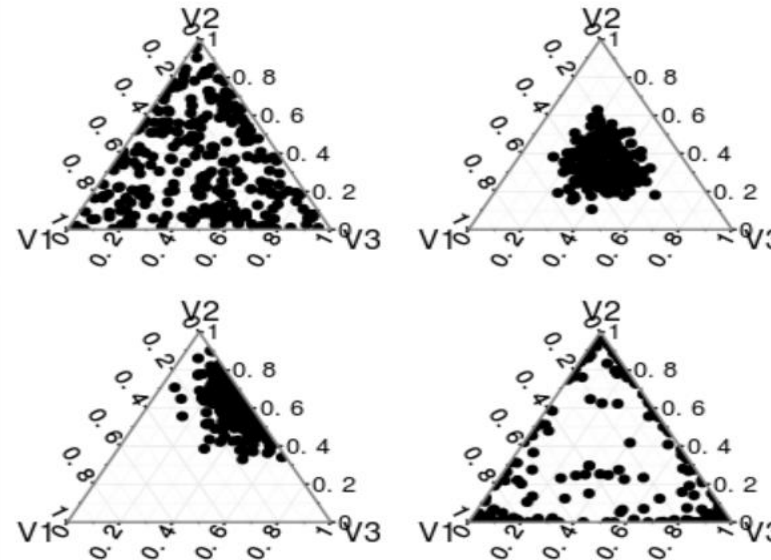
$$B(\alpha) = \frac{\prod_{i=1}^k \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^k \alpha_i)} \quad (\Gamma \text{ is gamma function})$$

continuous random variables

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

- Example (k=3)

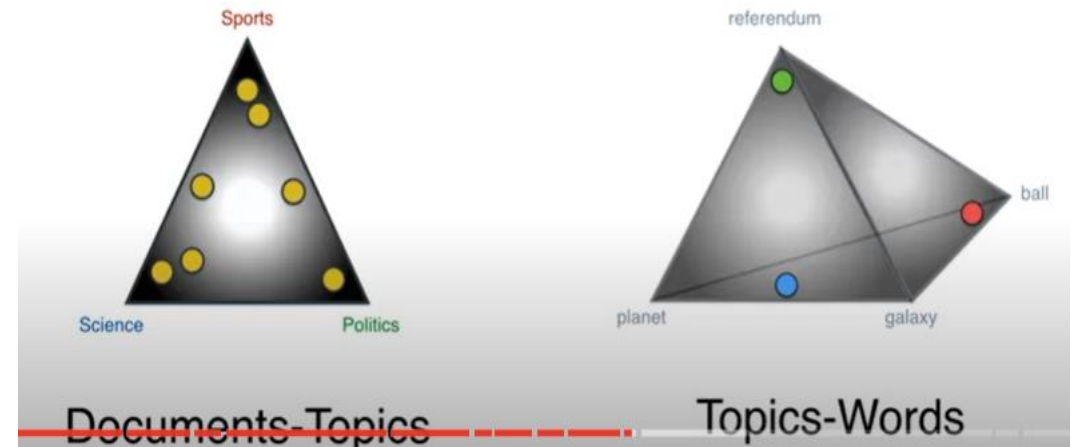
- (a) $\alpha_1 = \alpha_2 = \alpha_3 = 1$
- (b) $\alpha_1 = \alpha_2 = \alpha_3 = 10$
- (c) $\alpha_1 = 1, \alpha_2 = 10, \alpha_3 = 5$
- (d) $\alpha_1 = \alpha_2 = \alpha_3 = 0.2$



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)

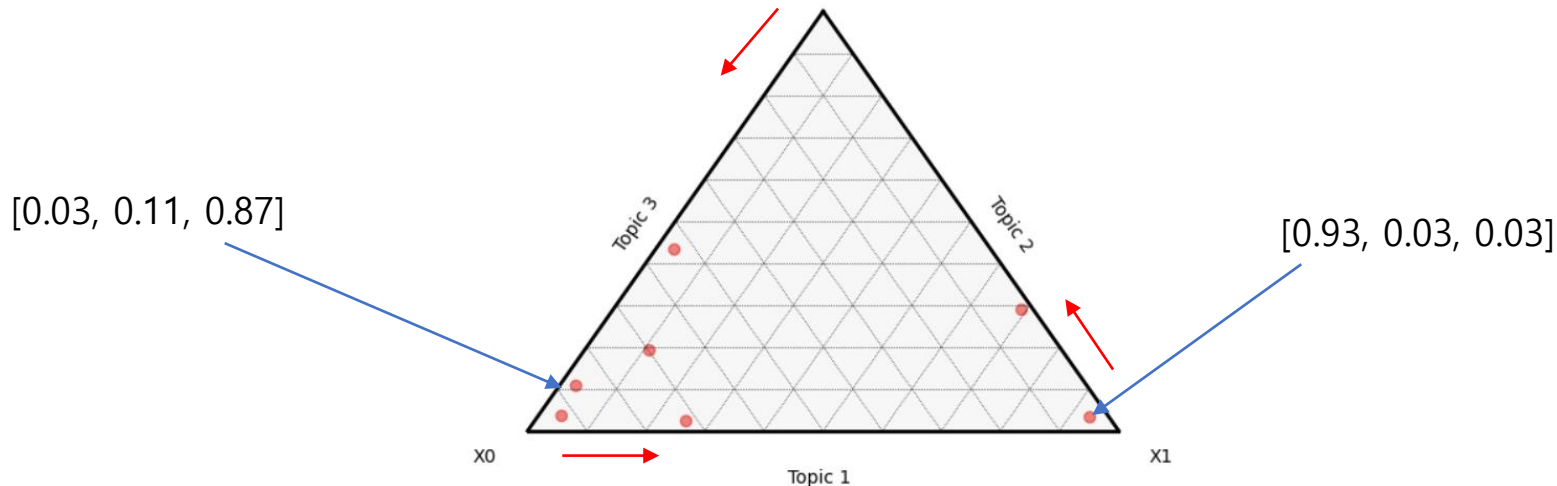
- Example

document_topic_distribution = $\begin{bmatrix} [0.03, 0.11, 0.87], \\ [0.93, 0.03, 0.03], \\ [0.26, 0.02, 0.72], \\ [0.03, 0.43, 0.54], \\ [0.11, 0.19, 0.7], \\ [0.69, 0.29, 0.02], \\ [0.04, 0.04, 0.92] \end{bmatrix}$

topic_word_distribution = $\begin{bmatrix} [0.31, 0.4 , 0.01, 0.27], \\ [0.02, 0.68, 0.27, 0.02], \\ [0.21, 0.01, 0.36, 0.43] \end{bmatrix}$



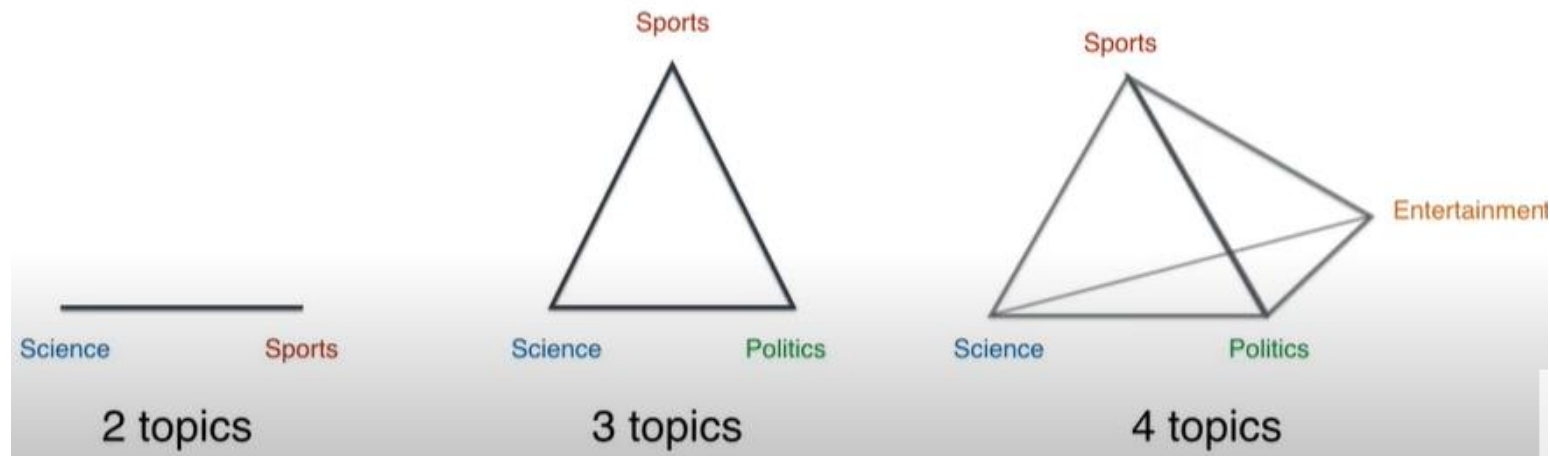
Document-Topic Distributions (Gibbs Sampling in LDA)



LDA (Latent Dirichlet Allocation)

- Dirichlet Distribution

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



LDA

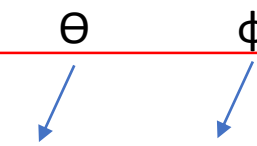
- **LDA**

- 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 확률**을 계산하여 새로운 주제를 샘플링
 - 이 과정을 반복하면서 Document-Topic 분포(θ)와 Topic-Word 분포(ϕ)를 점차적으로 업데이트

θ ϕ



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 d 의 i 번째 단어에 할당된 주제.
- k : 특정 주제 (주제의 인덱스).
- w_{di} : 문서 d 의 i 번째 단어.
- n_{dk}^{-i} : 문서 d 내의 주제 k 에 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 k 에 할당된 단어 w 의 수 (현재 단어 w_{di} 를 제외한).
- n_k^{-i} : 주제 k 에 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: <code>fit_transform()</code>	On-line approach (model is trained iteratively using smaller batches of documents) When you initialize the <code>LdaModel</code> , it starts with random topic assignments for words. Then, you use the <code>LdaModel</code> 's <code>train()</code> method to iteratively update the topic and document distributions until convergence.

LSA (Example)

📖 예시 데이터 (3문서, 단어 사전 8개)

- 문서1: "코스피 지수 상승 투자"
- 문서2: "정부 선거 국회 정책"
- 문서3: "선수 경기 골 팀"

👉 단어 사전 = [코스피, 지수, 투자, 정부, 선거, 국회, 선수, 경기] (총 8개)

(다른 단어는 불용어 처리로 가정)

(*) LSA의 doc-topic matrix: 좌표값 → 단독으로는 해석이 어렵고, 문서 간 관계 분석에 활용 (개별 값 자체는 큰 의미가 없음)

- 문서 간 유사도 계산 (코사인 유사도)
- 문서 클러스터링
- 차원 축소 후 시각화 등에 활용

◆ LSA 결과 (SVD 기반, k=2 축으로 축소)

doc_topic_matrix (문서 × 축):

코드 ^

```
[[ 0.92  0.10] # 문서1
 [-0.88  0.05] # 문서2
 [ 0.05  0.95]] # 문서3
```

topic_word_matrix (축 × 단어):

코드 ^

```
[[ 0.70  0.65  0.68 -0.72 -0.70 -0.68  0.05  0.02] # 축1
 [ 0.05  0.02  0.01  0.02  0.01  0.01  0.80  0.75]] # 축2
```

- 축1: 코스피, 지수, 투자 (+) vs 정부, 선거, 국회 (-)
- 축2: 선수, 경기 (+)

(*) 양수/음수 방향이 그대로 "경제 vs 정치" 축, "스포츠" 축으로 해석

LDA (Example)

📖 예시 데이터 (3문서, 단어 사전 8개)

- 문서1: "코스피 지수 상승 투자"
- 문서2: "정부 선거 국회 정책"
- 문서3: "선수 경기 골 팀"

👉 단어 사전 = [코스피, 지수, 투자, 정부, 선거, 국회, 선수, 경기] (총 8개)

(다른 단어는 불용어 처리로 가정)

(*) LDA의 doc-topic matrix: 확률 분포 → "문서의 주제 비율"로 해석 가능
(직접적인 의미가 있음)

- 문서 분류 (예: 경제 기사로 분류)
- 문서 검색 (특정 주제와 가까운 문서 찾기)
- 문서 클러스터링

◆ LDA 결과 (확률 기반, k=3 토픽)

doc_topic_matrix (문서 × 토픽):

코드 ^

```
[[0.90 0.05 0.05] # 문서1
 [0.05 0.92 0.03] # 문서2
 [0.05 0.08 0.87]] # 문서3
```

topic_word_matrix (토픽 × 단어):

코드 ^

```
[[0.20 0.18 0.15 0.01 0.01 0.01 0.01 0.01] # 토픽1 (경제)
 [0.01 0.01 0.01 0.20 0.18 0.15 0.01 0.01] # 토픽2 (정치)
 [0.01 0.01 0.01 0.01 0.01 0.01 0.20 0.18]] # 토픽3 (스포츠)
```

- 토픽1: 코스피, 지수, 투자 확률 높음 → "경제 주제"
- 토픽2: 정부, 선거, 국회 확률 높음 → "정치 주제"
- 토픽3: 선수, 경기 확률 높음 → "스포츠 주제"

(*) 확률 분포로 주제가 정의되고, 음수는 없다.